SAM4E Series

Atmel | SMART ARM-based Flash MCU

DATASHEET

Description

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

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

The peripheral set includes a full-speed USB device port with embedded transceiver, a 10/100 Mbps Ethernet MAC supporting IEEE 1588, a high-speed MCI for SDIO/SD/MMC, an external bus interface featuring a static memory controller providing connection to SRAM, PSRAM, NOR Flash, LCD Module and NAND Flash, a parallel I/O capture mode for camera interface, hardware acceleration for AES256, 2 USARTs, 2 UARTs, 2 TWIs, 3 SPIs, as well as a 4-channel PWM, 3 three-channel general-purpose 32-bit timers (with stepper motor and quadrature decoder logic support), a low-power RTC, a low-power RTT, 256-bit General Purpose Backup Registers, 2 Analog Front End interfaces (16-bit ADC, DAC, MUX and PGA), one 12-bit DAC (2-channel) and an analog comparator.

The SAM4E 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.

The Real-time Event Managment allows peripherals to receive, react to and send events in Active and Sleep modes without processor intervention.

1. Features

- Core
  - ARM Cortex-M4 with 2 Kbytes Cache running at up to 120 MHz[1]
  - Memory Protection Unit (MPU)
  - DSP Instruction
  - Floating Point Unit (FPU)
  - Thumb®-2 Instruction Set
- Memories
  - Up to 1024 Kbytes Embedded Flash
  - 128 Kbytes Embedded SRAM
  - 16 Kbytes ROM with Embedded Boot Loader Routines (UART) and IAP Routines
  - Static Memory Controller (SMC): SRAM, NOR, NAND Support
  - NAND Flash Controller
- System
  - Embedded Voltage Regulator for Single Supply Operation
  - Power-on-Reset (POR), Brown-out Detector (BOD) and Dual Watchdog for Safe Operation
  - Quartz or Ceramic Resonator Oscillators: 3 to 20 MHz Main Power with Failure Detection and Optional Low-power 32.768 kHz for RTC or Device Clock
  - RTC with Gregorian and Persian Calendar Mode, Waveform Generation in Backup mode
  - RTC counter calibration circuitry compensates for 32.768 kHz crystal frequency inaccuracy
  - High Precision 4/8/12 MHz Factory Trimmed Internal RC Oscillator with 4 MHz Default Frequency for Device Startup. In-application Trimming Access for Frequency Adjustment
  - Slow Clock Internal RC Oscillator as Permanent Low-power Mode Device Clock
  - One PLL up to 240 MHz for Device Clock and for USB
  - Temperature Sensor
  - Low-power tamper detection on two inputs, anti-tampering by immediate clear of general-purpose backup registers (GPBR)
  - Up to 2 Peripheral DMA Controllers (PDC) with up to 33 Channels
  - One 4-channel DMA Controller
- Low-power Modes
  - Sleep, Wait and Backup modes, down to 0.9 µA in Backup mode with RTC, RTT, and GPBR
- Peripherals
  - Two USARTs with USART1 (ISO7816, IrDA®, RS-485, SPI, Manchester and Modem Modes)
  - USB 2.0 Device: Full Speed (12 Mbits), 2668 byte FIFO, up to 8 Endpoints. On-chip Transceiver
  - Two 2-wire UARTs
  - Two 2-wire Interfaces (TWI)
  - High-speed Multimedia Card Interface (SDIO/SD Card/MMC)
  - One Master/Slave Serial Peripheral Interface (SPI) with Chip Select Signals
  - Three 3-channel 32-bit Timer/Counter blocks with Capture, Waveform, Compare and PWM Mode. Quadrature Decoder Logic and 2-bit Gray Up/Down Counter for Stepper Motor
  - 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)
  - One Ethernet MAC (GMAC) 10/100 Mbps in MII mode only with dedicated DMA and Support for IEEE1588, Wake-on-LAN
  - Two CAN Controllers with eight Mailboxes
  - 4-channel 16-bit PWM with Complementary Output, Fault Input, 12-bit Dead Time Generator Counter for Motor Control
  - Real-time Event Management

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

Note: 1. 120 MHz: -40/+105°C, VDDCORE = 1.2V

1.1 Configuration Summary

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

Table 1-1. Configuration Summary

Feature	SAM4E16E	SAM4E8E	SAM4E16C	SAM4E8C
Flash	1024 Kbytes	512 Kbytes	1024 Kbytes	512 Kbytes
SRAM	128 Kbytes		128 Kbytes	
CMCC	2 Kbytes		2 Kbytes	
Package	LFBGA 144 LQFP 144		TFBGA 100 LQFP 100	
Number of PIOs	117		79	
External Bus Interface	8-bit Data, 4 Chip Selects, 24-bit Address		–	
Analog Front End (AFE0\AFE1)	Up to 16 bits(1) 16 ch. / 8 ch. (2)		Up to 16 bits(1) 6 ch. / 4 ch. (3)	
GMAC	10/100 Mbps		10/100 Mbps	
CAN	2		1	
12-bit DAC	2 ch.		2 ch.	
Timer	9(4)		9(5)	
PDC Channels	24 +9		21 +9	
USART/ UART	2/2(6)		2/2(6)	
USB	Full Speed		Full Speed	
HSMCI	1 port, 4 bits		1 port, 4 bits	
TWI	2		2	

Notes:
1. ADC is 12-bit, up to 16 bits with averaging.
   For details, please refer to Section 46. “SAM4E Electrical Characteristics”.
2. AFE0 is 16 channels and AFE1 is 8 channels. The total number of AFE channels is 24.
   One channel is reserved for the internal temperature sensor.
3. AFE0 is 6 channels and AFE1 is 4 channels. The total number of AFE channels is 10.
   One channel is reserved for the internal temperature sensor.
4. Nine TC channels are accessible through PIO.
5. Three TC channels are accessible through PIO and 6 channels are reserved for internal use.
6. Full Modem support on USART1.

System Controller
TST
XIN
XOUT
PCK[2:0]
ERASE
WKUP[15:0]
XIN32
XOUT32
RTCOUT0
RTCOUT1
NRST
VDDPLL
VDDIO
VDDCORE
3–20 MHz Oscillator
RC Osc 4/8/12 MHz
PLLA
PMC
SUPC
Tamper Detection
32.768k Crystal Osc.
32k typ. RC Osc.
256-bit GPBR
RTC
RTT
POR
Backup
RSTC
SM
WDT
PIOA/B/C/D/E
TDI
TDO
TMS/SWDIO
TCK/SWCLK
JTAGSEL
JTAG and Serial Wire
In-Circuit Emulator
NVIC
Cortex-M4 Processor
fMAX 120 MHz
DSP
NVIC
24-bit SysTick Counter
MPU
FPU
I
D
HCACHE (CMCC)
VDDIN
VDDOUT
Voltage Regulator
ROM
16 Kbytes
SRAM
128 Kbytes
Flash
Unique ID
Flash
1024 Kbytes
512 Kbytes
External Bus Interface
Static Memory
NAND Flash
Ethernet MAC
MII
DMA
FIFO
128 byte TX
128 byte RX
S
M
7-layer Bus Matrix
fMAX 120 MHz
Peripheral Bridge 0
PDC0
PDC
DMA
2 x TWI
UART0
2 x USART
PIO
3 x TC
HSMCI
2 x 12-bit AFEC
Temp Sensor
ACC
12-bit DACC
SPI
UART1
PWM
TWD0..1
TWCK0..1
URXD0
UTXD0
SCK0..1
TXD0..1
RXD0..1
RTS0..1
CTS0..1
DSR1, DTR1
RI1, DCD1
PIOD0..7
PIODCEN1..2
PIODCCLK
TCLK0..9
TIOA0..9
TIOB0..9
MCCK
MCCDA
MCDA0..3
AFE_ADTRG
AFE_AD0..14
ADVREFN
ADVREFP
DAC0..1
DATRG
MISO
MOSI
SPCK
NPCS0..3
URXD1
UTXD1
PWMH0..3
PWML0..3
PWMFI0

3. Signal Description

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

Table 3-1. Signal Description List

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

Table 3-1. Signal Description List (Continued)

Signal Name	Function	Type	Active Level	Voltage Reference	Comments
Reset/Test					
NRST	Synchronous Microcontroller Reset	I/O	Low	VDDIO	Permanent Internal Pull-up
TST	Test Select	Input	–		Permanent Internal Pull-down
Wake-up					
WKUP[15:0]	Wake-up Inputs	Input	–	VDDIO	–
Universal Asynchronous Receiver Transceiver - UARTx					
URXDx	UART Receive Data	Input	–	–	–
UTXDx	UART Transmit Data	Output	–	–	–
PIO Controller - PIOA - PIOB - PIOC - PIOD - PIOE					
PA0–PA31	Parallel IO Controller A	I/O	–	VDDIO	Reset State: - PIO or System IOs(4) - Internal Pull-up enabled - Schmitt Trigger enabled(2)
PB0–PB14	Parallel IO Controller B	I/O	–		
PC0–PC31	Parallel IO Controller C	I/O	–		
PD0–PD31	Parallel IO Controller D	I/O	–		Reset State: - PIO or System IOs(4) - Internal Pull-up enabled - Schmitt Trigger enabled(2)
PE0–PE5	Parallel IO Controller E	I/O	–		
PIO Controller - Parallel Capture Mode					
PIODC0–PIODC7	Parallel Capture Mode Data	Input	–	VDDIO	–
PIODCCLK	Parallel Capture Mode Clock	Input	–		
PIODCEN1–2	Parallel Capture Mode Enable	Input	–		
High Speed Multimedia Card Interface - HSMCI					
MCCK	Multimedia Card Clock	I/O	–	–	–
MCCDA	Multimedia Card Slot A Command	I/O	–	–	–
MCDA0–MCDA3	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	–	–	–
DTR1	USART1 Data Terminal Ready	I/O	–	–	–
DSR1	USART1 Data Set Ready	Input	–	–	–
DCD1	USART1 Data Carrier Detect	Output	–	–	–
RI1	USART1 Ring Indicator	Input	–	–	–

Table 3-1. Signal Description List (Continued)

Signal Name	Function	Type	Active Level	Voltage Reference	Comments
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	–	–	–
Serial Peripheral Interface - SPI					
MISO	Master In Slave Out	I/O	–	–	–
MOSI	Master Out Slave In	I/O	–	–	–
SPCK	SPI Serial Clock	I/O	–	–	–
SPI_NPCS0	SPI Peripheral Chip Select 0	I/O	Low	–	–
SPI_NPCS1–SPI_NPCS3	SPI Peripheral Chip Select	Output	Low	–	–
Two-Wire Interface - TWIx					
TWDx	TWIx Two-wire Serial Data	I/O	–	–	–
TWCKx	TWIx Two-wire Serial Clock	I/O	–	–	–
Analog					
ADVREF	ADC, DAC and Analog Comparator Reference	Analog	–	–[1]	–
12-bit Analog-Front-End - AFEx					
AFE0_AD0–AFE0_AD14	Analog Inputs	Analog, Digital	–	–[1]	–
AFE1_AD0–AFE1_AD7	Analog Inputs	Analog, Digital	–	–[1]	–
ADTRG	Trigger	Input	–	VDDIO	–
12-bit Digital-to-Analog Converter - DAC					
DAC0–DAC1	Analog output	Analog, Digital	–	–[1]	–
DATRG	DAC Trigger	Input	–	VDDIO	–
Fast Flash Programming Interface - FFPI					
PGMEN0-PGMEN1	Programming Enable	Input		VDDIO	–
PGMM0-PGMM3	Programming Mode	Input			–
PGMD0-PGMD15	Programming Data	I/O			–
PGMRDY	Programming Ready	Output	High		–
PGMNVALID	Data Direction	Output	Low		–
PGMNOE	Programming Read	Input	Low		–
PGMCK	Programming Clock	Input			–
PGMNCMD	Programming Command	Input	Low		–
External Bus Interface					
D0–D7	Data Bus	I/O	–	–	–
A0–A23	Address Bus	Output	–	–	–
NWAIT	External Wait Signal	Input	Low	–	–

Table 3-1. Signal Description List (Continued)

Signal Name	Function	Type	Active Level	Voltage Reference	Comments
Static Memory Controller - SMC					
NCS0–NCS3	Chip Select Lines	Output	Low	–	–
NRD	Read Signal	Output	Low	–	–
NWE	Write Enable	Output	Low	–	–
NAND Flash Logic					
NANDOE	NAND Flash Output Enable	Output	Low	–	–
NANDWE	NAND Flash Write Enable	Output	Low	–	–
Pulse Width Modulation Controller - PWMC					
PWMH	PWM Waveform Output High for channel x	Output	–	–	–
PWML	PWM Waveform Output Low for channel x	Output	–	–	Only output in complementary mode when dead time insertion is enabled.
PWMFI0	**PWM Fault Input**	Input	–	–	–
Ethernet MAC 10/100 - GMAC					
GTXCK	Transmit Clock	Input	–	–	–
GRXCK	Receive Clock	Input	–	–	–
GTXEN	Transmit Enable	Output	–	–	–
GTX0–GTX3	Transmit Data	Output	–	–	–
GTXER	Transmit Coding Error	Output	–	–	–
GRXDV	Receive Data Valid	Input	–	–	–
GRX0–GRX3	Receive Data	Input	–	–	–
GRXER	Receive Error	Input	–	–	–
GCRS	Carrier Sense	Input	–	–	–
GCOL	Collision Detected	Input	–	–	–
GMDC	Management Data Clock	Output	–	–	–
GMDIO	Management Data Input/Output	I/O	–	–	–
Controller Area Network - CAN (x=[0:1])					
CANRXx	CAN Receive	Input	–	–	–
CANTXx	CAN Transmit	Output	–	–	–
USB Full Speed Device					
DDM	DDM USB Full Speed Data -	Analog, Digital	–	–(1)	Reset State: - USB Mode - Internal Pull-down
DDP	DDP USB Full Speed Data +		–	–(1)	Reset State: - USB Mode - Internal Pull-down

Notes: 1. See Section 5.4 “Typical Powering Schematics” for restrictions on voltage range of Analog Cells and USB.
2. Schmitt Triggers can be disabled through PIO registers.

3. TDO pin is set in input mode when the Cortex-M4 Core is not in debug mode. Thus the internal pull-up corresponding to this PIO line must be enabled to avoid current consumption due to floating input.
4. Some PIO lines are shared with System I/Os.

4. Package and Pinout

The SAM4E is available in TFBGA100, LFBGA144, LQFP100, and LQFP144 and packages described in Section 47. “SAM4E Mechanical Characteristics”.

4.1 100-ball TFBGA Package and Pinout

4.1.1 100-ball TFBGA Package Outline

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

4.1.2 100-ball TFBGA Pinout

Table 4-1. SAM4E 100-ball TFBGA Pinout

A1	PB9	C6	PD29	F1	PA19/PGMD7	H6	PA14/PGMD2
A2	PB8	C7	PA30	F2	PA20/PGMD8	H7	PA25/PGMD13
A3	PB14	C8	PB5	F3	PD23	H8	PA27/PGMD15
A4	PB10	C9	PD10	F4	GND	H9	PA5/PGMRDY
A5	PD4	C10	PA1/PGMEN1	F5	GND	H10	PA4/PGMNCMD
A6	PD7	D1	ADVREF	F6	GND	J1	PA21/PGMD9
A7	PA31	D2	PD1	F7	TST	J2	PA7/PGMNVALID
A8	PA6/PGMNOE	D3	GND	F8	PB12	J3	PA22/PGMD10
A9	PA28	D4	GND	F9	PA3	J4	PD22
A10	JTAGSEL	D5	PD5	F10	PD14	J5	PA16/PGMD4
B1	PD31	D6	VDDCORE	G1	PA17/PGMD5	J6	PA15/PGMD3
B2	PB13	D7	VDDCORE	G2	PA18/PGMD6	J7	PD28
B3	VDDPLL	D8	PA0/PGMEN0	G3	PD26	J8	PA11/PGMM3
B4	PB11	D9	PD11	G4	PD24	J9	PA9/PGMM1
B5	PD3	D10	PA2	G5	PA13/PGMD1	J10	PD17
B6	PD6	E1	PB0	G6	VDDCORE	K1	PD30
B7	PD8	E2	PB1	G7	VDDIO	K2	PA8/PGMM0
B8	PD9	E3	PD2	G8	PB6	K3	PD20
B9	PB4	E4	GND	G9	PD16	K4	PD19
B10	PD15	E5	VDDIO	G10	NRST	K5	PA23/PGMD11
C1	PD0	E6	VDDIO	H1	PB2	K6	PD18
C2	VDDIN	E7	GND	H2	PB3	K7	PA24/PGMD12
C3	VDDOUT	E8	PD13	H3	PD25	K8	PA26/PGMD14
C4	GND	E9	PB7	H4	PD27	K9	PA10/PGMM2
C5	PA29	E10	PD12	H5	PD21	K10	PA12/PGMD0

4.2 144-ball LFBGA Package and Pinout

4.2.1 144-ball LFBGA Package Outline

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

4.2.2 144-ball LFBGA Pinout

Table 4-2. SAM4E 144-ball LFBGA Pinout

A1	PE1	D1	ADVREF	G1	PC15	K1	PE4
A2	PB9	D2	GND	G2	PC13	K2	PA21/PGMD9
A3	PB8	D3	PD31	G3	PB1	K3	PA22/PGMD10
A4	PB11	D4	PD0	G4	GND	K4	PC2
A5	PD2	D5	GNDPLL	G5	GND	K5	PA16/PGMD4
A6	PA29	D6	PD4	G6	GND	K6	PA14/PGMD2
A7	PC21	D7	PD5	G7	GND	K7	PC6
A8	PD6	D8	PC19	G8	VDDIO	K8	PA25/PGMD13
A9	PC20	D9	PD9	G9	PD13	K9	PD20
A10	PA30	D10	PD29	G10	PD12	K10	PD28
A11	PD15	D11	PC16	G11	PC9	K11	PD16
A12	PB4	D12	PA1/PGMEN1	G12	PB12	K12	PA4/PGMNCMD
B1	PE2	E1	PC31	H1	PA19/PGMD7	L1	PE5
B2	PB13	E2	PC27	H2	PA18/PGMD6	L2	PA7/PGMNVALID
B3	VDDPLL	E3	PE3	H3	PA20/PGMD8	L3	PC3
B4	PB10	E4	PC0	H4	PB0	L4	PA23/PGMD11
B5	PD1	E5	GND	H5	VDDCORE	L5	PA15/PGMD3
B6	PC24	E6	GND	H6	VDDIO	L6	PD26
B7	PD3	E7	VDDIO	H7	VDDIO	L7	PA24/PGMD12
B8	PD7	E8	VDDCORE	H8	VDDCORE	L8	PC5
B9	PA6/PGMNOE	E9	PD8	H9	PD21	L9	PA10/PGMM2
B10	PC18	E10	PC14	H10	PD14	L10	PA12/PGMD0
B11	JTAGSEL	E11	PD11	H11	TEST	L11	PD17
B12	PC17	E12	PA2	H12	NRST	L12	PC28
C1	VDDIN	F1	PC30	J1	PA17/PGMD5	M1	PD30
C2	PE0	F2	PC26	J2	PB2	M2	PA8/PGMM0
C3	VDDOUT	F3	PC29	J3	PB3	M3	PA13/PGMD1
C4	PB14	F4	PC12	J4	PC1	M4	PC7
C5	PC25	F5	GND	J5	PC4	M5	PD25
C6	PC23	F6	GND	J6	PD27	M6	PD24
C7	PC22	F7	GND	J7	VDDCORE	M7	PD23
C8	PA31	F8	VDDIO	J8	PA26/PGMD14	M8	PD22
C9	PA28	F9	PB7	J9	PA11/PGMM3	M9	PD19
C10	PB5	F10	PC10	J10	PA27/PGMD15	M10	PD18
C11	PA0/PGMEN0	F11	PC11	J11	PB6	M11	PA5/PGMRDY
C12	PD10	F12	PA3	J12	PC8	M12	PA9/PGMM1

4.3 100-lead LQFP Package and Pinout

4.3.1 100-lead LQFP Package Outline

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

4.3.2 100-lead LQFP Pinout

Table 4-3. SAM4E 100-lead LQFP Pinout

1	PD0	26	PA22/PGMD10	51	PD28	76	PD29
2	PD31	27	PA13/PGMD1	52	PA5/PGMRDY	77	PB5
3	GND	28	VDDIO	53	PD17	78	PD9
4	VDDOUT	29	GND	54	PA9/PGMM1	79	PA28
5	VDDIN	30	PA16/PGMD4	55	PA4/PGMNCMD	80	PD8
6	GND	31	PA23/PGMD11	56	PD16	81	PA6/PGMNOE
7	GND	32	PD27	57	PB6	82	PA30
8	GND	33	PA15/PGMD3	58	NRST	83	PA31
9	ADVREF	34	PA14/PGMD2	59	PD14	84	PD7
10	GND	35	PD25	60	TST	85	PD6
11	PB1	36	PD26	61	PB12	86	VDDCORE
12	PB0	37	PD24	62	PD13	87	PD5
13	PA20/PGMD8	38	PA24PGMD12	63	PB7	88	PD4
14	PA19/PGMD7	39	PD23	64	PA3	89	PD3
15	PA18/PGMD6	40	PA25/PGMD13	65	PD12	90	PA29
16	PA17/PGMD5	41	PD22	66	PA2	91	PD2
17	PB2	42	PA26/PGMD14	67	GND	92	PD1
18	VDDCORE	43	PD21	68	VDDIO	93	VDDIO
19	VDDIO	44	PA11/PGMM3	69	PD11	94	PB10
20	PB3	45	PD20	70	PA1/PGMEN1	95	PB11
21	PA21/PGMD9	46	PA10/PGMM2	71	PD10	96	VDDPLL
22	VDDCORE	47	PD19	72	PA0/PGMEN0	97	PB14
23	PD30	48	PA12/PGMD0	73	JTAGSEL	98	PB8
24	PA7/PGMNVALID	49	PD18	74	PB4	99	PB9
25	PA8/PGMM0	50	PA27/PGMD15	75	PD15	100	PB13

4.4 144-lead LQFP Package and Pinout

4.4.1 144-lead LQFP Package Outline

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

4.4.2 144-lead LQFP Pinout

Table 4-4. SAM4E 144-lead LQFP Pinout

1	PD0	37	PA22/PGMD10	73	PA5/PGMRDY	109	PB5
2	PD31	38	PC1	74	PD17	110	PD9
3	VDDOUT	39	PC2	75	PA9/PGMM1	111	PC18
4	PE0	40	PC3	76	PC28	112	PA28
5	VDDIN	41	PC4	77	PA4/PGMNCMD	113	PD8
6	PE1	42	PA13/PGMD1	78	PD16	114	PA6/PGMNOE
7	PE2	43	VDDIO	79	PB6	115	GND
8	GND	44	GND	80	VDDIO	116	PA30
9	ADVREFP	45	PA16/PGMD4	81	VDDCORE	117	PC19
10	PE3	46	PA23/PGMD11	82	PC8	118	PA31
11	PC0	47	PD27	83	NRST	119	PD7
12	PC27	48	PC7	84	PD14	120	PC20
13	PC26	49	PA15/PGMD3	85	TEST	121	PD6
14	PC31	50	VDDCORE	86	PC9	122	PC21
15	PC30	51	PA14/PGMD2	87	PB12	123	VDDCORE
16	PC29	52	PD25	88	PD13	124	PC22
17	PC12	53	PD26	89	PB7	125	PD5
18	PC15	54	PC6	90	PC10	126	PD4
19	PC13	55	PD24	91	PA3	127	PC23
20	PB1	56	PA24/PGMD12	92	PD12	128	PD3
21	PB0	57	PD23	93	PA2	129	PA29
22	PA20/PGMD8	58	PC5	94	PC11	130	PC24
23	PA19/PGMD7	59	PA25/PGMD13	95	GND	131	PD2
24	PA18/PGMD6	60	PD22	96	VDDIO	132	PD1
25	PA17/PGMD5	61	GND	97	PC14	133	PC25
26	PB2	62	PA26/PGMD14	98	PD11	134	VDDIO
27	PE4	63	PD21	99	PA1/PGMEN1	135	GND
28	PE5	64	PA11/PGMM3	100	PC16	136	PB10
29	VDDCORE	65	PD20	101	PD10	137	PB11
30	VDDIO	66	PA10/PGMM2	102	PA0/PGMEN0	138	GND
31	PB3	67	PD19	103	PC17	139	VDDPLL
32	PA21/PGMD9	68	PA12/PGMD0	104	JTAGSEL	140	PB14
33	VDDCORE	69	PD18	105	PB4	141	PB8
34	PD30	70	PA27/PGMD15	106	PD15	142	PB9
35	PA7/PGMNVALID	71	PD28	107	VDDCORE	143	VDDIO
36	PA8/PGMM0	72	VDDIO	108	PD29	144	PB13

5. Power Considerations

5.1 Power Supplies

The SAM4E has several types of power supply pins:

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

5.2 Power-up Considerations

5.2.1 VDDIO Versus VDDCORE

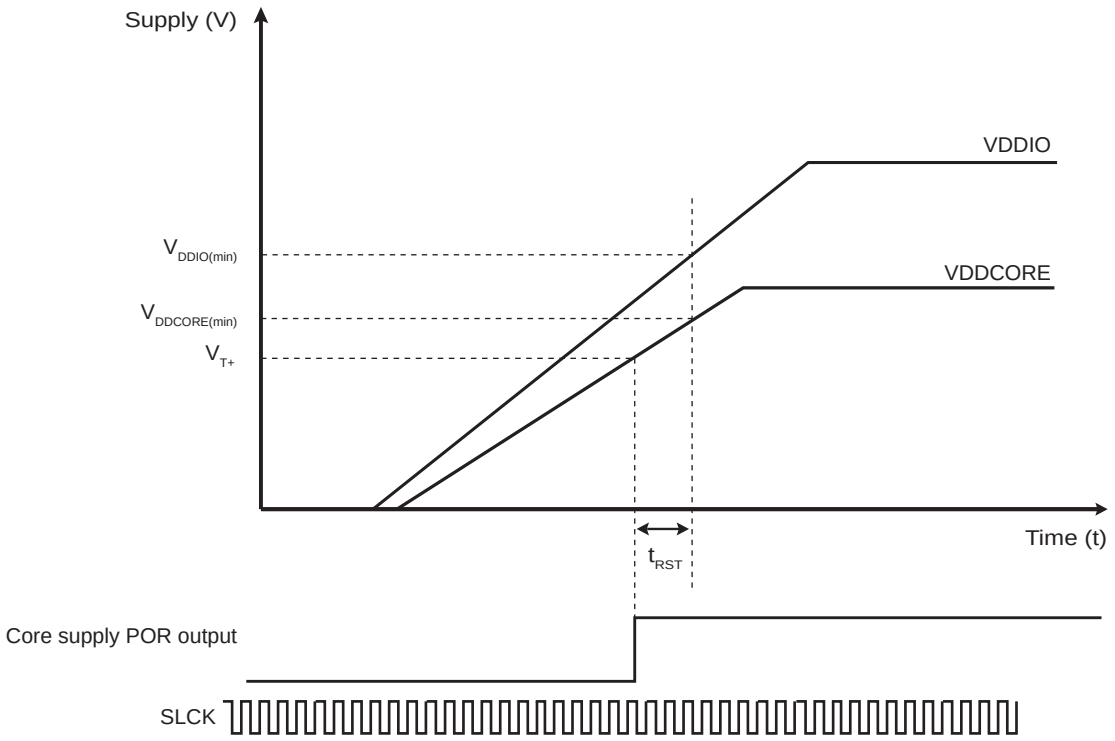
V_{DDIO} must always be higher than or equal to V_{DDCORE}.

V_{DDIO} must reach its minimum operating voltage (1.62 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 8.8 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 SAM4E embeds a voltage regulator that is managed by the Supply Controller.

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

- In Normal mode, the voltage regulator consumes less than 500 µA static current and draws 80 mA of output current. Internal adaptive biasing adjusts the regulator quiescent current depending on the required load current. In Wait Mode quiescent current is only 5 µA.
- In Backup mode, the voltage regulator consumes less than 1.5 µA while its output (VDDOUT) is driven internally to GND. The default output voltage is 1.20V and the start-up time to reach Normal mode is less than 300 µs.

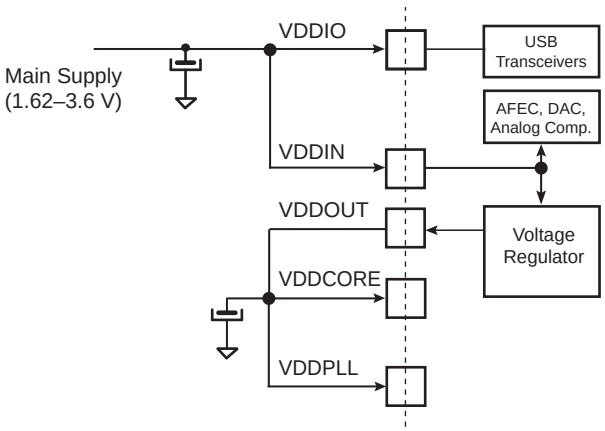
For adequate input and output power supply decoupling/bypassing, refer to Table 46-3, “1.2V Voltage Regulator Characteristics,” on page 1357.

5.4 Typical Powering Schematics

The SAM4E supports a 1.62–3.6 V single supply mode. The internal regulator input is connected to the source and its output feeds VDDCORE. Figure 5-2 shows the power schematics.

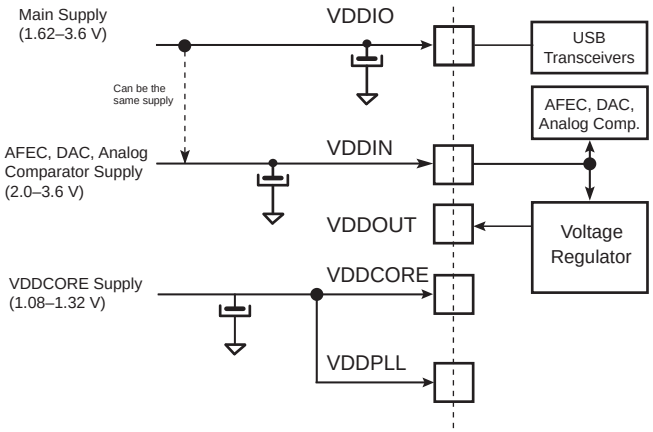
As VDDIN powers the voltage regulator, the DAC and the analog comparator, when the user does not want to use the embedded voltage regulator, it can be disabled by software via the SUPC (note that this is different from Backup mode).

Figure 5-2. Single Supply



Note: Restrictions:
- For USB, VDDIO needs to be greater than 3.0V
- For AFEC, DAC, and Analog Comparator, VDDIN needs to be greater than 2.4V

Figure 5-3. Core Externally Supplied



Note: Restrictions:
- For USB, VDDIO needs to be greater than 3.0V
- For AFEC, DAC, and Analog Comparator, VDDIN needs to be greater than 2.4V

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 SAM4E has the following low-power modes: Backup mode, Wait mode and Sleep mode.

Note: The Wait For Event instruction (WFE) of the Cortex-M4 core can be used to enter any of the low-power modes, however, this may add complexity in the design of application state machines. This is due to the fact that the WFE instruction goes along with an event flag of the Cortex core (cannot be managed by the software application). The event flag can be set by interrupts, a debug event or an event signal from another processor. Since it is possible for an interrupt to occur just before the execution of WFE, WFE takes into account events that happened in the past. As a result, WFE prevents the device from entering wait mode if an interrupt event has occurred.
Atmel has made provision to avoid using the WFE instruction. The workarounds to ease application design are as follows:
- For backup mode, switch off the voltage regulator and configure the VROFF bit in the Supply Controller Control Register (SUPC_CR).
- For wait mode, configure the WAITMODE bit in the PMC Clock Generator Main Oscillator Register of the Power Management Controller (PMC)
- For sleep mode, use the Wait for Interrupt (WFI) instruction.

Complete information is available in Table 5-1 “Low-power Mode Configuration Summary”.

5.6.1 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. Total current consumption is 1 µA typical (VDDIO = 1.8 V at 25°C).

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

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

Backup mode is entered by writing a 1 to the VROFF bit of the Supply Controller Control Register (SUPC_CR) (A key is needed to write the VROFF bit, refer to Section 18. “Supply Controller (SUPC)”) and with the SLEEPDEEP bit in the Cortex-M4 System Control Register set to 1. (See the power management description in Section 11. “ARM Cortex-M4 Processor”).

To enter Backup mode using the VROFF bit:

- Write a 1 to the VROFF bit of SUPC_CR.

To enter Backup mode using the WFE instruction:

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

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

- Level transition, configurable debouncing on pins WKUPEN0–15
- Supply Monitor alarm
- RTC alarm
- RTT alarm

5.6.2 Wait Mode

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

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

This mode is entered by setting the WAITMODE bit to 1 in the PMC Clock Generator Main Oscillator Register (CKGR_MOR) in conjunction with FLPM = 0 or FLPM = 1 bits of the PMC Fast Startup Mode Register (PMC_FSMR) or by the WFE instruction.

The Cortex-M4 is able to handle external or internal events in order to wake-up the core. This is done by configuring the external lines WKUP0–15 as fast startup wake-up pins (refer to Section 5.8 “Fast Start-up”). RTC or RTT Alarm and USB wake-up events can be used to wake up the CPU.

To enter Wait mode with WAITMODE bit:

1. Select the 4/8/12 MHz fast RC oscillator as Main Clock.
2. Set the FLPM field in the PMC_FSMR.
3. Set Flash Wait State to 0.
4. Set the WAITMODE bit = 1 in CKGR_MOR.
5. Wait for Master Clock Ready MCKRDY = 1 in the PMC Status Register (PMC_SR).

To enter Wait mode with WFE:

1. Select the 4/8/12 MHz fast RC oscillator as Main Clock.
2. Set the FLPM field in the PMC_FSMR.
3. Set Flash Wait State to 0.
4. Set the LPM bit in the PMC_FSMR.
5. Execute the Wait-For-Event (WFE) instruction of the processor.

In both cases, depending on the value of the field FLPM, the Flash enters one of three different modes:

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

Table 5-1 summarizes the power consumption, wake-up time and system state in Wait mode.

5.6.3 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. The current consumption in this mode is application dependent.

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

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

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 configured individually. Table 5-1 provides the configuration summary of the low-power modes.

Table 5-1. Low-power Mode Configuration Summary

Mode	SUPC, 32 kHz Osc., RTC, RTT, GPBR, POR (Backup Region)	Regulator	Core Memory Peripherals	Mode Entry	Potential Wake-Up Sources	Core at Wake-Up	PIO State while in Low-Power Mode
Backup Mode	ON	OFF	OFF (Not powered)	VROFF = 1 or WFE + SLEEPDEEP = 1	WKUP0–15 pins SM alarm RTC alarm RTT alarm	Reset	Previous state saved
Wait Mode w/Flash in Standby Mode	ON	ON	Powered (Not clocked)	WAITMODE = 1 + FLPM = 0 or WFE + SLEEPDEEP = 0 + LPM = 1 + FLPM = 0	Any Event from: Fast startup through WKUP0–15 RTC alarm RTT alarm USB wake-up	Clocked back	Previous state saved
Wait Mode w/Flash in Deep Power-down Mode	ON	ON	Powered (Not clocked)	WAITMODE = 1 + FLPM = 1 or WFE + SLEEPDEEP = 0 + LPM = 1 + FLPM = 1	Any Event from: Fast startup through WKUP0–15 RTC alarm RTT alarm USB wake-up	Clocked back	Previous state saved
Sleep Mode	ON	ON	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 start-up through WKUP0–15 RTC alarm RTT alarm USB wake-up	Clocked back	Previous state saved

Notes:
1. When considering wake-up time, the time required to start the PLL is not taken into account. Once started, the device w oscillator. The user has to add the PLL start-up time if it is needed in the system. The wake-up time is defined as the tim instruction is fetched.
2. The external loads on PIOs are not taken into account in the calculation.
3. Supply Monitor current consumption is not included.
4. Total consumption is 1 µA typical (VDDIO = 1.8 V at 25°C).
5. Power consumption on VDDCORE. For total current consumption, please refer to Section 46. “SAM4E Electrical Charac
6. Depends on MCK frequency.
7. 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 and the SRAM power supply, if they are not already enabled. See Figure 18-4 ”Wake-up Sources”.

5.8 Fast Start-up

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

The fast restart circuitry (shown in Figure 29-4 ”Fast Startup Circuitry”) is fully asynchronous and provides a fast start-up signal to the Power Management Controller. As soon as the fast start-up signal is asserted, the PMC automatically restarts the embedded 4/8/12 MHz Fast RC oscillator, switches the master clock on this 4 MHz clock by default and reenables the processor clock.

6. Input/Output Lines

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

6.1 General Purpose I/O Lines

GPIO Lines are managed by PIO Controllers. All I/Os have several input or output modes such as pull-up or pull-down, input Schmitt triggers, multi-drive (open-drain), glitch filters, debouncing or input change interrupt. Programming of these modes is performed independently for each I/O line through the PIO controller user interface. For more details, refer to Section 33. “Parallel Input/Output Controller (PIO)”.

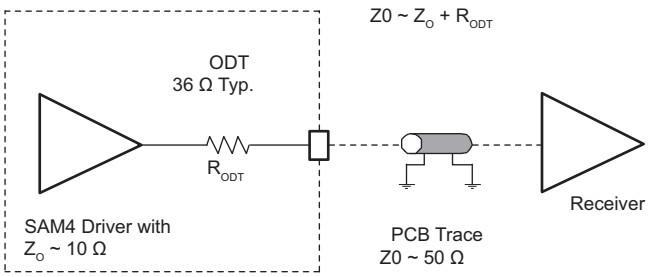
Some GPIOs can have an alternate function as analog input. When a GPIO is set in analog mode, all digital features of the I/O are disabled.

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

The SAM4E device embeds high speed pads able. See Section 46.11 “AC Characteristics” for more details. Typical pull-up and pull-down value is 100 kΩ for all I/Os.

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

Figure 6-1. On-die Termination



6.2 System I/O Lines

Table 6-1 lists the SAM4E 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

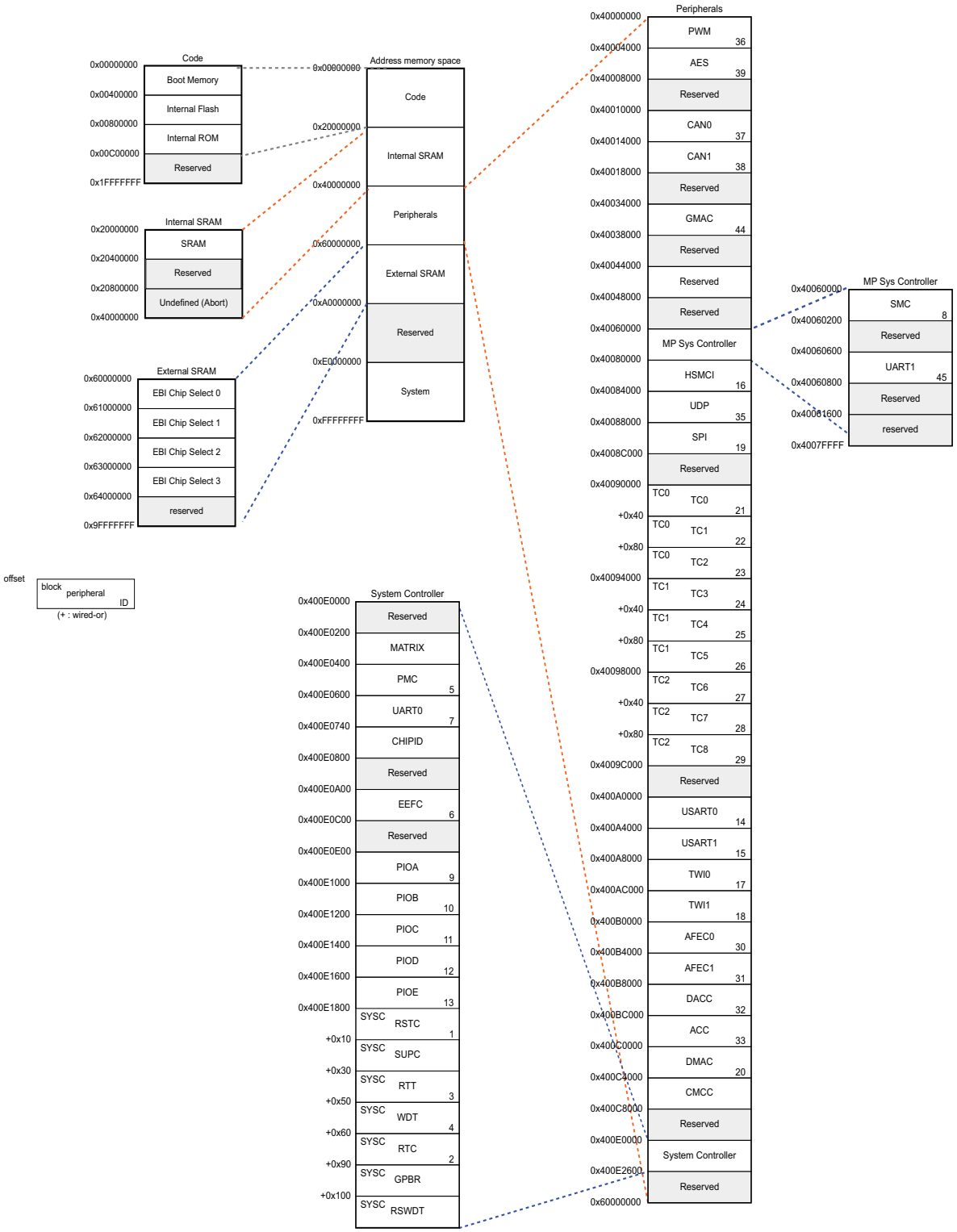
CCFG_SYSIO Bit No.	Default Function after Reset	Other Function	Constraints for Normal Start	Configuration
12	ERASE	PB12	Low Level at startup(1)	In Matrix User Interface Registers (Refer to the System I/O Configuration Register in Section 24. “Bus Matrix (MATRIX)”.)
7	TCK/SWCLK	PB7	–	
6	TMS/SWDIO	PB6	–	
5	TDO/TRACESWO	PB5	–	
4	TDI	PB4	–	
–	PA7	XIN32(2)	–	(3)
–	PA8	XOUT32(2)	–	
–	PB9	XIN	–	(4)
–	PB8	XOUT	–	

Notes:
1. If PB12 is used as PIO input in user applications, a low level must be ensured at startup to prevent Flash erase before the user application sets PB12 into PIO mode.
2. When the 32kHz oscillator is used in Bypass mode, XIN32 (PA7) is used as external clock source input and XOUT32 (PA8) can be left unconnected or used as GPIO.
3. Refer to Section 18.4.2 “Slow Clock Generator”.
4. Refer to Section 28.5.3 “3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator”.

7. Memories

7.1 Product Mapping

Figure 7-1. SAM4E Product Mapping



7.2 Embedded Memories

7.2.1 Internal SRAM

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

The SRAM is accessible over System Cortex-M4 bus at address 0x2000_0000.

The SRAM is in the bit band region. The bit band alias region is from 0x2200_0000 to 0x23FF_FFFF.

7.2.2 Internal ROM

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

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

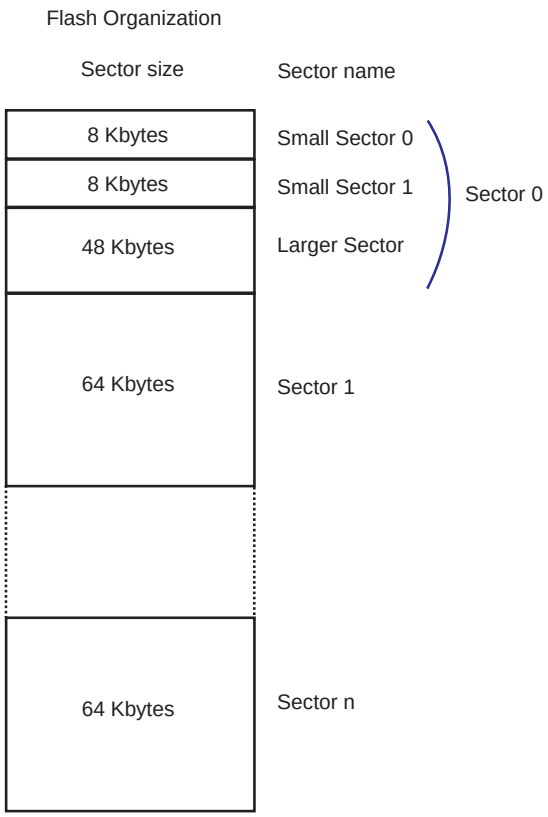
7.2.3 Embedded Flash

7.2.3.1 Flash Overview

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

The three smaller sectors are organized to consist of two sectors of 8 Kbytes and one sector of 48 Kbytes. Refer to Figure 7-2.

Figure 7-2. Global Flash Organization



Each Sector is organized in pages of 512 bytes.

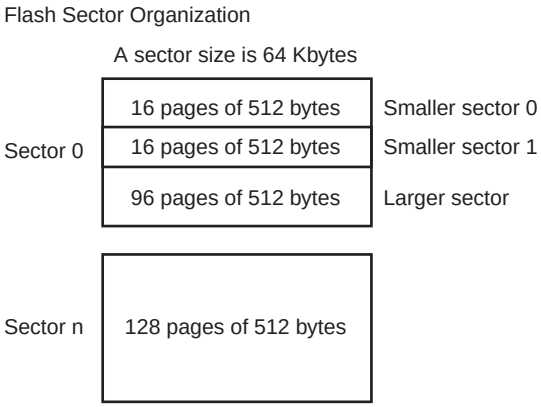
For sector 0:

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

From Sector 1 to n:

The rest of the array is composed of 64 Kbyte sector of each 128 pages of 512 bytes. Refer to Figure 7-3.

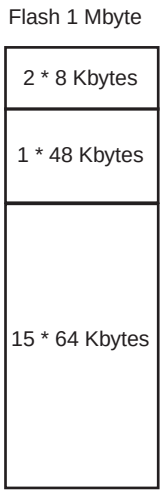
Figure 7-3. Flash Sector Organization



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

Refer to Figure 7-4 for the organization of the Flash following its size.

Figure 7-4. Flash Size



The following erase commands can be used depending on the sector size:

- 8 Kbyte small sector
  - Erase and write page (EWP)
  - Erase and write page and lock (EWPL)
  - Erase sector (ES) with FARG set to a page number in the sector to erase
  - Erase pages (EPA) with FARG [1:0] = 0 to erase four pages or FARG [1:0] = 1 to erase eight pages. FARG [1:0] = 2 and FARG [1:0] = 3 must not be used.
- 48 Kbyte and 64 Kbyte sectors
  - One block of 8 pages inside any sector, with the command Erase pages (EPA) with FARG[1:0] = 1
  - One block of 16 pages inside any sector, with the command Erase pages (EPA) and FARG[1:0] = 2
  - One block of 32 pages inside any sector, with the command Erase pages (EPA) and FARG[1:0] = 3
  - One sector with the command Erase sector (ES) and FARG set to a page number in the sector to erase
- Entire memory plane
  - The entire Flash, with the command Erase all (EA).

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

7.2.3.2 Enhanced Embedded Flash Controller

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

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

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

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

7.2.3.3 Flash Speed

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

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

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

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 7-1. Lock Bit Number

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

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

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

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

7.2.3.5 Security Bit Feature

The SAM4E device features a security bit, based on a specific General Purpose NVM bit (GPNVM bit 0). When the security is enabled, any access to the Flash, SRAM, Core Registers and Internal Peripherals either through the ICE interface or through the Fast Flash Programming Interface, is forbidden. This ensures the confidentiality of the code programmed in the Flash.

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

The ERASE pin integrates a permanent pull-down. Consequently, it can be left unconnected during normal operation. However, it is recommended, in harsh environment, to connect it directly to GND if the erase operation is not used in the application.

To avoid unexpected erase at power-up, a minimum ERASE pin assertion time is required. This time is defined in Table 46-68 “AC Flash Characteristics”.

The erase operation is not performed when the system is in Wait mode with the Flash in Deep-power-down mode.

To make sure that the erase operation is performed after power-up, the system must not reconfigure the ERASE pin as GPIO or enter Wait mode with Flash in Deep-power-down mode before the ERASE pin assertion time has elapsed.

The following sequence ensures the erase operation in all cases:

1. Assert the ERASE pin (High)
2. Assert the NRST pin (Low)
3. Power cycle the device
4. Maintain the ERASE pin high for at least the minimum assertion time.

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 User Signature

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

7.2.3.9 Fast Flash Programming Interface

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

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

The Fast Flash Programming Interface is enabled and the Fast Programming Mode is entered when TST and PA0 and PA1are tied low.

7.2.3.10 SAM-BA Boot

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

The SAM-BA Boot Assistant supports serial communication via the UART.

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

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

7.2.3.11 GPNVM Bits

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

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

Table 7-2. General-purpose Non-volatile Memory Bits

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

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

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

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

7.3 External Memories

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

7.4 Cortex-M Cache Controller (CMCC)

The SAM4E device features one cache memory and his controller which improve code execution when the code runs out of Code section (memory from 0x0 to 0x2000_0000).

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

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

8. Real-time Event Management

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

8.1 Embedded Characteristics

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

8.2 Real-time Event Mapping

Table 8-1. Real-time Event Mapping List

Function	Application	Description	Event Source	Event Destination
Security	General-purpose	Immediate GPBR clear (asynchronous) on Tamper detection through WKUP0/1 IO pins (1)	Parallel Input/Output Controller (PIO): WKUP0/1	General Purpose Backup Registers (GPBR)
Safety	General-purpose	Automatic Switch to reliable main RC oscillator in case of Main Crystal Clock Failure (2)	Power Management Controller (PMC)	PMC
	General-purpose, motor control	Puts the PWM Outputs in Safe Mode (Main Crystal Clock Failure Detection) (2)(3)	PMC	Pulse Width Modulation (PWM)
	Motor control	Puts the PWM Outputs in Safe Mode (Overcurrent sensor, ...) (3)(4)	Analog Comparator Controller (ACC)	
		Puts the PWM Outputs in Safe Mode (Overspeed, Overcurrent detection ...) (3)(5)	Analog-Front-End-Controller (AFEC0/1)	
		Puts the PWM Outputs in Safe Mode (Overspeed detection through TIMER Quadrature Decoder) (3)(6)	Timer Counter (TC)	
	General-purpose, motor control	Puts the PWM Outputs in Safe Mode (General Purpose Fault Inputs) (3)	PIO	
Image capture	Low-cost image sensor	PC is embedded in PIO (Capture Image from Sensor directly to System Memory) (7)	PIO	DMA

Table 8-1. Real-time Event Mapping List (Continued)

Function	Application	Description	Event Source	Event Destination
Measurement trigger	General-purpose	Trigger source selection in AFEC [8]	PIO (ADTRG)	AFEC
			TC Output 0	
			TC Output 1	
			TC Output 2	
	Motor control	ADC-PWM synchronization [9][10] Trigger source selection in AFEC [8]	PWM Event Line 0	
			PWM Event Line 1	
Delay measurement	Motor control	Propagation delay of external components (IOs, power transistor bridge driver, etc.) [11][12]	PWM Output Compare Line 0	TC Input (A/B) 0
			PWM Output Compare Line 1	TC Input (A/B) 1
			PWM Output Compare Line 2	TC Input (A/B) 2
Conversion trigger	General-purpose	Trigger source selection in DACC [13]	PIO DATRG	Digital-Analog Converter Controller (DACC)
			TC Output 0	
			TC Output 1	
			TC Output 2	
			PWM Event Line 0 [10]	
			PWM Event Line 1 [10]	

Notes:
1. Refer to “Low-power Tamper Detection and Anti-Tampering” in Section 18. “Supply Controller (SUPC)” and “General Purpose Backup Register x” in Section 19. “General Purpose Backup Registers (GPBR)”.
2. Refer to “Main Clock Failure Detector” in Section 29. “Power Management Controller (PMC)”.
3. Refer to “Fault Inputs” and “Fault Protection” in “Pulse Width Modulation Controller (PWM)” .
4. Refer to “Fault Mode” in “Analog Comparator Controller (ACC)” .
5. Refer to “Fault Output” in Section 43. “Analog Front-End Controller (AFEC)”.
6. Refer to “Fault Mode” in “Timer Counter (TC)” .
7. Refer to “Parallel Capture Mode” in Section 33. “Parallel Input/Output Controller (PIO)”.
8. Refer to “Conversion Triggers” and the AFEC Mode Register (AFEC_MR) in Section 43. “Analog Front-End Controller (AFEC)”.
9. Refer to PWM Comparison Value Register (PWM_CMPV) in Section 39. “Pulse Width Modulation Controller (PWM)”.
10. Refer to “PWM Comparison Units” and “PWM Event Lines” in Section 39. “Pulse Width Modulation Controller (PWM)”.
11. Refer to “Comparator” in Section 39. “Pulse Width Modulation Controller (PWM)”.
12. Refer to “Synchronization with PWM” in Section 38. “Timer Counter (TC)”.
13. Refer to DACC Trigger Register (DACC_TRIGR) in Section 44. “Digital-to-Analog Converter Controller (DACC)”.

9. System Controller

9.1 System Controller and Peripherals Mapping

Please refer to Figure 7-1 "SAM4E Product Mapping".

9.2 Power-on-Reset, Brownout and Supply Monitor

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

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

9.2.1 Power-on-Reset

The Power-on-Reset monitors VDDIO. It is always activated and monitors voltage at start up but also during power down. If VDDIO goes below the threshold voltage, the entire chip is reset. For more information, refer to Section 46. "SAM4E Electrical Characteristics".

9.2.2 Brownout Detector on VDDCORE

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

If VDDCORE goes below the threshold voltage, the reset of the core is asserted. For more information, refer to the Section 18. "Supply Controller (SUPC)" and Section 46. "SAM4E Electrical Characteristics".

9.2.3 Supply Monitor on VDDIO

The Supply Monitor monitors VDDIO. It is not active by default. It can be activated by software and is fully programmable with 16 steps for the threshold (between 1.6V to 3.4V). It is controlled by the Supply Controller (SUPC). A sample mode is possible. It allows to divide the supply monitor power consumption by a factor of up to 2048. For more information, refer to the Section 18. "Supply Controller (SUPC)" and Section 46. "SAM4E Electrical Characteristics".

10. Peripherals

10.1 Peripheral Identifiers

Table 10-1 defines the Peripheral Identifiers of the SAM4E device. A peripheral identifier is required for the control of the peripheral interrupt with the Nested Vectored Interrupt Controller and control of the peripheral clock with the Power Management Controller.

Table 10-1. Peripheral Identifiers

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

Table 10-1. Peripheral Identifiers (Continued)

Instance ID	Instance Name	NVIC Interrupt	PMC Clock Control	Instance Description
29	TC8	X	X	Timer/Counter Channel 8
30	AFEC0	X	X	Analog Front End Controller 0
31	AFEC1	X	X	Analog Front End Controller 1
32	DACC	X	X	Digital to Analog Converter Controller
33	ACC	X	X	Analog Comparator Controller
34	ARM	X		FPU signals: FPIXC, FPOFC, FPUFC, FPIOC, FPDZC, FPIDC, FPIXC
35	UDP	X	X	USB Device Port
36	PWM	X	X	Pulse Width Modulation Controller
37	CAN0	X	X	Controller Area Network 0
38	CAN1	X	X	Controller Area Network 1
39	AES	X	X	Advanced Encryption Standard
40				Reserved
41				Reserved
42				Reserved
43				Reserved
44	GMAC	X	X	Ethernet MAC
45	UART1	X	X	Universal Asynchronous Receiver Transmitter 1
46				Reserved

10.2 Peripheral Signal Multiplexing on I/O Lines

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

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

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

10.2.1 PIO Controller A Multiplexing

Table 10-2. Multiplexing on PIO Controller A (PIOA)

I/O Line	Peripheral A	Peripheral B	Peripheral C	Extra Function	System Function
PA0	PWMH0	TIOA0	A17	WKUP0[1]	
PA1	PWMH1	TIOB0	A18	WKUP1[1]	
PA2	PWMH2		DATRG	WKUP2[1]	
PA3	TWD0	NPCS3			
PA4	TWCK0	TCLK0		WKUP3[1]	
PA5		NPCS3	URXD1	WKUP4[1]	
PA6		PCK0	UTXD1		
PA7		PWMH3			XIN32[2]
PA8		AFE0_ADTRG		WKUP5[1]	XOUT32[2]
PA9	URXD0	NPCS1	PWMFI0	WKUP6[1]	
PA10	UTXD0	NPCS2			
PA11	NPCS0	PWMH0		WKUP7[1]	
PA12	MISO	PWMH1			
PA13	MOSI	PWMH2			
PA14	SPCK	PWMH3		WKUP8[1]	
PA15		TIOA1	PWML3	WKUP14/PIODCEN1[3]	
PA16		TIOB1	PWML2	WKUP15/PIODCEN2[3]	
PA17		PCK1	PWMH3	AFE0_AD0[4]	
PA18		PCK2	A14	AFE0_AD1[4]	
PA19		PWML0	A15	AFE0_AD2/WKUP9[5]	
PA20		PWML1	A16	AFE0_AD3/WKUP10[5]	
PA21	RXD1	PCK1		AFE1_AD2[4]	
PA22	TXD1	NPCS3	NCS2	AFE1_AD3[4]	
PA23	SCK1	PWMH0	A19	PIODCCLK[6]	
PA24	RTS1	PWMH1	A20	PIODC0[6]	
PA25	CTS1	PWMH2	A23	PIODC1[6]	
PA26	DCD1	TIOA2	MCDA2	PIODC2[6]	
PA27	DTR1	TIOB2	MCDA3	PIODC3[6]	
PA28	DSR1	TCLK1	MCCDA	PIODC4[6]	
PA29	RI1	TCLK2	MCCK	PIODC5[6]	
PA30	PWML2	NPCS2	MCDA0	WKUP11/PIODC6[3]	
PA31	NPCS1	PCK2	MCDA1	PIODC7[6]	

Notes:
1. WKUPx can be used if PIO controller defines the I/O line as "input".
2. Refer to Section 6.2 “System I/O Lines”.
3. PIODCENx/PIODCx has priority over WKUPx. Refer to Section 33.5.14 “Parallel Capture Mode”.
4. To select this extra function, refer to Section 43.5.1 “I/O Lines”.

5. Analog input has priority over WKUPx pin.
6. To select this extra function, refer to Section 33.5.14 “Parallel Capture Mode”.

10.2.2 PIO Controller B Multiplexing

Table 10-3. Multiplexing on PIO Controller B (PIOB)

I/O Line	Peripheral A	Peripheral B	Peripheral C	Extra Function	System Function
PB0	PWMH0		RXD0	AFE0_AD4/RTCOUT0(1)	
PB1	PWMH1		TXD0	AFE0_AD5/RTCOUT1(1)	
PB2	CANTX0	NPCS2	CTS0	AFE1_AD0/WKUP12(2)	
PB3	CANRX0	PCK2	RTS0	AFE1_AD1(3)	
PB4	TWD1	PWMH2			TDI(5)
PB5	TWCK1	PWML0		WKUP13(4)	TDO/TRACESWO(5)
PB6					TMS/SWDIO(5)
PB7					TCK/SWCLK(5)
PB8					XOUT(5)
PB9					XIN(5)
PB10					DDM
PB11					DDP
PB12	PWML1				ERASE(5)
PB13	PWML2	PCK0	SCK0	DAC0(6)	
PB14	NPCS1	PWMH3		DAC1(6)	

Notes:
1. Analog input has priority over RTCOUTx pin. See Section 15.5.8 “Waveform Generation”.
2. Analog input has priority over WKUPx pin.
3. To select this extra function, refer to Section 43.5.1 “I/O Lines”.
4. WKUPx can be used if PIO controller defines the I/O line as "input".
5. Refer to Section 6.2 “System I/O Lines”.
6. 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”.

10.2.3 PIO Controller C Multiplexing

Table 10-4. Multiplexing on PIO Controller C (PIOC)

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

Notes: 1. To select this extra function, refer to Section 43.5.1 “I/O Lines”.

10.2.4 PIO Controller D Multiplexing

Table 10-5. Multiplexing on PIO Controller D (PIOD)

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

10.2.5 PIO Controller E Multiplexing

Table 10-6. Multiplexing on PIO Controller E (PIOE)

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

11. Cortex-M4 processor

11.1 Description

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

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

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

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

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

11.1.1 System Level Interface

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

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

11.1.2 Integrated Configurable Debug

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

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

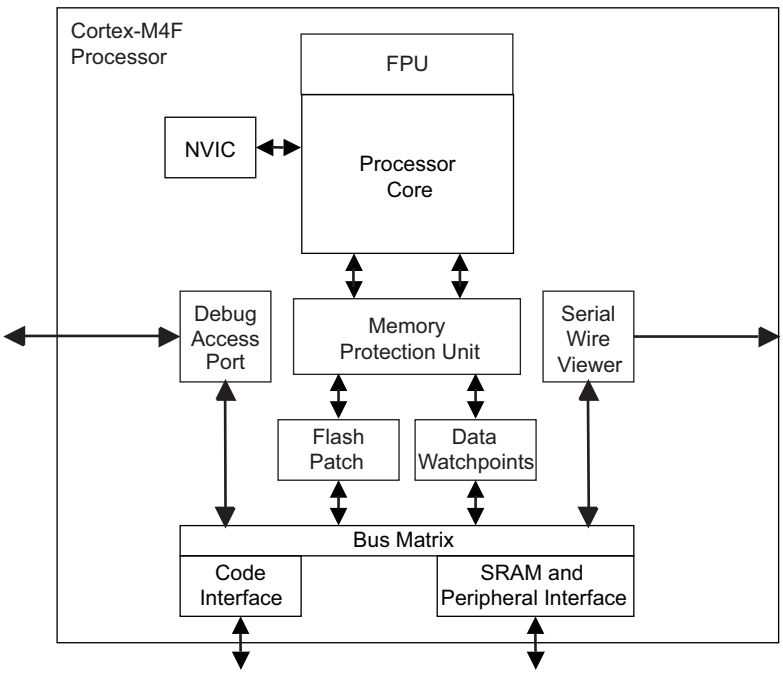
The Flash Patch and Breakpoint Unit (FPB) provides up to eight hardware breakpoint comparators that debuggers can use. The comparators in the FPB also provide remap functions of up to eight words in the program code in the CODE memory region. This enables applications stored on a non-erasable, ROM-based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

11.2 Embedded Characteristics

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

11.3 Block Diagram

Figure 11-1. Typical Cortex-M4F Implementation



11.4 Cortex-M4 Models

11.4.1 Programmers Model

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

11.4.1.1 Processor Modes and Privilege Levels for Software Execution

The processor *modes* are:

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

The *privilege levels* for software execution are:

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

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

In Thread mode, the Control Register controls whether the software execution is privileged or unprivileged, see "Control Register" . In Handler mode, software execution is always privileged.

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

11.4.1.2 Stacks

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

In Thread mode, the Control Register controls whether the processor uses the main stack or the process stack, see "Control Register" .

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

The options for processor operations are:

Table 11-1. 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

Note: 1. See "Control Register" .

11.4.1.3 Core Registers

Figure 11-2. Processor Core Registers

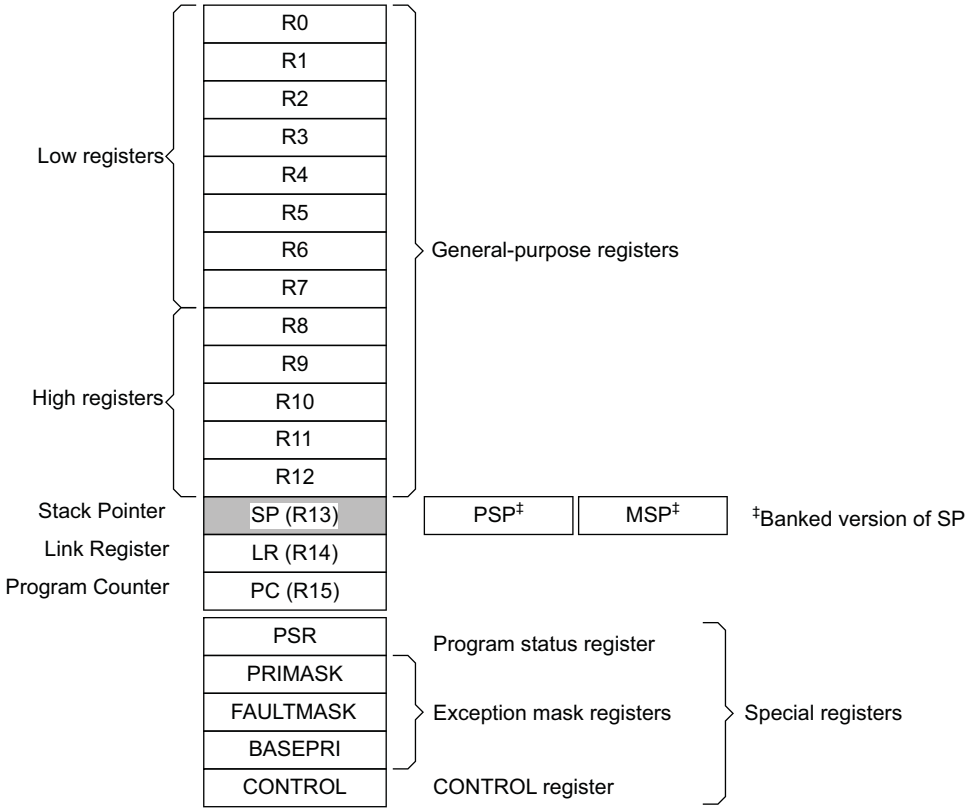


Table 11-2. Core Processor Registers

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

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

11.4.1.4 General-purpose Registers

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

11.4.1.5 Stack Pointer

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

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

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

11.4.1.6 Link Register

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

11.4.1.7 Program Counter

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

11.4.1.8 Program Status Register

Name: PSR

Access: Read/Write

Reset: 0x000000000

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

The *Program Status Register* (PSR) combines:

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

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

The PSR accesses 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 of all the registers using PSR with the MRS instruction
- Write to the APSR N, Z, C, V and Q bits using APSR_nzcvq with the MSR instruction.

The PSR combinations and attributes are:

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

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

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

11.4.1.9 Application Program Status Register

Name: APSR

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
N	Z	C	V	Q	–		

23	22	21	20	19	18	17	16
–				GE[3:0]			

15	14	13	12	11	10	9	8
–							

7	6	5	4	3	2	1	0
–							

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

- **N: Negative Flag**

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

1: Operation result was negative or less than.

- **Z: Zero Flag**

0: Operation result was not zero

1: Operation result was zero.

- **C: Carry or Borrow Flag**

Carry or borrow flag:

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

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

- **V: Overflow Flag**

0: Operation did not result in an overflow

1: Operation resulted in an overflow.

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

Sticky saturation flag:

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

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

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

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

See “SEL” for more information.

11.4.1.10 Interrupt Program Status Register

Name: IPSR

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–							ISR_NUMBER

7	6	5	4	3	2	1	0
ISR_NUMBER							

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

- **ISR_NUMBER: Number of the Current Exception**

0 = Thread mode

1 = Reserved

2 = NMI

3 = Hard fault

4 = Memory management fault

5 = Bus fault

6 = Usage fault

7–10 = Reserved

11 = SVCall

12 = Reserved for Debug

13 = Reserved

14 = PendSV

15 = SysTick

16 = IRQ0

49 = IRQ46

See “Exception Types” for more information.

11.4.1.11 Execution Program Status Register

Name: EPSR

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
–					ICI/IT		T

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
ICI/IT						–	

7	6	5	4	3	2	1	0
–							

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

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

- **ICI: Interruptible-continuable Instruction**

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

– Stops the load multiple or store multiple instruction operation temporarily

– Stores the next register operand in the multiple operation to EPSR bits[15:12].

After servicing the interrupt, the processor:

– Returns to the register pointed to by bits[15:12]

– Resumes the execution of the multiple load or store instruction.

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

- **IT: If-Then Instruction**

Indicates the execution state bits of the IT instruction.

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

- **T: Thumb State**

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

– Instructions BLX, BX and POP{PC}

– Restoration from the stacked xPSR value on an exception return

– Bit[0] of the vector value on an exception entry or reset.

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

11.4.1.12 Exception Mask Registers

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

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

11.4.1.13 Priority Mask Register

Name: PRIMASK

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–							

7	6	5	4	3	2	1	0
–							PRIMASK

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

- **PRIMASK**

0: No effect

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

11.4.1.14 Fault Mask Register

Name: FAULTMASK

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–							

7	6	5	4	3	2	1	0
–							FAULTMASK

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

- **FAULTMASK**

0: No effect.

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

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

11.4.1.15 Base Priority Mask Register

Name: BASEPRI

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–							

7	6	5	4	3	2	1	0
BASEPRI							

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

- **BASEPRI**

Priority mask bits:

0x0000: No effect

Nonzero: Defines the base priority for exception processing

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

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

11.4.1.16 Control Register

Name: CONTROL

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–							

7	6	5	4	3	2	1	0
–					FPCA	SPSEL	nPRIV

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

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

Indicates whether the floating-point context is currently active:

0: No floating-point context active.

1: Floating-point context active.

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

- **SPSEL: Active Stack Pointer**

Defines the current stack:

0: MSP is the current stack pointer.

1: PSP is the current stack pointer.

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

- **nPRIV: Thread Mode Privilege Level**

Defines the Thread mode privilege level:

0: Privileged.

1: Unprivileged.

Handler mode always uses the MSP, so the processor ignores explicit writes to the active stack pointer bit of the Control Register when in Handler mode. The exception entry and return mechanisms update the Control Register based on the EXC_RETURN value.

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

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

- Use the MSR instruction to set the Active stack pointer bit to 1, see “MSR” , or
- Perform an exception return to Thread mode with the appropriate EXC_RETURN value, see Table 11-10.

Note: When changing the stack pointer, the software must use an ISB instruction immediately after the MSR instruction. This ensures that instructions after the ISB execute using the new stack pointer. See "ISB" .

11.4.1.17 Exceptions and Interrupts

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

The NVIC registers control interrupt handling. See "Nested Vectored Interrupt Controller (NVIC)" for more information.

11.4.1.18 Data Types

The processor supports the following data types:

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

11.4.1.19 Cortex Microcontroller Software Interface Standard (CMSIS)

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

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

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

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

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

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

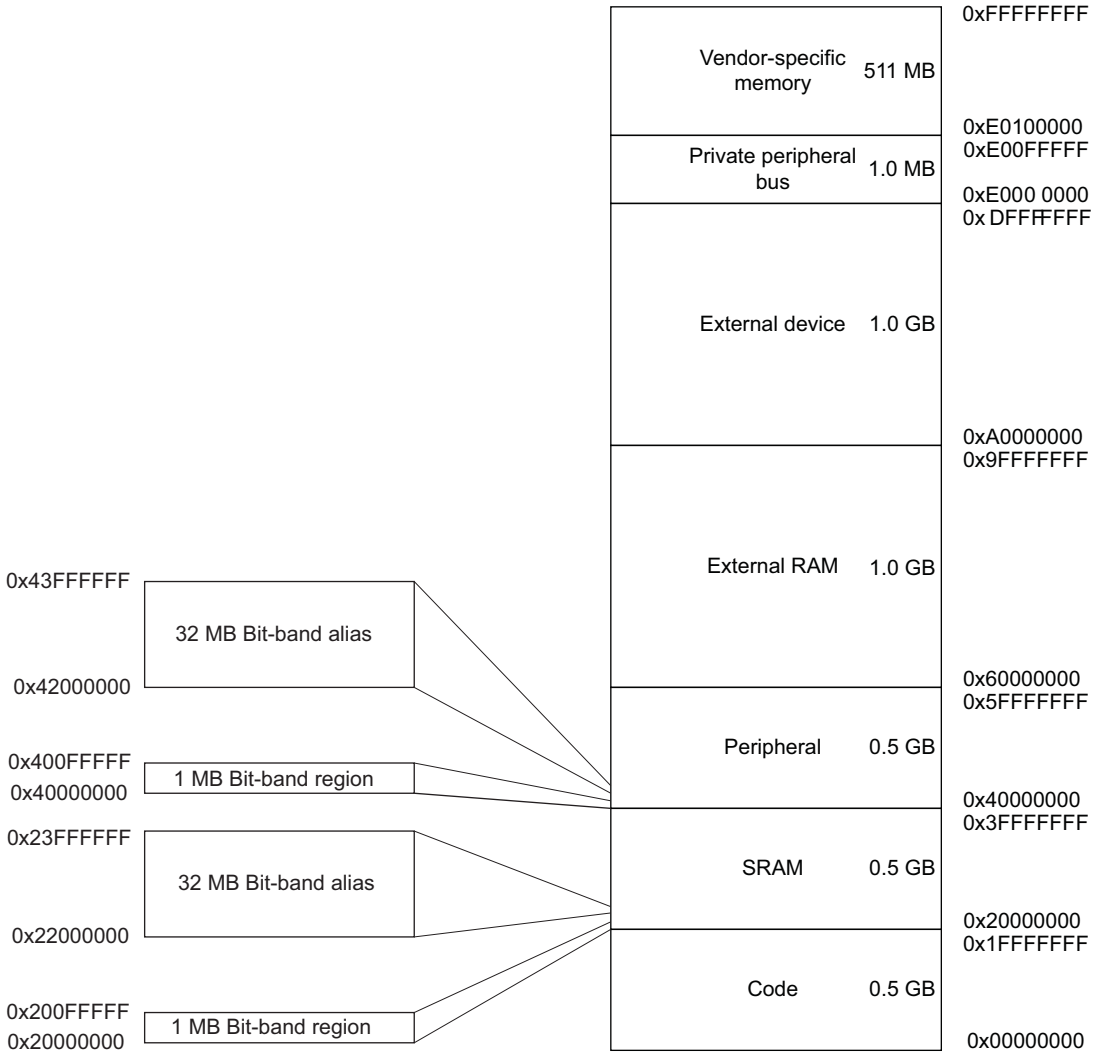
The following sections give more information about the CMSIS:

- Section 11.5.3 "Power Management Programming Hints"
- Section 11.6.2 "CMSIS Functions"
- Section 11.8.2.1 "NVIC Programming Hints".

11.4.2 Memory Model

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

Figure 11-3. Memory Map



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

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

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

11.4.2.1 Memory Regions, Types and Attributes

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

Memory Types

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

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

Additional Memory Attributes

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

11.4.2.2 Memory System Ordering of Memory Accesses

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

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

Table 11-3. Ordering of the Memory Accesses Caused by Two Instructions

A2 / A1	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.

11.4.2.3 Behavior of Memory Accesses

The following table describes the behavior of accesses to each region in the memory map.

Table 11-4. Memory Access Behavior

Address Range	Memory Region	Memory Type	XN	Description
0x00000000–0x1FFFFFFF	Code	Normal(1)	–	Executable region for program code. Data can also be put here.
0x20000000–0x3FFFFFFF	SRAM	Normal(1)	–	Executable region for data. Code can also be put here. This region includes bit band and bit band alias areas, see Table 11-6.
0x40000000–0x5FFFFFFF	Peripheral	Device(1)	XN	This region includes bit band and bit band alias areas, see Table 11-6.
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

Note: 1. See "Memory Regions, Types and Attributes" for more information.

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

The MPU can override the default memory access behavior described in this section. For more information, see "Memory Protection Unit (MPU)" .

Additional Memory Access Constraints For Caches and Shared Memory

When a system includes caches or shared memory, some memory regions have additional access constraints, and some regions are subdivided, as Table 11-5 shows.

Table 11-5. Memory Region Shareability and Cache Policies

Address Range	Memory Region	Memory Type	Shareability	Cache Policy
0x00000000–0x1FFFFFFF	Code	Normal(1)	–	WT(2)
0x20000000–0x3FFFFFFF	SRAM	Normal(1)	–	WBWA(2)
0x40000000–0x5FFFFFFF	Peripheral	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	Device(1)	–	–

Notes: 1. See "Memory Regions, Types and Attributes" for more information.
2. WT = Write through, no write allocate. WBWA = Write back, write allocate. See the "Glossary" for more information.

Instruction Prefetch and Branch Prediction

The Cortex-M4 processor:

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

11.4.2.4 Software Ordering of Memory Accesses

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

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

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

DMB

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

DSB

The *Data Synchronization Barrier* (DSB) instruction ensures that outstanding memory transactions complete before subsequent instructions execute. See "DSB" .

ISB

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

MPU Programming

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

11.4.2.5 Bit-banding

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

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

- Accesses to the 32 MB SRAM alias region map to the 1 MB SRAM bit-band region, as shown in Table 11-6.
- Accesses to the 32 MB peripheral alias region map to the 1 MB peripheral bit-band region, as shown in Table 11-7.

Table 11-6. 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 11-7. Peripheral Memory Bit-banding Regions

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

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

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

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

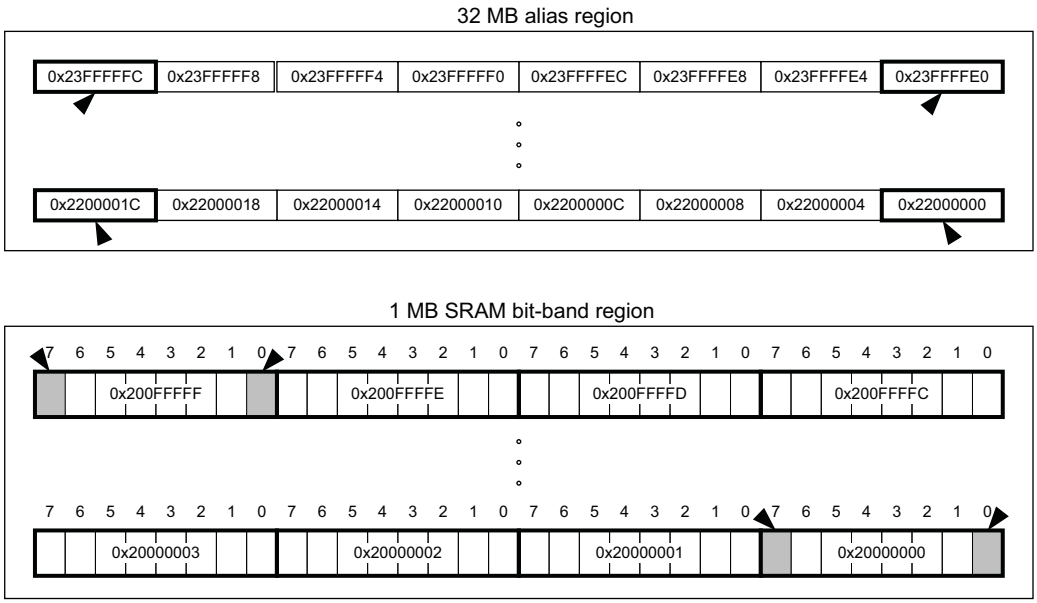
where:

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

Figure 11-4 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 11-4. Bit-band Mapping



Directly Accessing an Alias Region

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

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

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

Reading a word in the alias region:

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

Directly Accessing a Bit-band Region

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

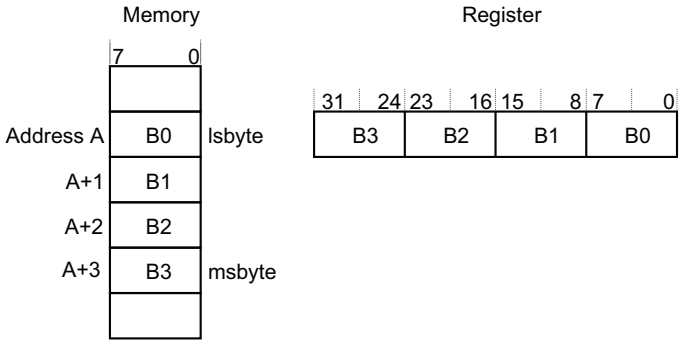
11.4.2.6 Memory Endianness

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

Little-endian Format

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

Figure 11-5. ***Little-endian Format***



11.4.2.7 Synchronization Primitives

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

A pair of synchronization primitives comprises:

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

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

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

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

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

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

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

1. Use a Load-Exclusive instruction to read the value of the location.
2. Update the value, as required.
3. Use a Store-Exclusive instruction to attempt to write the new value back to the memory location
4. Test the returned status bit. If this bit is:

   0: The read-modify-write completed successfully.

   1: No write was performed. This indicates that the value returned at step 1 might be out of date. The software must retry the read-modify-write sequence.

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

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

3. If the returned status bit from step 2 indicates that the Store-Exclusive instruction succeeded then the software has claimed the semaphore. However, if the Store-Exclusive instruction failed, another process might have claimed the semaphore after the software performed the first step.

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

The processor removes its exclusive access tag if:

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

In a multiprocessor implementation:

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

For more information about the synchronization primitive instructions, see “LDREX and STREX” and “CLREX” .

11.4.2.8 Programming Hints for the Synchronization Primitives

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

Table 11-8. CMSIS Functions for Exclusive Access Instructions

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

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

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

11.4.3 Exception Model

This section describes the exception model.

11.4.3.1 Exception States

Each exception is in one of the following states:

Inactive

The exception is not active and not pending.

Pending

The exception is waiting to be serviced by the processor.

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

Active

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

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

Active and Pending

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

11.4.3.2 Exception Types

The exception types are:

Reset

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

Non Maskable Interrupt (NMI)

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

NMIs cannot be:

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

Hard Fault

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

Memory Management Fault (MemManage)

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

Bus Fault

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

Usage Fault

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

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

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

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

SVCall

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

PendSV

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

SysTick

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

Interrupt (IRQ)

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

Table 11-9. 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	Interrupt (IRQ)	Configurable[4]	0x00000040 and above[5]	Asynchronous

Notes:
1. To simplify the software layer, the CMSIS only uses IRQ numbers and therefore uses negative values for exceptions other than interrupts. The IPSR returns the Exception number, see "Interrupt Program Status Register" .
2. See "Vector Table" for more information
3. See "System Handler Priority Registers"
4. See "Interrupt Priority Registers"
5. Increasing in steps of 4.

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

Privileged software can disable the exceptions that Table 11-9 shows as having configurable priority, see:

- "System Handler Control and State Register"
- "Interrupt Clear-enable Registers" .

For more information about hard faults, memory management faults, bus faults, and usage faults, see “Fault Handling” .

11.4.3.3 Exception Handlers

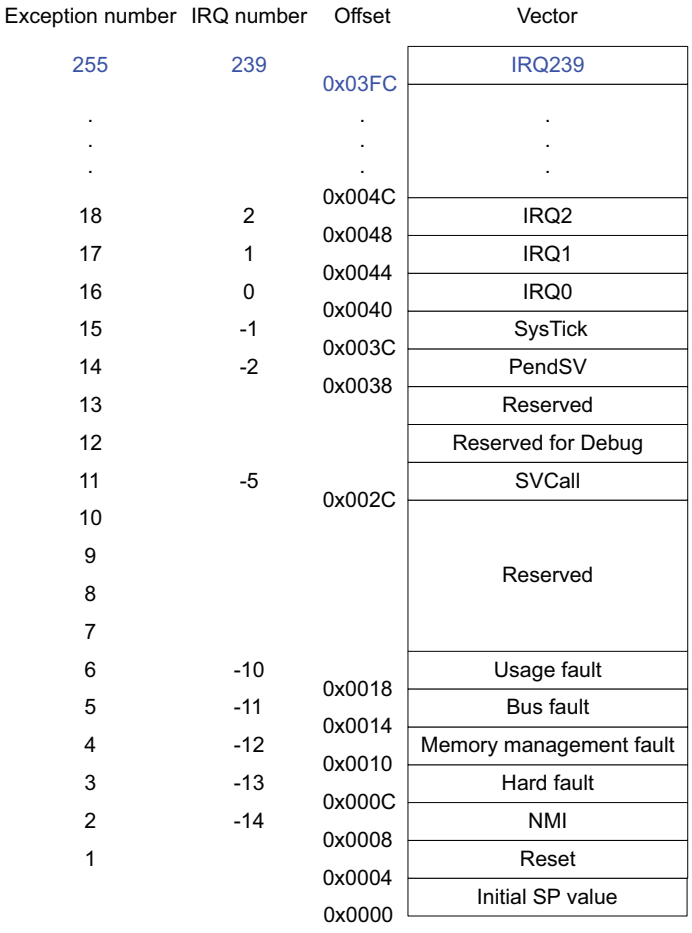
The processor handles exceptions using:

- Interrupt Service Routines (ISRs)
  Interrupts IRQ0 to IRQ46 are the exceptions handled by ISRs.
- Fault Handlers
  Hard fault, memory management fault, usage fault, bus fault are fault exceptions handled by the fault handlers.
- System Handlers
  NMI, PendSV, SVCall SysTick, and the fault exceptions are all system exceptions that are handled by system handlers.

11.4.3.4 Vector Table

The vector table contains the reset value of the stack pointer, and the start addresses, also called exception vectors, for all exception handlers. Figure 11-6 shows the order of the exception vectors in the vector table. The least-significant bit of each vector must be 1, indicating that the exception handler is Thumb code.

Figure 11-6. Vector Table



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

11.4.3.5 Exception Priorities

As Table 11-9 shows, all exceptions have an associated priority, with:

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

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

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

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

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

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

11.4.3.6 Interrupt Priority Grouping

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

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

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

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

For information about splitting the interrupt priority fields into group priority and subpriority, see "Application Interrupt and Reset Control Register" .

11.4.3.7 Exception Entry and Return

Descriptions of exception handling use the following terms:

Preemption

When the processor is executing an exception handler, an exception can preempt the exception handler if its priority is higher than the priority of the exception being handled. See "Interrupt Priority Grouping" for more information about preemption by an interrupt.

When one exception preempts another, the exceptions are called nested exceptions. See "Exception Entry" more information.

Return

This occurs when the exception handler is completed, and:

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

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

Tail-chaining

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

Late-arriving

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

Exception Entry

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

When one exception preempts another, the exceptions are nested.

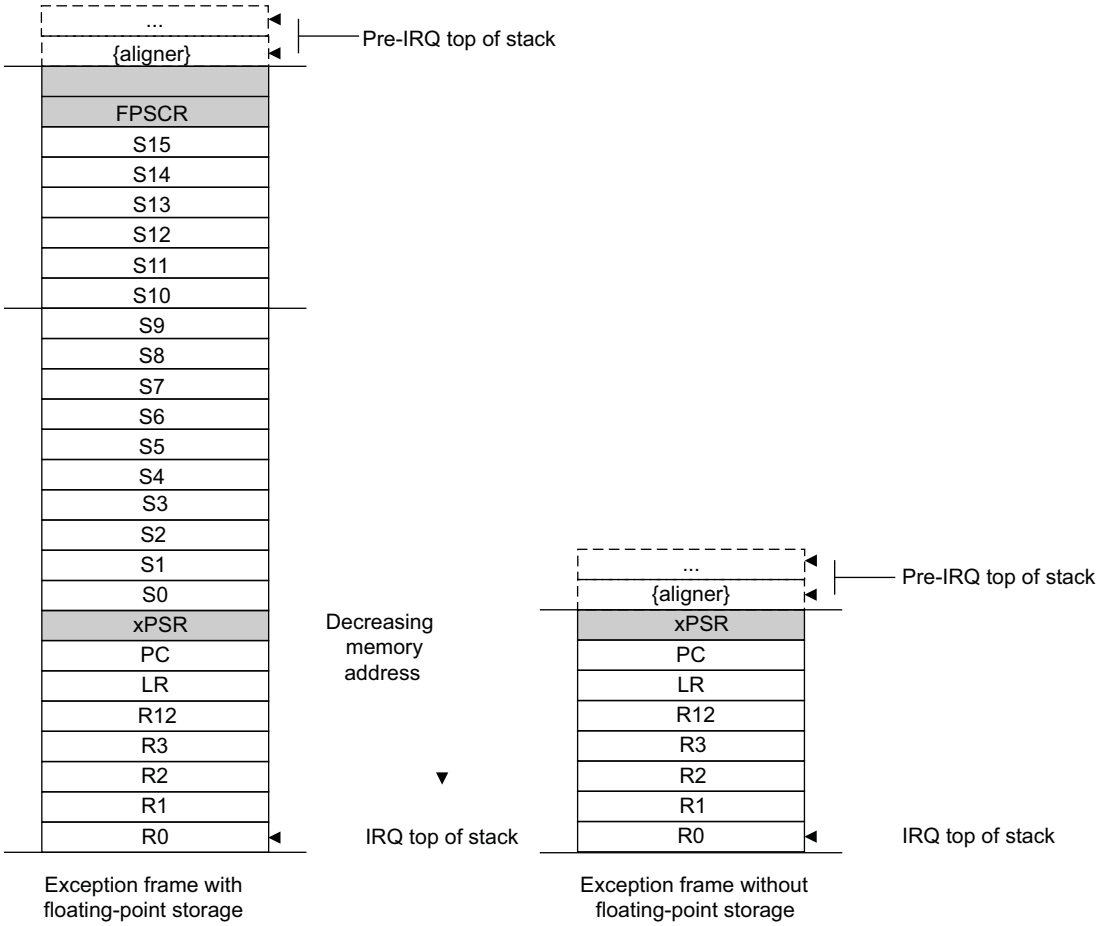
Sufficient priority means that the exception has more priority than any limits set by the mask registers, see “Exception Mask Registers” . An exception with less priority than this is pending but is not handled by the processor.

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

When using floating-point routines, the Cortex-M4 processor automatically stacks the architected floating-point state on exception entry. Figure 11-7 on page 70 shows the Cortex-M4 stack frame layout when floating-point state is preserved on the stack as the result of an interrupt or an exception.

Note: Where stack space for floating-point state is not allocated, the stack frame is the same as that of ARMv7-M implementations without an FPU. Figure 11-7 on page 70 shows this stack frame also.

Figure 11-7. Exception Stack Frame



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

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

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

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

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

Exception Return

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

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

EXC_RETURN is the value loaded into the LR on exception entry. The exception mechanism relies on this value to detect when the processor has completed an exception handler. The lowest five bits of this value provide information on the return stack and processor mode. Table 11-10 shows the EXC_RETURN values with a description of the exception return behavior.

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

Table 11-10. Exception Return Behavior

EXC_RETURN[31:0]	Description
0xFFFFFFF1	Return to Handler mode, exception return uses non-floating-point state from the MSP and execution uses MSP after return.
0xFFFFFFF9	Return to Thread mode, exception return uses state from MSP and execution uses MSP after return.
0xFFFFFFFD	Return to Thread mode, exception return uses state from the PSP and execution uses PSP after return.
0xFFFFFFE1	Return to Handler mode, exception return uses floating-point-state from MSP and execution uses MSP after return.
0xFFFFFFE9	Return to Thread mode, exception return uses floating-point state from MSP and execution uses MSP after return.
0xFFFFFFED	Return to Thread mode, exception return uses floating-point state from PSP and execution uses PSP after return.

11.4.3.8 Fault Handling

Faults are a subset of the exceptions, see "Exception Model" . The following generate a fault:

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

Fault Types

Table 11-11 shows the types of fault, the handler used for the fault, the corresponding fault status register, and the register bit that indicates that the fault has occurred. See "Configurable Fault Status Register" for more information about the fault status registers.

Table 11-11. Faults

Fault	Handler	Bit Name	Fault Status Register
Bus error on a vector read	Hard fault	VECTTBL	"Hard Fault Status Register"
Fault escalated to a hard fault		FORCED	
MPU or default memory map mismatch:	Memory management fault	–	–
on instruction access		IACCVIOL(1)	"MMFSR: Memory Management Fault Status Subregister"
on data access		DACCVIOL(2)	
during exception stacking		MSTKERR	
during exception unstacking		MUNSTKERR	
during lazy floating-point state preservation		MLSPERR(3)	

Table 11-11. Faults (Continued)

Fault	Handler	Bit Name	Fault Status Register
Bus error:	Bus fault	–	–
during exception stacking		STKERR	"BFSR: Bus Fault Status Subregister"
during exception unstacking		UNSTKERR	
during instruction prefetch		IBUSERR	
during lazy floating-point state preservation		LSPERR(3)	
Precise data bus error		PRECISERR	
Imprecise data bus error		IMPRECISERR	
Attempt to access a coprocessor	Usage fault	NOCP	"UFSR: Usage Fault Status Subregister"
Undefined instruction		UNDEFINSTR	
Attempt to enter an invalid instruction set state		INVSTATE	
Invalid EXC_RETURN value		INVPC	
Illegal unaligned load or store		UNALIGNED	
Divide By 0		DIVBYZERO	

Notes:
1. Occurs on an access to an XN region even if the processor does not include an MPU or the MPU is disabled.
2. Attempt to use an instruction set other than the Thumb instruction set, or return to a non load/store-multiple instruction with ICI continuation.
3. Only present in a Cortex-M4F device

Fault Escalation and Hard Faults

All faults exceptions except for hard fault have configurable exception priority, see "System Handler Priority Registers" . The software can disable the execution of the handlers for these faults, see "System Handler Control and State Register" .

Usually, the exception priority, together with the values of the exception mask registers, determines whether the processor enters the fault handler, and whether a fault handler can preempt another fault handler, as described in "Exception Model" .

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

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

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

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

Fault Status Registers and Fault Address Registers

The fault status registers indicate the cause of a fault. For bus faults and memory management faults, the fault address register indicates the address accessed by the operation that caused the fault, as shown in Table 11-12.

Table 11-12. Fault Status and Fault Address Registers

Handler	Status Register Name	Address Register Name	Register Description
Hard fault	SCB_HFSR	–	"Hard Fault Status Register"
Memory management fault	MMFSR	SCB_MMFAR	"MMFSR: Memory Management Fault Status Subregister" "MemManage Fault Address Register"
Bus fault	BFSR	SCB_BFAR	"BFSR: Bus Fault Status Subregister" "Bus Fault Address Register"
Usage fault	UFSR	–	"UFSR: Usage Fault Status Subregister"

Lockup

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

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

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

11.5 Power Management

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

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

The SLEEPDEEP bit of the SCR selects which sleep mode is used; see "System Control Register" .

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

11.5.1 Entering Sleep Mode

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

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

11.5.1.1 Wait for Interrupt

The *wait for interrupt* instruction, WFI, causes immediate entry to sleep mode. When the processor executes a WFI instruction it stops executing instructions and enters sleep mode. See "WFI" for more information.

11.5.1.2 Wait for Event

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

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

See "WFE" for more information.

11.5.1.3 Sleep-on-exit

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

11.5.2 Wakeup from Sleep Mode

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

11.5.2.1 Wakeup from WFI or Sleep-on-exit

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

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

11.5.2.2 Wakeup from WFE

The processor wakes up if:

- It detects an exception with sufficient priority to cause an exception entry
- It detects an external event signal. See "External Event Input"
- In a multiprocessor system, another processor in the system executes an SEV instruction.

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

11.5.2.3 External Event Input

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

11.5.3 Power Management Programming Hints

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

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

11.6 Cortex-M4 Instruction Set

11.6.1 Instruction Set Summary

The processor implements a version of the Thumb instruction set. Table 11-13 lists the supported instructions.

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

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

Table 11-13. Cortex-M4 Instructions

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

Table 11-13. Cortex-M4 Instructions (Continued)

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

Table 11-13. Cortex-M4 Instructions (Continued)

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

Table 11-13. Cortex-M4 Instructions (Continued)

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

Table 11-13. Cortex-M4 Instructions (Continued)

Mnemonic	Operands	Description	Flags
UHADD16	{Rd,} Rn, Rm	Unsigned Halving Add 16	–
UHADD8	{Rd,} Rn, Rm	Unsigned Halving Add 8	–
UHASX	{Rd,} Rn, Rm	Unsigned Halving Add and Subtract with Exchange	–
UHSAX	{Rd,} Rn, Rm	Unsigned Halving Subtract and Add with Exchange	–
UHSUB16	{Rd,} Rn, Rm	Unsigned Halving Subtract 16	–
UHSUB8	{Rd,} Rn, Rm	Unsigned Halving Subtract 8	–
UBFX	Rd, Rn, #lsb, #width	Unsigned Bit Field Extract	–
UDIV	{Rd,} Rn, Rm	Unsigned Divide	–
UMAAL	RdLo, RdHi, Rn, Rm	Unsigned Multiply Accumulate Accumulate Long ($32 \times 32 + 32 + 32$), 64-bit result	–
UMLAL	RdLo, RdHi, Rn, Rm	Unsigned Multiply with Accumulate ($32 \times 32 + 64$), 64-bit result	–
UMULL	RdLo, RdHi, Rn, Rm	Unsigned Multiply (32×32), 64-bit result	–
UQADD16	{Rd,} Rn, Rm	Unsigned Saturating Add 16	–
UQADD8	{Rd,} Rn, Rm	Unsigned Saturating Add 8	–
UQASX	{Rd,} Rn, Rm	Unsigned Saturating Add and Subtract with Exchange	–
UQSAX	{Rd,} Rn, Rm	Unsigned Saturating Subtract and Add with Exchange	–
UQSUB16	{Rd,} Rn, Rm	Unsigned Saturating Subtract 16	–
UQSUB8	{Rd,} Rn, Rm	Unsigned Saturating Subtract 8	–
USAD8	{Rd,} Rn, Rm	Unsigned Sum of Absolute Differences	–
USADA8	{Rd,} Rn, Rm, Ra	Unsigned Sum of Absolute Differences and Accumulate	–
USAT	Rd, #n, Rm {,shift #s}	Unsigned Saturate	Q
USAT16	Rd, #n, Rm	Unsigned Saturate 16	Q
UASX	{Rd,} Rn, Rm	Unsigned Add and Subtract with Exchange	GE
USUB16	{Rd,} Rn, Rm	Unsigned Subtract 16	GE
USUB8	{Rd,} Rn, Rm	Unsigned Subtract 8	GE
UXTAB	{Rd,} Rn, Rm,{,ROR #}	Rotate, extend 8 bits to 32 and Add	–
UXTAB16	{Rd,} Rn, Rm,{,ROR #}	Rotate, dual extend 8 bits to 16 and Add	–
UXTAH	{Rd,} Rn, Rm,{,ROR #}	Rotate, unsigned extend and Add Halfword	–
UXTB	{Rd,} Rm {,ROR #n}	Zero extend a byte	–
UXTB16	{Rd,} Rm {,ROR #n}	Unsigned Extend Byte 16	–
UXTH	{Rd,} Rm {,ROR #n}	Zero extend a halfword	–
VABS.F32	Sd, Sm	Floating-point Absolute	–
VADD.F32	{Sd,} Sn, Sm	Floating-point Add	–
VCMP.F32	Sd, <Sm | #0.0>	Compare two floating-point registers, or one floating-point register and zero	FPSCR
VCMPE.F32	Sd, <Sm | #0.0>	Compare two floating-point registers, or one floating-point register and zero with Invalid Operation check	FPSCR
VCVT.S32.F32	Sd, Sm	Convert between floating-point and integer	–

Table 11-13. Cortex-M4 Instructions (Continued)

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

11.6.2 CMSIS Functions

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

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

Table 11-14. CMSIS Functions to Generate some Cortex-M4 Instructions

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

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

Table 11-15. 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)

11.6.3 Instruction Descriptions

11.6.3.1 Operands

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

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

11.6.3.2 Restrictions when Using PC or SP

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

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

11.6.3.3 Flexible Second Operand

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

Operand2 can be a:

- “Constant”
- “Register with Optional Shift”

Constant

Specify an Operand2 constant in the form:

```
#constant
```

where *constant* can be:

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

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

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

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

Instruction Substitution

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

Register with Optional Shift

Specify an Operand2 register in the form:

```
Rm {, shift}
```

where:

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

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

ASR #*n*	arithmetic shift right *n* bits, $1 \le n \le 32$.
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 the user omits the shift, or specifies LSL #0, the instruction uses the value in *Rm*.

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

11.6.3.4 Shift Operations

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

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

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

ASR

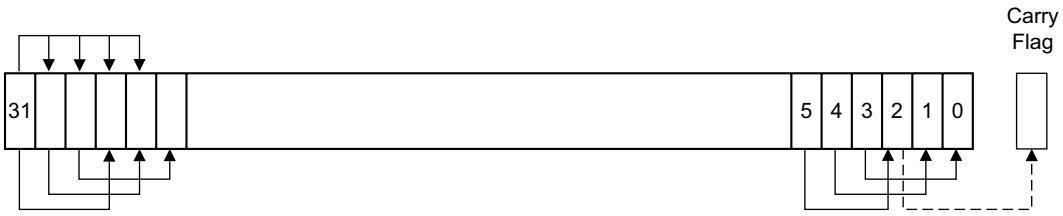
Arithmetic shift right by *n* bits moves the left-hand 32-n bits of the register, *Rm*, to the right by *n* places, into the right-hand 32-n bits of the result. And it copies the original bit[31] of the register into the left-hand *n* bits of the result. See Figure 11-8.

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

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

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

Figure 11-8. ASR #3



LSR

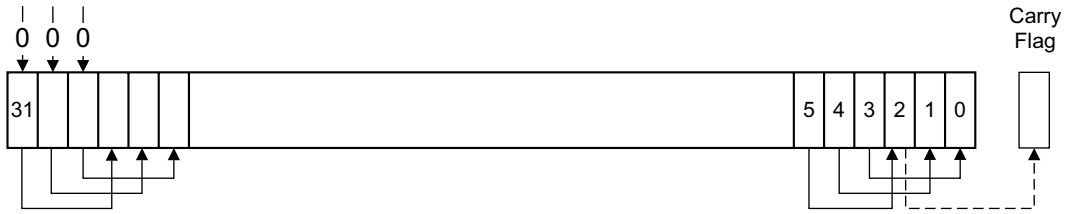
Logical shift right by n bits moves the left-hand 32-n bits of the register Rm, to the right by *n* places, into the right-hand 32-n bits of the result. And it sets the left-hand n bits of the result to 0. See Figure 11-9.

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

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

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

Figure 11-9. LSR #3



LSL

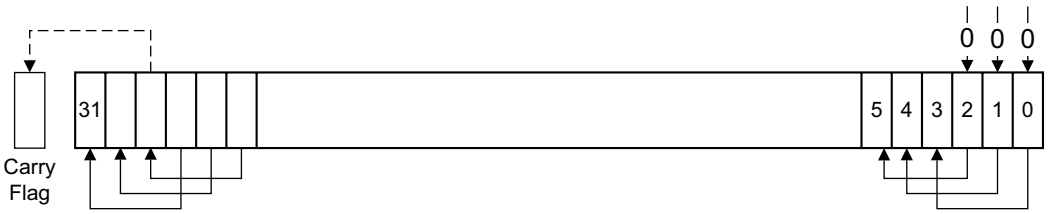
Logical shift left by *n* bits moves the right-hand 32-n bits of the register Rm, to the left by *n* places, into the left-hand 32-n bits of the result; and it sets the right-hand *n* bits of the result to 0. See Figure 11-10.

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

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

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

Figure 11-10. LSL #3



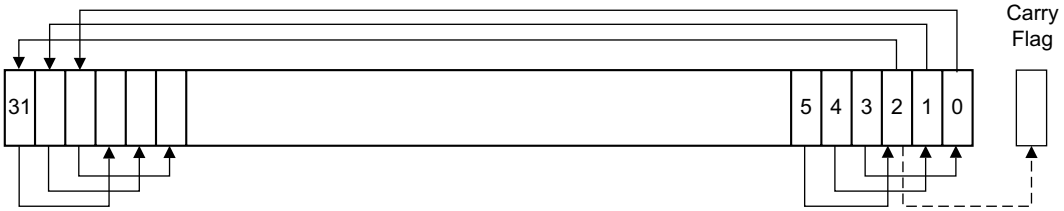
ROR

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 11-11.

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

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

Figure 11-11. ROR #3

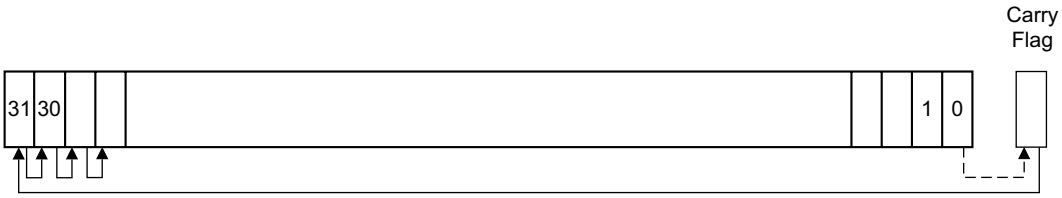


RRX

Rotate right with extend moves the bits of the register *Rm* to the right by one bit; and it copies the carry flag into bit[31] of the result. See Figure 11-12.

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

Figure 11-12. RRX



11.6.3.5 Address Alignment

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

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

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

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

Unaligned accesses are usually slower than aligned accesses. In addition, some memory regions might not support unaligned accesses. Therefore, ARM recommends that programmers ensure that accesses are aligned. To avoid accidental generation of unaligned accesses, use the UNALIGN_TRP bit in the Configuration and Control Register to trap all unaligned accesses, see “Configuration and Control Register” .

11.6.3.6 PC-relative Expressions

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

- For B, BL, CBNZ, and CBZ instructions, the value of the PC is the address of the current instruction plus 4 bytes.

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

11.6.3.7 Conditional Execution

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

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

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

Conditional execution is available by using conditional branches or by adding condition code suffixes to instructions. See Table 11-16 for a list of the suffixes to add to instructions to make them conditional instructions. The condition code suffix enables the processor to test a condition based on the flags. If the condition test of a conditional instruction fails, the instruction:

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

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

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

This section describes:

- "Condition Flags"
- "Condition Code Suffixes" .

Condition Flags

The APSR contains the following condition flags:

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

For more information about the APSR, see "Program Status Register" .

A carry occurs:

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

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

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

- If subtracting a negative value from a positive value generates a negative value.

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

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

Condition Code Suffixes

The instructions that can be conditional have an optional condition code, shown in syntax descriptions as {cond}. Conditional execution requires a preceding IT instruction. An instruction with a condition code is only executed if the condition code flags in the APSR meet the specified condition. Table 11-16 shows the condition codes to use.

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

Table 11-16 also shows the relationship between condition code suffixes and the N, Z, C, and V flags.

Table 11-16. 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.

Absolute Value

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

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

Compare and Update Value

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

```
CMP     R0, R1   ; Compare R0 and R1, setting flags
ITT     GT       ; 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
```

11.6.3.8 Instruction Width Selection

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

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

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

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

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

11.6.4 Memory Access Instructions

The table below shows the memory access instructions.

Table 11-17. Memory Access Instructions

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

11.6.4.1 ADR

Load PC-relative address.

Syntax

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

where:

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

label is a PC-relative expression. See “PC-relative Expressions” .

Operation

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

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

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

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

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

Restrictions

Rd must not be SP and must not be PC.

Condition Flags

This instruction does not change the flags.

Examples

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

11.6.4.2 LDR and STR, Immediate Offset

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

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” .

Rt is the register to load or store.

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

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

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

Operation

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

STR instructions store one or two register values to memory.

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

Offset Addressing

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

```
[Rn, #offset]
```

Pre-indexed Addressing

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

```
[Rn, #offset]!
```

Post-indexed Addressing

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

```
[Rn], #offset
```

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

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

Table 11-18. Offset Ranges

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

Restrictions

For load instructions:

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

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

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

For store instructions:

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

Condition Flags

These instructions do not change the flags.

Examples

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

11.6.4.3 LDR and STR, Register Offset

Load and Store with register offset.

Syntax

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

where:

op is one of:

- LDR Load Register.
- STR Store Register.

type is one of:

- B unsigned byte, zero extend to 32 bits on loads.
- SB signed byte, sign extend to 32 bits (LDR only).
- H unsigned halfword, zero extend to 32 bits on loads.
- SH signed halfword, sign extend to 32 bits (LDR only).
- \- omit, for word.

cond is an optional condition code, see “Conditional Execution” .

Rt is the register to load or store.

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

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

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

Operation

LDR instructions load a register with a value from memory.

STR instructions store a register value into memory.

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

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

Restrictions

In these instructions:

- *Rn* must not be PC

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

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

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

Condition Flags

These instructions do not change the flags.

Examples

```
STR    R0, [R5, R1]          ; Store value of R0 into an address equal to
                             ; sum of R5 and R1
LDRSB  R0, [R5, R1, LSL #1] ; Read byte value from an address equal to
                             ; sum of R5 and two times R1, sign extended it
                             ; to a word value and put it in R0
STR    R0, [R1, R2, LSL #2] ; Stores R0 to an address equal to sum of R1
                             ; and four times R2
```

11.6.4.4 LDR and STR, Unprivileged

Load and Store with unprivileged access.

Syntax

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

where:

op is one of:

- LDR Load Register.
- STR Store Register.

type is one of:

- B unsigned byte, zero extend to 32 bits on loads.
- SB signed byte, sign extend to 32 bits (LDR only).
- H unsigned halfword, zero extend to 32 bits on loads.
- SH signed halfword, sign extend to 32 bits (LDR only).
- - omit, for word.

cond is an optional condition code, see “Conditional Execution” .

Rt is the register to load or store.

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

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

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

Operation

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

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

Restrictions

In these instructions:

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

Condition Flags

These instructions do not change the flags.

Examples

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

11.6.4.5 LDR, PC-relative

Load register from memory.

Syntax

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

where:

type is one of:

- B unsigned byte, zero extend to 32 bits.
- SB signed byte, sign extend to 32 bits.
- H unsigned halfword, zero extend to 32 bits.
- SH signed halfword, sign extend to 32 bits.
- - omit, for word.

cond is an optional condition code, see "Conditional Execution" .

Rt is the register to load or store.

Rt2 is the second register to load or store.

label is a PC-relative expression. See "PC-relative Expressions" .

Operation

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

The value to load or store can be a byte, halfword, or word. For load instructions, bytes and halfwords can either be signed or unsigned. See "Address Alignment" .

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

Table 11-19. Offset Ranges

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

The user might have to use the .W suffix to get the maximum offset range. See "Instruction Width Selection" .

Restrictions

In these instructions:

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

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

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

Condition Flags

These instructions do not change the flags.

Examples

```
LDR     R0, LookUpTable   ; Load R0 with a word of data from an address
                          ; labelled as LookUpTable
LDRSB   R7, localdata     ; Load a byte value from an address labelled
                          ; as localdata, sign extend it to a word
                          ; value, and put it in R7
```

11.6.4.6 LDM and STM

Load and Store Multiple registers.

Syntax

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

where:

op — is one of:

- LDM — Load Multiple registers.
- STM — Store Multiple registers.

addr_mode — is any one of the following:

- IA — Increment address After each access. This is the default.
- DB — Decrement address Before each access.

cond — is an optional condition code, see “Conditional Execution” .

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

! — is an optional writeback suffix.
If ! is present, the final address, that is loaded from or stored to, is written back into *Rn*.

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

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

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

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

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

Operation

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

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

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

highest number register using the highest memory address. If the writeback suffix is specified, the value of *Rn* + 4 * (*n*-1) is written back to *Rn*.

For LDMDB, LDMEA, STMDB, and 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” for details.

Restrictions

In these instructions:

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

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

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

Condition Flags

These instructions do not change the flags.

Examples

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

Incorrect Examples

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

11.6.4.7 PUSH and POP

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

Syntax

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

where:

cond — is an optional condition code, see “Conditional Execution” .

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

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

Operation

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

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

See “LDM and STM” for more information.

Restrictions

In these instructions:

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

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

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

Condition Flags

These instructions do not change the flags.

Examples

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

11.6.4.8 LDREX and STREX

Load and Store Register Exclusive.

Syntax

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

where:

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register for the returned status.

Rt is the register to load or store.

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

offset is an optional offset applied to the value in *Rn*.

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

Operation

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

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

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

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

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

Restrictions

In these instructions:

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

Condition Flags

These instructions do not change the flags.

Examples

```
MOV     R1, #0x1            ; Initialize the ‘lock taken’ value try
LDREX   R0, [LockAddr]      ; Load the lock value
CMP     R0, #0              ; Is the lock free?
```

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

11.6.4.9 CLREX

Clear Exclusive.

Syntax

`CLREX{`*cond*`}`

where:

cond is an optional condition code, see “Conditional Execution” .

Operation

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

See “Synchronization Primitives” for more information.

Condition Flags

These instructions do not change the flags.

Examples

```
CLREX
```

11.6.5 General Data Processing Instructions

The table below shows the data processing instructions.

Table 11-20. Data Processing Instructions

Mnemonic	Description
ADC	Add with Carry
ADD	Add
ADDW	Add
AND	Logical AND
ASR	Arithmetic Shift Right
BIC	Bit Clear
CLZ	Count leading zeros
CMN	Compare Negative
CMP	Compare
EOR	Exclusive OR
LSL	Logical Shift Left
LSR	Logical Shift Right
MOV	Move
MOVT	Move Top
MOVW	Move 16-bit constant
MVN	Move NOT
ORN	Logical OR NOT
ORR	Logical OR
RBIT	Reverse Bits
REV	Reverse byte order in a word
REV16	Reverse byte order in each halfword
REVSH	Reverse byte order in bottom halfword and sign extend
ROR	Rotate Right
RRX	Rotate Right with Extend
RSB	Reverse Subtract
SADD16	Signed Add 16
SADD8	Signed Add 8
SASX	Signed Add and Subtract with Exchange
SSAX	Signed Subtract and Add with Exchange
SBC	Subtract with Carry
SHADD16	Signed Halving Add 16
SHADD8	Signed Halving Add 8
SHASX	Signed Halving Add and Subtract with Exchange
SHSAX	Signed Halving Subtract and Add with Exchange

Table 11-20. Data Processing Instructions (Continued)

Mnemonic	Description
SHSUB16	Signed Halving Subtract 16
SHSUB8	Signed Halving Subtract 8
SSUB16	Signed Subtract 16
SSUB8	Signed Subtract 8
SUB	Subtract
SUBW	Subtract
TEQ	Test Equivalence
TST	Test
UADD16	Unsigned Add 16
UADD8	Unsigned Add 8
UASX	Unsigned Add and Subtract with Exchange
USAX	Unsigned Subtract and Add with Exchange
UHADD16	Unsigned Halving Add 16
UHADD8	Unsigned Halving Add 8
UHASX	Unsigned Halving Add and Subtract with Exchange
UHSAX	Unsigned Halving Subtract and Add with Exchange
UHSUB16	Unsigned Halving Subtract 16
UHSUB8	Unsigned Halving Subtract 8
USAD8	Unsigned Sum of Absolute Differences
USADA8	Unsigned Sum of Absolute Differences and Accumulate
USUB16	Unsigned Subtract 16
USUB8	Unsigned Subtract 8

11.6.5.1 ADD, ADC, SUB, SBC, and RSB

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

Syntax

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

where:

op is one of:

ADD Add.

ADC Add with Carry.

SUB Subtract.

SBC Subtract with Carry.

RSB Reverse Subtract.

S is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see "Conditional Execution" .

cond is an optional condition code, see "Conditional Execution" .

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

Rn is the register holding the first operand.

Operand2 is a flexible second operand. See "Flexible Second Operand" for details of the options.

imm12 is any value in the range 0–4095.

Operation

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

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

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

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

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

Use ADC and SBC to synthesize multiword arithmetic, see *Multiword arithmetic examples* on.

See also "ADR" .

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

Restrictions

In these instructions:

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

- If the instruction is conditional, it must be the last instruction in the IT block

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

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

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

Condition Flags

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

Examples

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

Multiword Arithmetic Examples

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

64-bit Addition Example

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

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

96-bit Subtraction Example

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

11.6.5.2 AND, ORR, EOR, BIC, and ORN

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

Syntax

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

where:

op is one of:

AND logical AND.

ORR logical OR, or bit set.

EOR logical Exclusive OR.

BIC logical AND NOT, or bit clear.

ORN logical OR NOT.

S is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see "Conditional Execution" .

cond is an optional condition code, see "Conditional Execution" .

Rd is the destination register.

Rn is the register holding the first operand.

Operand2 is a flexible second operand. See "Flexible Second Operand" for details of the options.

Operation

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

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

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

If S is specified, these instructions:

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

Examples

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

11.6.5.3 ASR, LSL, LSR, ROR, and RRX

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

Syntax

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

where:

op is one of:

ASR Arithmetic Shift Right.

LSL Logical Shift Left.

LSR Logical Shift Right.

ROR Rotate Right.

S is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see "Conditional Execution" .

Rd is the destination register.

Rm is the register holding the value to be shifted.

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

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

ASR shift length from 1 to 32

LSL shift length from 0 to 31

LSR shift length from 1 to 32

ROR shift length from 0 to 31

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

Operation

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

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

In all these instructions, the result is written to *Rd*, but the value in register *Rm* remains unchanged. For details on what result is generated by the different instructions, see "Shift Operations" .

Restrictions

Do not use SP and do not use PC.

Condition Flags

If S is specified:

- These instructions update the N and Z flags according to the result
- The C flag is updated to the last bit shifted out, except when the shift length is 0, see "Shift Operations" .

Examples

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

11.6.5.4 CLZ

Count Leading Zeros.

Syntax

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

where:

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

Rm is the operand register.

Operation

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

This instruction does not change the flags.

Examples

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

11.6.5.5 CMP and CMN

Compare and Compare Negative.

Syntax

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

where:

cond is an optional condition code, see “Conditional Execution” .

Rn is the register holding the first operand.

Operand2 is a flexible second operand. See “Flexible Second Operand” for details of the options.

Operation

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

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

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

Restrictions

In these instructions:

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

Condition Flags

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

Examples

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

11.6.5.6 MOV and MVN

Move and Move NOT.

Syntax

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

where:

S is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see "Conditional Execution" .

cond is an optional condition code, see "Conditional Execution" .

Rd is the destination register.

Operand2 is a flexible second operand. See "Flexible Second Operand" for details of the options.

imm16 is any value in the range 0–65535.

Operation

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

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

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

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

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

See "ASR, LSL, LSR, ROR, and RRX" .

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

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

Restrictions

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

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

When *Rd* is PC in a MOV instruction:

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

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

Condition Flags

If S is specified, these instructions:

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

Examples

```
   MOVS  R11, #0x000B                              ; Write value of 0x000B to
R11, flags get updated
   MOV   R1, #0xFA05                               ; Write value of 0xFA05 to
R1, flags are not updated
   MOVS  R10, R12                                  ; Write value in R12 to R10,
flags get updated
   MOV   R3, #23      ; Write value of 23 to R3
   MOV   R8, SP       ; Write value of stack pointer to R8
   MVNS  R2, #0xF     ; Write value of 0xFFFFFFF0 (bitwise inverse of 0xF)
                      ; to the R2 and update flags.
```

11.6.5.7 MOVT

Move Top.

Syntax

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

where:

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

imm16 is a 16-bit immediate constant.

Operation

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

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

Restrictions

Rd must not be SP and must not be PC.

Condition Flags

This instruction does not change the flags.

Examples

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

11.6.5.8 REV, REV16, REVSH, and RBIT

Reverse bytes and Reverse bits.

Syntax

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

where:

op is any of:

REV Reverse byte order in a word.

REV16 Reverse byte order in each halfword independently.

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

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

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

Rn is the register holding the operand.

Operation

Use these instructions to change endianness of data:

REV converts either:

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

REV16 converts either:

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

REVSH converts either:

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

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

11.6.5.9 SADD16 and SADD8

Signed Add 16 and Signed Add 8

Syntax

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

where:

op is any of:

SADD16 Performs two 16-bit signed integer additions.

SADD8 Performs four 8-bit signed integer additions.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

Rn is the first register holding the operand.

Rm is the second register holding the operand.

Operation

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

The SADD16 instruction:

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

The SADD8 instruction:

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

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

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

11.6.5.10 SHADD16 and SHADD8

Signed Halving Add 16 and Signed Halving Add 8

Syntax

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

where:

op is any of:

SHADD16 Signed Halving Add 16.

SHADD8 Signed Halving Add 8.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

Rn is the first operand register.

Rm is the second operand register.

Operation

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

The SHADD16 instruction:

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

The SHADDB8 instruction:

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

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

11.6.5.11 SHASX and SHSAX

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

Syntax

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

where:

op is any of:

SHASX Add and Subtract with Exchange and Halving.

SHSAX Subtract and Add with Exchange and Halving.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

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

Operation

The SHASX instruction:

1. Adds the top halfword of the first operand with the bottom halfword of the second operand.
2. Writes the halfword result of the addition to the top halfword of the destination register, shifted by one bit to the right causing a divide by two, or halving.
3. Subtracts the top halfword of the second operand from the bottom highword of the first operand.
4. Writes the halfword result of the division in the bottom halfword of the destination register, shifted by one bit to the right causing a divide by two, or halving.

The SHSAX instruction:

1. Subtracts the bottom halfword of the second operand from the top highword of the first operand.
2. Writes the halfword result of the addition to the bottom halfword of the destination register, shifted by one bit to the right causing a divide by two, or halving.
3. Adds the bottom halfword of the first operand with the top halfword of the second operand.
4. Writes the halfword result of the division in the top halfword of the destination register, shifted by one bit to the right causing a divide by two, or halving.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

Examples

```
SHASX   R7, R4, R2  ; Adds top halfword of R4 to bottom halfword of R2
                    ; and writes halved result to top halfword of R7
                    ; Subtracts top halfword of R2 from bottom halfword of
                    ; R4 and writes halved result to bottom halfword of R7
SHSAX   R0, R3, R5  ; Subtracts bottom halfword of R5 from top halfword
                    ; of R3 and writes halved result to top halfword of R0
                    ; Adds top halfword of R5 to bottom halfword of R3 and
                    ; writes halved result to bottom halfword of R0.
```

11.6.5.12 SHSUB16 and SHSUB8

Signed Halving Subtract 16 and Signed Halving Subtract 8

Syntax

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

where:

op is any of:

SHSUB16 Signed Halving Subtract 16.

SHSUB8 Signed Halving Subtract 8.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

Rn is the first operand register.

Rm is the second operand register.

Operation

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

The SHSUB16 instruction:

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

The SHSUBB8 instruction:

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
SHSUB16 R1, R0     ; Subtracts halfwords in R0 from corresponding halfword
                   ; of R1 and writes to corresponding halfword of R1
SHSUB8  R4, R0, R5 ; Subtracts bytes of R0 from corresponding byte in R5,
                   ; and writes to corresponding byte in R4.
```

11.6.5.13 SSUB16 and SSUB8

Signed Subtract 16 and Signed Subtract 8

Syntax

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

where:

op is any of:

SSUB16 Performs two 16-bit signed integer subtractions.

SSUB8 Performs four 8-bit signed integer subtractions.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

Rn is the first operand register.

Rm is the second operand register.

Operation

Use these instructions to change endianness of data:

The SSUB16 instruction:

1. Subtracts each halfword from the second operand from the corresponding halfword of the first operand
2. Writes the difference result of two signed halfwords in the corresponding halfword of the destination register.

The SSUB8 instruction:

1. Subtracts each byte of the second operand from the corresponding byte of the first operand
2. Writes the difference result of four signed bytes in the corresponding byte of the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
SSUB16 R1, R0     ; Subtracts halfwords in R0 from corresponding halfword
                  ; of R1 and writes to corresponding halfword of R1
SSUB8  R4, R0, R5 ; Subtracts bytes of R5 from corresponding byte in
                  ; R0, and writes to corresponding byte of R4.
```

11.6.5.14 SASX and SSAX

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

Syntax

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

where:

op is any of:

SASX Signed Add and Subtract with Exchange.

SSAX Signed Subtract and Add with Exchange.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

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

Operation

The SASX instruction:

1. Adds the signed top halfword of the first operand with the signed bottom halfword of the second operand.
2. Writes the signed result of the addition to the top halfword of the destination register.
3. Subtracts the signed bottom halfword of the second operand from the top signed highword of the first operand.
4. Writes the signed result of the subtraction to the bottom halfword of the destination register.

The SSAX instruction:

1. Subtracts the signed bottom halfword of the second operand from the top signed highword of the first operand.
2. Writes the signed result of the addition to the bottom halfword of the destination register.
3. Adds the signed top halfword of the first operand with the signed bottom halfword of the second operand.
4. Writes the signed result of the subtraction to the top halfword of the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

Examples

```
SASX  R0, R4, R5 ; Adds top halfword of R4 to bottom halfword of R5 and
                 ; writes to top halfword of R0
                 ; Subtracts bottom halfword of R5 from top halfword of R4
                 ; and writes to bottom halfword of R0
SSAX  R7, R3, R2 ; Subtracts top halfword of R2 from bottom halfword of R3
                 ; and writes to bottom halfword of R7
                 ; Adds top halfword of R3 with bottom halfword of R2 and
                 ; writes to top halfword of R7.
```

11.6.5.15 TST and TEQ

Test bits and Test Equivalence.

Syntax

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

where

cond — is an optional condition code, see "Conditional Execution" .

Rn — is the register holding the first operand.

Operand2 — is a flexible second operand. See "Flexible Second Operand" for details of the options.

Operation

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

The TST instruction performs a bitwise AND operation on the value in *Rn* and the value of *Operand2*. This is the same as the ANDS instruction, except that it discards the result.

To test whether a bit of *Rn* is 0 or 1, use the TST instruction with an *Operand2* constant that has that bit set to 1 and all other bits cleared to 0.

The TEQ instruction performs a bitwise Exclusive OR operation on the value in *Rn* and the value of *Operand2*. This is the same as the EORS instruction, except that it discards the result.

Use the TEQ instruction to test if two values are equal without affecting the V or C flags.

TEQ is also useful for testing the sign of a value. After the comparison, the N flag is the logical Exclusive OR of the sign bits of the two operands.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions:

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

Examples

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

11.6.5.16 UADD16 and UADD8

Unsigned Add 16 and Unsigned Add 8

Syntax

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

where:

op is any of:

UADD16 Performs two 16-bit unsigned integer additions.

UADD8 Performs four 8-bit unsigned integer additions.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

Rn is the first register holding the operand.

Rm is the second register holding the operand.

Operation

Use these instructions to add 16- and 8-bit unsigned data:

The UADD16 instruction:

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

The UADD16 instruction:

1. Adds each byte of the first operand to the corresponding byte of the second operand.
2. Writes the unsigned result in the corresponding byte of the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
UADD16 R1, R0      ; Adds halfwords in R0 to corresponding halfword of R1,
                   ; writes to corresponding halfword of R1
UADD8  R4, R0, R5  ; Adds bytes of R0 to corresponding byte in R5 and
                   ; writes to corresponding byte in R4.
```

11.6.5.17 UASX and USAX

Add and Subtract with Exchange and Subtract and Add with Exchange.

Syntax

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

where:

op is one of:

UASX Add and Subtract with Exchange.

USAX Subtract and Add with Exchange.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

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

Operation

The UASX instruction:

1. Subtracts the top halfword of the second operand from the bottom halfword of the first operand.
2. Writes the unsigned result from the subtraction to the bottom halfword of the destination register.
3. Adds the top halfword of the first operand with the bottom halfword of the second operand.
4. Writes the unsigned result of the addition to the top halfword of the destination register.

The USAX instruction:

1. Adds the bottom halfword of the first operand with the top halfword of the second operand.
2. Writes the unsigned result of the addition to the bottom halfword of the destination register.
3. Subtracts the bottom halfword of the second operand from the top halfword of the first operand.
4. Writes the unsigned result from the subtraction to the top halfword of the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

Examples

```
UASX  R0, R4, R5 ; Adds top halfword of R4 to bottom halfword of R5 and
                 ; writes to top halfword of R0
                 ; Subtracts bottom halfword of R5 from top halfword of R0
                 ; and writes to bottom halfword of R0
USAX  R7, R3, R2 ; Subtracts top halfword of R2 from bottom halfword of R3
                 ; and writes to bottom halfword of R7
                 ; Adds top halfword of R3 to bottom halfword of R2 and
                 ; writes to top halfword of R7.
```

11.6.5.18 UHADD16 and UHADD8

Unsigned Halving Add 16 and Unsigned Halving Add 8

Syntax

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

where:

op is any of:

UHADD16 Unsigned Halving Add 16.

UHADD8 Unsigned Halving Add 8.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

Rn is the register holding the first operand.

Rm is the register holding the second operand.

Operation

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

The UHADD16 instruction:

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

The UHADD8 instruction:

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
UHADD16 R7, R3      ; Adds halfwords in R7 to corresponding halfword of R3
                    ; and writes halved result to corresponding halfword
                    ; in R7
UHADD8  R4, R0, R5  ; Adds bytes of R0 to corresponding byte in R5 and
                    ; writes halved result to corresponding byte in R4.
```

11.6.5.19 UHASX and UHSAX

Unsigned Halving Add and Subtract with Exchange and Unsigned Halving Subtract and Add with Exchange.

Syntax

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

where:

op is one of:

UHASX Add and Subtract with Exchange and Halving.

UHSAX Subtract and Add with Exchange and Halving.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

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

Operation

The UHASX instruction:

1. Adds the top halfword of the first operand with the bottom halfword of the second operand.
2. Shifts the result by one bit to the right causing a divide by two, or halving.
3. Writes the halfword result of the addition to the top halfword of the destination register.
4. Subtracts the top halfword of the second operand from the bottom highword of the first operand.
5. Shifts the result by one bit to the right causing a divide by two, or halving.
6. Writes the halfword result of the division in the bottom halfword of the destination register.

The UHSAX instruction:

1. Subtracts the bottom halfword of the second operand from the top highword of the first operand.
2. Shifts the result by one bit to the right causing a divide by two, or halving.
3. Writes the halfword result of the subtraction in the top halfword of the destination register.
4. Adds the bottom halfword of the first operand with the top halfword of the second operand.
5. Shifts the result by one bit to the right causing a divide by two, or halving.
6. Writes the halfword result of the addition to the bottom halfword of the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

Examples

```
UHASX  R7, R4, R2 ; Adds top halfword of R4 with bottom halfword of R2
                  ; and writes halved result to top halfword of R7
                  ; Subtracts top halfword of R2 from bottom halfword of
                  ; R7 and writes halved result to bottom halfword of R7
UHSAX  R0, R3, R5 ; Subtracts bottom halfword of R5 from top halfword of
                  ; R3 and writes halved result to top halfword of R0
                  ; Adds top halfword of R5 to bottom halfword of R3 and
                  ; writes halved result to bottom halfword of R0.
```

11.6.5.20 UHSUB16 and UHSUB8

Unsigned Halving Subtract 16 and Unsigned Halving Subtract 8

Syntax

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

where:

op is any of:

UHSUB16 Performs two unsigned 16-bit integer additions, halves the results, and writes the results to the destination register.

UHSUB8 Performs four unsigned 8-bit integer additions, halves the results, and writes the results to the destination register.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

Rn is the first register holding the operand.

Rm is the second register holding the operand.

Operation

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

The UHSUB16 instruction:

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

The UHSUB8 instruction:

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
UHSUB16 R1, R0      ; Subtracts halfwords in R0 from corresponding halfword of
                    ; R1 and writes halved result to corresponding halfword in R1
UHSUB8  R4, R0, R5  ; Subtracts bytes of R5 from corresponding byte in R0 and
                    ; writes halved result to corresponding byte in R4.
```

11.6.5.21 SEL

Select Bytes. Selects each byte of its result from either its first operand or its second operand, according to the values of the GE flags.

Syntax

```
SEL{<c>}{<q>} {<Rd>,} <Rn>, <Rm>
```

where:

c, q are standard assembler syntax fields.

Rd is the destination register.

Rn is the first register holding the operand.

Rm is the second register holding the operand.

Operation

The SEL instruction:

1. Reads the value of each bit of APSR.GE.
2. Depending on the value of APSR.GE, assigns the destination register the value of either the first or second operand register.

Restrictions

None.

Condition Flags

These instructions do not change the flags.

Examples

```
SADD16 R0, R1, R2    ; Set GE bits based on result
SEL    R0, R0, R3    ; Select bytes from R0 or R3, based on GE.
```

11.6.5.22 USAD8

Unsigned Sum of Absolute Differences

Syntax

```
USAD8{cond}{Rd,} Rn, Rm
```

where:

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

Rn is the first operand register.

Rm is the second operand register.

Operation

The USAD8 instruction:

1. Subtracts each byte of the second operand register from the corresponding byte of the first operand register.
2. Adds the absolute values of the differences together.
3. Writes the result to the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
USAD8 R1, R4, R0 ; Subtracts each byte in R0 from corresponding byte of R4
                 ; adds the differences and writes to R1
USAD8 R0, R5     ; Subtracts bytes of R5 from corresponding byte in R0,
                 ; adds the differences and writes to R0.
```

11.6.5.23 USADA8

Unsigned Sum of Absolute Differences and Accumulate

Syntax

```
USADA8{cond}{Rd,} Rn, Rm, Ra
```

where:

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

Rn is the first operand register.

Rm is the second operand register.

Ra is the register that contains the accumulation value.

Operation

The USADA8 instruction:

1. Subtracts each byte of the second operand register from the corresponding byte of the first operand register.
2. Adds the unsigned absolute differences together.
3. Adds the accumulation value to the sum of the absolute differences.
4. Writes the result to the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
USADA8 R1, R0, R6      ; Subtracts bytes in R0 from corresponding halfword of R1
                       ; adds differences, adds value of R6, writes to R1
USADA8 R4, R0, R5, R2  ; Subtracts bytes of R5 from corresponding byte in R0
                       ; adds differences, adds value of R2 writes to R4.
```

11.6.5.24 USUB16 and USUB8

Unsigned Subtract 16 and Unsigned Subtract 8

Syntax

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

where

op is any of:

USUB16 Unsigned Subtract 16.

USUB8 Unsigned Subtract 8.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

Rn is the first operand register.

Rm is the second operand register.

Operation

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

The USUB16 instruction:

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

The USUB8 instruction:

1. Subtracts each byte of the second operand register from the corresponding byte of the first operand register.
2. Writes the unsigned byte result in the corresponding byte of the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
USUB16 R1, R0  ; Subtracts halfwords in R0 from corresponding halfword of R1
               ; and writes to corresponding halfword in R1USUB8  R4, R0, R5
               ; Subtracts bytes of R5 from corresponding byte in R0 and
               ; writes to the corresponding byte in R4.
```

11.6.6 Multiply and Divide Instructions

The table below shows the multiply and divide instructions.

Table 11-21. Multiply and Divide Instructions

Mnemonic	Description
MLA	Multiply with Accumulate, 32-bit result
MLS	Multiply and Subtract, 32-bit result
MUL	Multiply, 32-bit result
SDIV	Signed Divide
SMLA[B,T]	Signed Multiply Accumulate (halfwords)
SMLAD, SMLADX	Signed Multiply Accumulate Dual
SMLAL	Signed Multiply with Accumulate (32 × 32 + 64), 64-bit result
SMLAL[B,T]	Signed Multiply Accumulate Long (halfwords)
SMLALD, SMLALDX	Signed Multiply Accumulate Long Dual
SMLAW[B|T]	Signed Multiply Accumulate (word by halfword)
SMLSD	Signed Multiply Subtract Dual
SMLSLD	Signed Multiply Subtract Long Dual
SMMLA	Signed Most Significant Word Multiply Accumulate
SMMLS, SMMLSR	Signed Most Significant Word Multiply Subtract
SMUAD, SMUADX	Signed Dual Multiply Add
SMUL[B,T]	Signed Multiply (word by halfword)
SMMUL, SMMULR	Signed Most Significant Word Multiply
SMULL	Signed Multiply (32x32), 64-bit result
SMULWB, SMULWT	Signed Multiply (word by halfword)
SMUSD, SMUSDX	Signed Dual Multiply Subtract
UDIV	Unsigned Divide
UMAAL	Unsigned Multiply Accumulate Accumulate Long (32 × 32 + 32 + 32), 64-bit result
UMLAL	Unsigned Multiply with Accumulate (32 × 32 + 64), 64-bit result
UMULL	Unsigned Multiply (32 × 32), 64-bit result

11.6.6.1 MUL, MLA, and MLS

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

Syntax

```
MUL{S}{cond} {Rd,} Rn, Rm ; Multiply
MLA{cond} Rd, Rn, Rm, Ra  ; Multiply with accumulate
MLS{cond} Rd, Rn, Rm, Ra  ; Multiply with subtract
```

where:

cond is an optional condition code, see “Conditional Execution” .

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

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

Rn, Rm are registers holding the values to be multiplied.

Ra is a register holding the value to be added or subtracted from.

Operation

The MUL instruction multiplies the values from *Rn* and *Rm*, and places the least significant 32 bits of the result in *Rd*.

The MLA instruction multiplies the values from *Rn* and *Rm*, adds the value from *Ra*, and places the least significant 32 bits of the result in *Rd*.

The MLS instruction multiplies the values from *Rn* and *Rm*, subtracts the product from the value from *Ra*, and places the least significant 32 bits of the result in *Rd*.

The results of these instructions do not depend on whether the operands are signed or unsigned.

Restrictions

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

If the S suffix is used with the MUL instruction:

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

Condition Flags

If S is specified, the MUL instruction:

- Updates the N and Z flags according to the result
- Does not affect the C and V flags.

Examples

```
MUL    R10, R2, R5     ; Multiply, R10 = R2 x R5
MLA    R10, R2, R1, R5 ; Multiply with accumulate, R10 = (R2 x R1) + R5
MULS   R0, R2, R2      ; Multiply with flag update, R0 = R2 x R2
MULLT  R2, R3, R2      ; Conditionally multiply, R2 = R3 x R2
MLS    R4, R5, R6, R7  ; Multiply with subtract, R4 = R7 - (R5 x R6)
```

11.6.6.2 UMULL, UMAAL, UMLAL

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

Syntax

```
op{cond} RdLo, RdHi, Rn, Rm
```

where:

op — is one of:

UMULL Unsigned Long Multiply.

UMAAL Unsigned Long Multiply with Accumulate Accumulate.

UMLAL Unsigned Long Multiply, with Accumulate.

cond — is an optional condition code, see “Conditional Execution” .

RdHi, RdLo — are the destination registers. For UMAAL, UMLAL and UMLAL they also hold the accumulating value.

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

Operation

These instructions interpret the values from *Rn* and *Rm* as unsigned 32-bit integers.
The UMULL instruction:

- Multiplies the two unsigned integers in the first and second operands.
- Writes the least significant 32 bits of the result in *RdLo.*
- Writes the most significant 32 bits of the result in *RdHi*.

The UMAAL instruction:

- Multiplies the two unsigned 32-bit integers in the first and second operands.
- Adds the unsigned 32-bit integer in *RdHi* to the 64-bit result of the multiplication.
- Adds the unsigned 32-bit integer in *RdLo* to the 64-bit result of the addition.
- Writes the top 32-bits of the result to *RdHi.*
- Writes the lower 32-bits of the result to *RdLo*.

The UMLAL instruction:

- Multiplies the two unsigned integers in the first and second operands.
- Adds the 64-bit result to the 64-bit unsigned integer contained in *RdHi* and *RdLo*.
- Writes the result back to *RdHi* and *RdLo*.

Restrictions

In these instructions:

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

Condition Flags

These instructions do not affect the condition code flags.

Examples

```
UMULL   R0, R4, R5, R6   ; Multiplies R5 and R6, writes the top 32 bits to R4
                         ; and the bottom 32 bits to R0
UMAAL   R3, R6, R2, R7   ; Multiplies R2 and R7, adds R6, adds R3, writes the
                         ; top 32 bits to R6, and the bottom 32 bits to R3
UMLAL   R2, R1, R3, R5   ; Multiplies R5 and R3, adds R1:R2, writes to R1:R2.
```

11.6.6.3 SMLA and SMLAW

Signed Multiply Accumulate (halfwords).

Syntax

```
op{XY}{cond} Rd, Rn, Rm
op{Y}{cond} Rd, Rn, Rm, Ra
```

where:

op — is one of:

SMLA Signed Multiply Accumulate Long (halfwords).

X and *Y* specifies which half of the source registers *Rn* and *Rm* are used as the first and second multiply operand.

If *X* is *B*, then the bottom halfword, bits [15:0], of *Rn* is used.
If *X* is *T*, then the top halfword, bits [31:16], of *Rn* is used.

If Y is *B*, then the bottom halfword, bits [15:0], of *Rm* is used.
If *Y* is *T*, then the top halfword, bits [31:16], of *Rm* is used

SMLAW Signed Multiply Accumulate (word by halfword).

Y specifies which half of the source register *Rm* is used as the second multiply operand.

If *Y* is T, then the top halfword, bits [31:16] of *Rm* is used.

If *Y* is B, then the bottom halfword, bits [15:0] of *Rm* is used.

cond — is an optional condition code, see “Conditional Execution” .

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

Rn, Rm — are registers holding the values to be multiplied.

Ra — is a register holding the value to be added or subtracted from.

Operation

The SMALBB, SMLABT, SMLATB, SMLATT instructions:

- Multiplies the specified signed halfword, top or bottom, values from *Rn* and *Rm*.
- Adds the value in *Ra* to the resulting 32-bit product.
- Writes the result of the multiplication and addition in *Rd*.

The non-specified halfwords of the source registers are ignored.

The SMLAWB and SMLAWT instructions:

- Multiply the 32-bit signed values in *Rn* with:
  - The top signed halfword of *Rm*, *T* instruction suffix.
  - The bottom signed halfword of *Rm*, *B* instruction suffix.
- Add the 32-bit signed value in *Ra* to the top 32 bits of the 48-bit product
- Writes the result of the multiplication and addition in *Rd*.

The bottom 16 bits of the 48-bit product are ignored.

If overflow occurs during the addition of the accumulate value, the instruction sets the Q flag in the APSR. No overflow can occur during the multiplication.

Restrictions

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

Condition Flags

If an overflow is detected, the `Q` flag is set.

Examples

```
SMLABB  R5, R6, R4, R1  ; Multiplies bottom halfwords of R6 and R4, adds
                        ; R1 and writes to R5
SMLATB  R5, R6, R4, R1  ; Multiplies top halfword of R6 with bottom halfword
                        ; of R4, adds R1 and writes to R5
SMLATT  R5, R6, R4, R1  ; Multiplies top halfwords of R6 and R4, adds
                        ; R1 and writes the sum to R5
SMLABT  R5, R6, R4, R1  ; Multiplies bottom halfword of R6 with top halfword
                        ; of R4, adds R1 and writes to R5
SMLABT  R4, R3, R2      ; Multiplies bottom halfword of R4 with top halfword of
                        ; R3, adds R2 and writes to R4
SMLAWB  R10, R2, R5, R3 ; Multiplies R2 with bottom halfword of R5, adds
                        ; R3 to the result and writes top 32-bits to R10
SMLAWT  R10, R2, R1, R5 ; Multiplies R2 with top halfword of R1, adds R5
                        ; and writes top 32-bits to R10.
```

11.6.6.4 SMLAD

Signed Multiply Accumulate Long Dual

Syntax

```
op{X}{cond} Rd, Rn, Rm, Ra  ;
```

where:

op is one of:

SMLAD Signed Multiply Accumulate Dual.

SMLADX Signed Multiply Accumulate Dual Reverse.

X specifies which halfword of the source register *Rn* is used as the multiply operand.
If *X* is omitted, the multiplications are bottom × bottom and top × top.
If *X* is present, the multiplications are bottom × top and top × bottom.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

Rn is the first operand register holding the values to be multiplied.

Rm the second operand register.

Ra is the accumulate value.

Operation

The SMLAD and SMLADX instructions regard the two operands as four halfword 16-bit values. The SMLAD and SMLADX instructions:

- If *X* is not present, multiply the top signed halfword value in *Rn* with the top signed halfword of *Rm* and the bottom signed halfword values in *Rn* with the bottom signed halfword of *Rm*.
- Or if *X* is present, multiply the top signed halfword value in *Rn* with the bottom signed halfword of *Rm* and the bottom signed halfword values in *Rn* with the top signed halfword of *Rm*.
- Add both multiplication results to the signed 32-bit value in *Ra*.
- Writes the 32-bit signed result of the multiplication and addition to *Rd*.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
SMLAD   R10, R2, R1, R5 ; Multiplies two halfword values in R2 with
                        ; corresponding halfwords in R1, adds R5 and
                        ; writes to R10
SMLALDX R0, R2, R4, R6  ; Multiplies top halfword of R2 with bottom
                        ; halfword of R4, multiplies bottom halfword of R2
                        ; with top halfword of R4, adds R6 and writes to
                        ; R0.
```

11.6.6.5 SMLAL and SMLALD

Signed Multiply Accumulate Long, Signed Multiply Accumulate Long (halfwords) and Signed Multiply Accumulate Long Dual.

Syntax

```
op{cond} RdLo, RdHi, Rn, Rm
op{XY}{cond} RdLo, RdHi, Rn, Rm
op{X}{cond} RdLo, RdHi, Rn, Rm
```

where:

op — is one of:

MLAL Signed Multiply Accumulate Long.

SMLAL Signed Multiply Accumulate Long (halfwords, X and Y).

X and Y specify which halfword of the source registers *Rn* and *Rm* are used as the first and second multiply operand:

If *X* is B, then the bottom halfword, bits [15:0], of *Rn* is used.
If *X* is T, then the top halfword, bits [31:16], of *Rn* is used.

If *Y* is B, then the bottom halfword, bits [15:0], of *Rm* is used.
If *Y* is T, then the top halfword, bits [31:16], of *Rm* is used.

SMLALD Signed Multiply Accumulate Long Dual.

SMLALDX Signed Multiply Accumulate Long Dual Reversed.

If the *X* is omitted, the multiplications are bottom × bottom and top × top.

If *X* is present, the multiplications are bottom × top and top × bottom.

cond — is an optional condition code, see “Conditional Execution” .

RdHi, RdLo — are the destination registers.
RdLo is the lower 32 bits and *RdHi* is the upper 32 bits of the 64-bit integer.
For SMLAL, SMLALBB, SMLALBT, SMLALTB, SMLALTT, SMLALD and SMLALDX, they also hold the accumulating value.

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

Operation

The SMLAL instruction:

- Multiplies the two’s complement signed word values from *Rn* and *Rm*.
- Adds the 64-bit value in *RdLo* and *RdHi* to the resulting 64-bit product.
- Writes the 64-bit result of the multiplication and addition in *RdLo* and *RdHi*.

The SMLALBB, SMLALBT, SMLALTB and SMLALTT instructions:

- Multiplies the specified signed halfword, Top or Bottom, values from *Rn* and *Rm*.
- Adds the resulting sign-extended 32-bit product to the 64-bit value in *RdLo* and *RdHi*.
- Writes the 64-bit result of the multiplication and addition in *RdLo* and *RdHi*.

The non-specified halfwords of the source registers are ignored.

The SMLALD and SMLALDX instructions interpret the values from *Rn* and *Rm* as four halfword two’s complement signed 16-bit integers. These instructions:

- If *X* is not present, multiply the top signed halfword value of *Rn* with the top signed halfword of *Rm* and the bottom signed halfword values of *Rn* with the bottom signed halfword of *Rm*.
- Or if *X* is present, multiply the top signed halfword value of *Rn* with the bottom signed halfword of *Rm* and the bottom signed halfword values of *Rn* with the top signed halfword of *Rm*.

- Add the two multiplication results to the signed 64-bit value in *RdLo* and *RdHi* to create the resulting 64-bit product.
- Write the 64-bit product in *RdLo* and *RdHi*.

Restrictions

In these instructions:

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

Condition Flags

These instructions do not affect the condition code flags.

Examples

```
SMLAL    R4, R5, R3, R8  ; Multiplies R3 and R8, adds R5:R4 and writes to
                         ; R5:R4
SMLALBT  R2, R1, R6, R7  ; Multiplies bottom halfword of R6 with top
                         ; halfword of R7, sign extends to 32-bit, adds
                         ; R1:R2 and writes to R1:R2
SMLALTB  R2, R1, R6, R7  ; Multiplies top halfword of R6 with bottom
                         ; halfword of R7,sign extends to 32-bit, adds R1:R2
                         ; and writes to R1:R2
SMLALD   R6, R8, R5, R1  ; Multiplies top halfwords in R5 and R1 and bottom
                         ; halfwords of R5 and R1, adds R8:R6 and writes to
                         ; R8:R6
SMLALDX  R6, R8, R5, R1  ; Multiplies top halfword in R5 with bottom
                         ; halfword of R1, and bottom halfword of R5 with
                         ; top halfword of R1, adds R8:R6 and writes to
                         ; R8:R6.
```

11.6.6.6 SMLSD and SMLSLD

Signed Multiply Subtract Dual and Signed Multiply Subtract Long Dual

Syntax

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

where:

op is one of:

SMLSD Signed Multiply Subtract Dual.

SMLSDX Signed Multiply Subtract Dual Reversed.

SMLSLD Signed Multiply Subtract Long Dual.

SMLSLDX Signed Multiply Subtract Long Dual Reversed.

SMLAW Signed Multiply Accumulate (word by halfword).

If *X* is present, the multiplications are bottom × top and top × bottom.
If the *X* is omitted, the multiplications are bottom × bottom and top × top.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

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

Ra is the register holding the accumulate value.

Operation

The SMLSD instruction interprets the values from the first and second operands as four signed halfwords. This instruction:

- Optionally rotates the halfwords of the second operand.
- Performs two signed 16 × 16-bit halfword multiplications.
- Subtracts the result of the upper halfword multiplication from the result of the lower halfword multiplication.
- Adds the signed accumulate value to the result of the subtraction.
- Writes the result of the addition to the destination register.

The SMLSLD instruction interprets the values from *Rn* and *Rm* as four signed halfwords.
This instruction:

- Optionally rotates the halfwords of the second operand.
- Performs two signed 16 × 16-bit halfword multiplications.
- Subtracts the result of the upper halfword multiplication from the result of the lower halfword multiplication.
- Adds the 64-bit value in *RdHi* and *RdLo* to the result of the subtraction.
- Writes the 64-bit result of the addition to the *RdHi* and *RdLo*.

Restrictions

In these instructions:

- Do not use SP and do not use PC.

Condition Flags

This instruction sets the Q flag if the accumulate operation overflows. Overflow cannot occur during the multiplications or subtraction.

For the Thumb instruction set, these instructions do not affect the condition code flags.

Examples

```
SMLSD   R0, R4, R5, R6 ; Multiplies bottom halfword of R4 with bottom
                       ; halfword of R5, multiplies top halfword of R4
                       ; with top halfword of R5, subtracts second from
                       ; first, adds R6, writes to R0
SMLSDX  R1, R3, R2, R0 ; Multiplies bottom halfword of R3 with top
                       ; halfword of R2, multiplies top halfword of R3
                       ; with bottom halfword of R2, subtracts second from
                       ; first, adds R0, writes to R1
SMLSLD  R3, R6, R2, R7 ; Multiplies bottom halfword of R6 with bottom
                       ; halfword of R2, multiplies top halfword of R6
                       ; with top halfword of R2, subtracts second from
                       ; first, adds R6:R3, writes to R6:R3
SMLSLDX R3, R6, R2, R7 ; Multiplies bottom halfword of R6 with top
                       ; halfword of R2, multiplies top halfword of R6
                       ; with bottom halfword of R2, subtracts second from
                       ; first, adds R6:R3, writes to R6:R3.
```

11.6.6.7 SMMLA and SMMLS

Signed Most Significant Word Multiply Accumulate and Signed Most Significant Word Multiply Subtract

Syntax

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

where:

op	is one of: SMMLA Signed Most Significant Word Multiply Accumulate. SMMLS Signed Most Significant Word Multiply Subtract. If the *X* is omitted, the multiplications are bottom × bottom and top × top.
R	is a rounding error flag. If *R* is specified, the result is rounded instead of being truncated. In this case the constant 0x80000000 is added to the product before the high word is extracted.
cond	is an optional condition code, see “Conditional Execution” .
Rd	is the destination register.
Rn, Rm	are registers holding the first and second multiply operands.
Ra	is the register holding the accumulate value.

Operation

The SMMLA instruction interprets the values from *Rn* and *Rm* as signed 32-bit words.

The SMMLA instruction:

- Multiplies the values in *Rn* and *Rm*.
- Optionally rounds the result by adding 0x80000000.
- Extracts the most significant 32 bits of the result.
- Adds the value of *Ra* to the signed extracted value.
- Writes the result of the addition in *Rd*.

The SMMLS instruction interprets the values from *Rn* and *Rm* as signed 32-bit words.

The SMMLS instruction:

- Multiplies the values in *Rn* and *Rm*.
- Optionally rounds the result by adding 0x80000000.
- Extracts the most significant 32 bits of the result.
- Subtracts the extracted value of the result from the value in *Ra*.
- Writes the result of the subtraction in *Rd*.

Restrictions

In these instructions:

- Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

Examples

```
SMMLA  R0, R4, R5, R6  ; Multiplies R4 and R5, extracts top 32 bits, adds
                       ; R6, truncates and writes to R0
SMMLAR R6, R2, R1, R4  ; Multiplies R2 and R1, extracts top 32 bits, adds
                       ; R4, rounds and writes to R6
SMMLSR R3, R6, R2, R7  ; Multiplies R6 and R2, extracts top 32 bits,
                       ; subtracts R7, rounds and writes to R3
SMMLS  R4, R5, R3, R8  ; Multiplies R5 and R3, extracts top 32 bits,
                       ; subtracts R8, truncates and writes to R4.
```

11.6.6.8 SMMUL

Signed Most Significant Word Multiply

Syntax

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

where:

op — is one of:

SMMUL Signed Most Significant Word Multiply.

R — is a rounding error flag. If *R* is specified, the result is rounded instead of being truncated. In this case the constant 0x80000000 is added to the product before the high word is extracted.

cond — is an optional condition code, see “Conditional Execution” .

Rd — is the destination register.

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

Operation

The SMMUL instruction interprets the values from *Rn* and *Rm* as two’s complement 32-bit signed integers. The SMMUL instruction:

- Multiplies the values from *Rn* and *Rm*.
- Optionally rounds the result, otherwise truncates the result.
- Writes the most significant signed 32 bits of the result in *Rd*.

Restrictions

In this instruction:

- do not use SP and do not use PC.

Condition Flags

This instruction does not affect the condition code flags.

Examples

```
SMULL   R0, R4, R5  ; Multiplies R4 and R5, truncates top 32 bits
                    ; and writes to R0
SMULLR  R6, R2      ; Multiplies R6 and R2, rounds the top 32 bits
                    ; and writes to R6.
```

11.6.6.9 SMUAD and SMUSD

Signed Dual Multiply Add and Signed Dual Multiply Subtract

Syntax

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

where:

op — is one of:

SMUAD Signed Dual Multiply Add.

SMUADX Signed Dual Multiply Add Reversed.

SMUSD Signed Dual Multiply Subtract.

SMUSDX Signed Dual Multiply Subtract Reversed.

If *X* is present, the multiplications are bottom × top and top × bottom.
If the *X* is omitted, the multiplications are bottom × bottom and top × top.

cond — is an optional condition code, see “Conditional Execution” .

Rd — is the destination register.

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

Operation

The SMUAD instruction interprets the values from the first and second operands as two signed halfwords in each operand. This instruction:

- Optionally rotates the halfwords of the second operand.
- Performs two signed 16 × 16-bit multiplications.
- Adds the two multiplication results together.
- Writes the result of the addition to the destination register.

The SMUSD instruction interprets the values from the first and second operands as two’s complement signed integers. This instruction:

- Optionally rotates the halfwords of the second operand.
- Performs two signed 16 × 16-bit multiplications.
- Subtracts the result of the top halfword multiplication from the result of the bottom halfword multiplication.
- Writes the result of the subtraction to the destination register.

Restrictions

In these instructions:

- Do not use SP and do not use PC.

Condition Flags

Sets the Q flag if the addition overflows. The multiplications cannot overflow.

Examples

```
SMUAD    R0, R4, R5  ; Multiplies bottom halfword of R4 with the bottom
                     ; halfword of R5, adds multiplication of top halfword
                     ; of R4 with top halfword of R5, writes to R0
SMUADX   R3, R7, R4  ; Multiplies bottom halfword of R7 with top halfword
                     ; of R4, adds multiplication of top halfword of R7
                     ; with bottom halfword of R4, writes to R3
SMUSD    R3, R6, R2  ; Multiplies bottom halfword of R4 with bottom halfword
                     ; of R6, subtracts multiplication of top halfword of R6
                     ; with top halfword of R3, writes to R3
SMUSDX   R4, R5, R3  ; Multiplies bottom halfword of R5 with top halfword of
                     ; R3, subtracts multiplication of top halfword of R5
                     ; with bottom halfword of R3, writes to R4.
```

11.6.6.10 SMUL and SMULW

Signed Multiply (halfwords) and Signed Multiply (word by halfword)

Syntax

```
op{XY}{cond} Rd,Rn, Rm
op{Y}{cond} Rd. Rn, Rm
```

For *SMULXY* only:

op is one of:

SMUL{*XY*} Signed Multiply (halfwords).

X and *Y* specify which halfword of the source registers *Rn* and *Rm* is used as the first and second multiply operand.
If *X* is B, then the bottom halfword, bits [15:0] of *Rn* is used.
If *X* is T, then the top halfword, bits [31:16] of *Rn* is used.If *Y* is B, then the bot tom halfword, bits [15:0], of *Rm* is used.
If *Y* is T, then the top halfword, bits [31:16], of *Rm* is used.

SMULW{Y} Signed Multiply (word by halfword).

Y specifies which halfword of the source register *Rm* is used as the second mul tiply operand.
If *Y* is B, then the bottom halfword (bits [15:0]) of *Rm* is used.
If *Y* is T, then the top halfword (bits [31:16]) of *Rm* is used.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

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

Operation

The SMULBB, SMULTB, SMULBT and SMULTT instructions interprets the values from *Rn* and *Rm* as four signed 16-bit integers. These instructions:

- Multiplies the specified signed halfword, Top or Bottom, values from *Rn* and *Rm*.
- Writes the 32-bit result of the multiplication in *Rd.*

The SMULWT and SMULWB instructions interprets the values from *Rn* as a 32-bit signed integer and *Rm* as two halfword 16-bit signed integers. These instructions:

- Multiplies the first operand and the top, T suffix, or the bottom, B suffix, halfword of the second operand.
- Writes the signed most significant 32 bits of the 48-bit result in the destination register.

Restrictions

In these instructions:

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

Examples

```
SMULBT       R0, R4, R5  ; Multiplies the bottom halfword of R4 with the
                         ; top halfword of R5, multiplies results and
                         ; writes to R0
SMULBB       R0, R4, R5  ; Multiplies the bottom halfword of R4 with the
                         ; bottom halfword of R5, multiplies results and
                         ; writes to R0
SMULTT       R0, R4, R5  ; Multiplies the top halfword of R4 with the top
                         ; halfword of R5, multiplies results and writes
                         ; to R0
SMULTB       R0, R4, R5  ; Multiplies the top halfword of R4 with the
```

```
                        ; bottom halfword of R5, multiplies results and
                        ; and writes to R0
SMULWT      R4, R5, R3  ; Multiplies R5 with the top halfword of R3,
                        ; extracts top 32 bits and writes to R4
SMULWB      R4, R5, R3  ; Multiplies R5 with the bottom halfword of R3,
                        ; extracts top 32 bits and writes to R4.
```

11.6.6.11 UMULL, UMLAL, SMULL, and SMLAL

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

Syntax

```
op{cond} RdLo, RdHi, Rn, Rm
```

where:

op — is one of:

- UMULL Unsigned Long Multiply.
- UMLAL Unsigned Long Multiply, with Accumulate.
- SMULL Signed Long Multiply.
- SMLAL Signed Long Multiply, with Accumulate.

cond — is an optional condition code, see “Conditional Execution” .

RdHi, RdLo — are the destination registers. For UMLAL and SMLAL they also hold the accumulating value.

Rn, Rm — are registers holding the operands.

Operation

The UMULL instruction interprets the values from *Rn* and *Rm* as unsigned integers. It multiplies these integers and places the least significant 32 bits of the result in *RdLo*, and the most significant 32 bits of the result in *RdHi*.

The UMLAL instruction interprets the values from *Rn* and *Rm* as unsigned integers. It multiplies these integers, adds the 64-bit result to the 64-bit unsigned integer contained in *RdHi* and *RdLo*, and writes the result back to *RdHi* and *RdLo*.

The SMULL instruction interprets the values from *Rn* and *Rm* as two’s complement signed integers. It multiplies these integers and places the least significant 32 bits of the result in *RdLo*, and the most significant 32 bits of the result in *RdHi*.

The SMLAL instruction interprets the values from *Rn* and *Rm* as two’s complement signed integers. It multiplies these integers, adds the 64-bit result to the 64-bit signed integer contained in *RdHi* and *RdLo*, and writes the result back to *RdHi* and *RdLo*.

Restrictions

In these instructions:

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

Condition Flags

These instructions do not affect the condition code flags.

Examples

```
UMULL       R0, R4, R5, R6   ; Unsigned (R4,R0) = R5 x R6
SMLAL       R4, R5, R3, R8   ; Signed (R5,R4) = (R5,R4) + R3 x R8
```

11.6.6.12 SDIV and UDIV

Signed Divide and Unsigned Divide.

Syntax

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

where:

cond is an optional condition code, see "Conditional Execution" .

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

Rn is the register holding the value to be divided.

Rm is a register holding the divisor.

Operation

SDIV performs a signed integer division of the value in *Rn* by the value in *Rm*.

UDIV performs an unsigned integer division of the value in *Rn* by the value in *Rm*.

For both instructions, if the value in *Rn* is not divisible by the value in *Rm*, the result is rounded towards zero.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

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

11.6.7 Saturating Instructions

The table below shows the saturating instructions.

Table 11-22. Saturating Instructions

Mnemonic	Description
SSAT	Signed Saturate
SSAT16	Signed Saturate Halfword
USAT	Unsigned Saturate
USAT16	Unsigned Saturate Halfword
QADD	Saturating Add
QSUB	Saturating Subtract
QSUB16	Saturating Subtract 16
QASX	Saturating Add and Subtract with Exchange
QSAX	Saturating Subtract and Add with Exchange
QDADD	Saturating Double and Add
QDSUB	Saturating Double and Subtract
UQADD16	Unsigned Saturating Add 16
UQADD8	Unsigned Saturating Add 8
UQASX	Unsigned Saturating Add and Subtract with Exchange
UQSAX	Unsigned Saturating Subtract and Add with Exchange
UQSUB16	Unsigned Saturating Subtract 16
UQSUB8	Unsigned Saturating Subtract 8

For signed *n*-bit saturation, this means that:

- If the value to be saturated is less than -2^{n-1}, the result returned is -2^{n-1}
- If the value to be saturated is greater than $2^{n-1}-1$, the result returned is $2^{n-1}-1$
- Otherwise, the result returned is the same as the value to be saturated.

For unsigned *n*-bit saturation, this means that:

- If the value to be saturated is less than 0, the result returned is 0
- If the value to be saturated is greater than $2^{n}-1$, the result returned is $2^{n}-1$
- Otherwise, the result returned is the same as the value to be saturated.

If the returned result is different from the value to be saturated, it is called *saturation*. If saturation occurs, the instruction sets the Q flag to 1 in the APSR. Otherwise, it leaves the Q flag unchanged. To clear the Q flag to 0, the MSR instruction must be used; see “MSR” .

To read the state of the Q flag, the MRS instruction must be used; see “MRS” .

11.6.7.1 SSAT and USAT

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

Syntax

```
op{cond} Rd, #n, Rm {, shift #s}
```

where:

op	is one of:
	SSAT Saturates a signed value to a signed range.
	USAT Saturates a signed value to an unsigned range.
cond	is an optional condition code, see “Conditional Execution” .
Rd	is the destination register.
n	specifies the bit position to saturate to:
n ranges from 1	n ranges from 0 to 31 for USAT.
to 32 for SSAT	
Rm	is the register containing the value to saturate.
shift #s	is an optional shift applied to Rm before saturating. It must be one of the following:
ASR #s	where s is in the range 1 to 31.
LSL #s	where *s* is in the range 0 to 31.

Operation

These instructions saturate to a signed or unsigned *n*-bit value.

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

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

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

Examples

```
        SSAT    R7, #16, R7, LSL #4  ; Logical shift left value in R7 by 4, then
                                     ; saturate it as a signed 16-bit value and
                                     ; write it back to R7
        USATNE  R0, #7, R5           ; Conditionally saturate value in R5 as an
                                     ; unsigned 7 bit value and write it to R0.
```

11.6.7.2 SSAT16 and USAT16

Signed Saturate and Unsigned Saturate to any bit position for two halfwords.

Syntax

```
op{cond} Rd, #n, Rm
```

where:

op	is one of:
	SSAT16 Saturates a signed halfword value to a signed range.
	USAT16 Saturates a signed halfword value to an unsigned range.
cond	is an optional condition code, see “Conditional Execution” .
Rd	is the destination register.
n	specifies the bit position to saturate to:
n ranges from 1	n ranges from 0 to 15 for USAT.
to 16 for SSAT	
Rm	is the register containing the value to saturate.

Operation

The SSAT16 instruction:

Saturates two signed 16-bit halfword values of the register with the value to saturate from selected by the bit position in *n*.

Writes the results as two signed 16-bit halfwords to the destination register.

The USAT16 instruction:

Saturates two unsigned 16-bit halfword values of the register with the value to saturate from selected by the bit position in *n*.

Writes the results as two unsigned halfwords in the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

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

Examples

```
SSAT16    R7, #9, R2   ; Saturates the top and bottom highwords of R2
                       ; as 9-bit values, writes to corresponding halfword
                       ; of R7
USAT16NE  R0, #13, R5  ; Conditionally saturates the top and bottom
                       ; halfwords of R5 as 13-bit values, writes to
                       ; corresponding halfword of R0.
```

11.6.7.3 QADD and QSUB

Saturating Add and Saturating Subtract, signed.

Syntax

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

where:

op is one of:

QADD Saturating 32-bit add.

QADD8 Saturating four 8-bit integer additions.

QADD16 Saturating two 16-bit integer additions.

QSUB Saturating 32-bit subtraction.

QSUB8 Saturating four 8-bit integer subtraction.

QSUB16 Saturating two 16-bit integer subtraction.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

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

Operation

These instructions add or subtract two, four or eight values from the first and second operands and then writes a signed saturated value in the destination register.

The QADD and QSUB instructions apply the specified add or subtract, and then saturate the result to the signed range $-2^{n-1} \le x \le 2^{n-1}-1$, where x is given by the number of bits applied in the instruction, 32, 16 or 8.

If the returned result is different from the value to be saturated, it is called *saturation*. If saturation occurs, the QADD and QSUB instructions set the Q flag to 1 in the APSR. Otherwise, it leaves the Q flag unchanged. The 8-bit and 16-bit QADD and QSUB instructions always leave the Q flag unchanged.

To clear the Q flag to 0, the MSR instruction must be used; see “MSR” .

To read the state of the Q flag, the MRS instruction must be used; see “MRS” .

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

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

Examples

```
QADD16   R7, R4, R2  ; Adds halfwords of R4 with corresponding halfword of
                     ; R2, saturates to 16 bits and writes to
                     ; corresponding halfword of R7
QADD8    R3, R1, R6  ; Adds bytes of R1 to the corresponding bytes of R6,
                     ; saturates to 8 bits and writes to corresponding
                     ; byte of R3
QSUB16   R4, R2, R3  ; Subtracts halfwords of R3 from corresponding
                     ; halfword of R2, saturates to 16 bits, writes to
                     ; corresponding halfword of R4
QSUB8    R4, R2, R5  ; Subtracts bytes of R5 from the corresponding byte
                     ; in R2, saturates to 8 bits, writes to corresponding
                     ; byte of R4.
```

11.6.7.4 QASX and QSAX

Saturating Add and Subtract with Exchange and Saturating Subtract and Add with Exchange, signed.

Syntax

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

where:

op is one of:

QASX Add and Subtract with Exchange and Saturate.

QSAX Subtract and Add with Exchange and Saturate.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

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

Operation

The QASX instruction:

1. Adds the top halfword of the source operand with the bottom halfword of the second operand.
2. Subtracts the top halfword of the second operand from the bottom highword of the first operand.
3. Saturates the result of the subtraction and writes a 16-bit signed integer in the range $-2^{15} \leq x \leq 2^{15} - 1$, where *x* equals 16, to the bottom halfword of the destination register.
4. Saturates the results of the sum and writes a 16-bit signed integer in the range $-2^{15} \leq x \leq 2^{15} - 1$, where *x* equals 16, to the top halfword of the destination register.

The QSAX instruction:

1. Subtracts the bottom halfword of the second operand from the top highword of the first operand.
2. Adds the bottom halfword of the source operand with the top halfword of the second operand.
3. Saturates the results of the sum and writes a 16-bit signed integer in the range $-2^{15} \leq x \leq 2^{15} - 1$, where *x* equals 16, to the bottom halfword of the destination register.
4. Saturates the result of the subtraction and writes a 16-bit signed integer in the range $-2^{15} \leq x \leq 2^{15} - 1$, where *x* equals 16, to the top halfword of the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

Examples

```
QASX   R7, R4, R2 ; Adds top halfword of R4 to bottom halfword of R2,
                  ; saturates to 16 bits, writes to top halfword of R7
                  ; Subtracts top highword of R2 from bottom halfword of
                  ; R4, saturates to 16 bits and writes to bottom halfword
                  ; of R7
QSAX   R0, R3, R5 ; Subtracts bottom halfword of R5 from top halfword of
                  ; R3, saturates to 16 bits, writes to top halfword of R0
                  ; Adds bottom halfword of R3 to top halfword of R5,
                  ; saturates to 16 bits, writes to bottom halfword of R0.
```

11.6.7.5 QDADD and QDSUB

Saturating Double and Add and Saturating Double and Subtract, signed.

Syntax

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

where:

op is one of:

QDADD Saturating Double and Add.

QDSUB Saturating Double and Subtract.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

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

Operation

The QDADD instruction:

- Doubles the second operand value.
- Adds the result of the doubling to the signed saturated value in the first operand.
- Writes the result to the destination register.

The QDSUB instruction:

- Doubles the second operand value.
- Subtracts the doubled value from the signed saturated value in the first operand.
- Writes the result to the destination register.

Both the doubling and the addition or subtraction have their results saturated to the 32-bit signed integer range $-2^{31} \leq x \leq 2^{31} - 1$. If saturation occurs in either operation, it sets the Q flag in the APSR.

Restrictions

Do not use SP and do not use PC.

Condition Flags

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

Examples

```
QDADD     R7, R4, R2   ; Doubles and saturates R4 to 32 bits, adds R2,
                       ; saturates to 32 bits, writes to R7
QDSUB     R0, R3, R5   ; Subtracts R3 doubled and saturated to 32 bits
                       ; from R5, saturates to 32 bits, writes to R0.
```

11.6.7.6 UQASX and UQSAX

Saturating Add and Subtract with Exchange and Saturating Subtract and Add with Exchange, unsigned.

Syntax

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

where:

type is one of:

UQASX Add and Subtract with Exchange and Saturate.

UQSAX Subtract and Add with Exchange and Saturate.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

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

Operation

The UQASX instruction:

1. Adds the bottom halfword of the source operand with the top halfword of the second operand.
2. Subtracts the bottom halfword of the second operand from the top highword of the first operand.
3. Saturates the results of the sum and writes a 16-bit unsigned integer in the range $0 \le x \le 2^{16} - 1$, where *x* equals 16, to the top halfword of the destination register.
4. Saturates the result of the subtraction and writes a 16-bit unsigned integer in the range $0 \le x \le 2^{16} - 1$, where *x* equals 16, to the bottom halfword of the destination register.

The UQSAX instruction:

1. Subtracts the bottom halfword of the second operand from the top highword of the first operand.
2. Adds the bottom halfword of the first operand with the top halfword of the second operand.
3. Saturates the result of the subtraction and writes a 16-bit unsigned integer in the range $0 \le x \le 2^{16} - 1$, where *x* equals 16, to the top halfword of the destination register.
4. Saturates the results of the addition and writes a 16-bit unsigned integer in the range $0 \le x \le 2^{16} - 1$, where *x* equals 16, to the bottom halfword of the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

Examples

```
UQASX   R7, R4, R2  ; Adds top halfword of R4 with bottom halfword of R2,
                    ; saturates to 16 bits, writes to top halfword of R7
                    ; Subtracts top halfword of R2 from bottom halfword of
                    ; R4, saturates to 16 bits, writes to bottom halfword of R7
UQSAX   R0, R3, R5  ; Subtracts bottom halfword of R5 from top halfword of R3,
                    ; saturates to 16 bits, writes to top halfword of R0
                    ; Adds bottom halfword of R4 to top halfword of R5
                    ; saturates to 16 bits, writes to bottom halfword of R0.
```

11.6.7.7 UQADD and UQSUB

Saturating Add and Saturating Subtract Unsigned.

Syntax

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

where:

op is one of:

UQADD8 Saturating four unsigned 8-bit integer additions.

UQADD16 Saturating two unsigned 16-bit integer additions.

UDSUB8 Saturating four unsigned 8-bit integer subtractions.

UQSUB16 Saturating two unsigned 16-bit integer subtractions.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

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

Operation

These instructions add or subtract two or four values and then writes an unsigned saturated value in the destination register.

The UQADD16 instruction:

- Adds the respective top and bottom halfwords of the first and second operands.
- Saturates the result of the additions for each halfword in the destination register to the unsigned range $0 \le x \le 2^{16}-1$, where x is 16.

The UQADD8 instruction:

- Adds each respective byte of the first and second operands.
- Saturates the result of the addition for each byte in the destination register to the unsigned range $0 \le x \le 2^{8}-1$, where x is 8.

The UQSUB16 instruction:

- Subtracts both halfwords of the second operand from the respective halfwords of the first operand.
- Saturates the result of the differences in the destination register to the unsigned range $0 \le x \le 2^{16}-1$, where x is 16.

The UQSUB8 instructions:

- Subtracts the respective bytes of the second operand from the respective bytes of the first operand.
- Saturates the results of the differences for each byte in the destination register to the unsigned range $0 \le x \le 2^{8}-1$, where x is 8.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

Examples

```
UQADD16  R7, R4, R2   ; Adds halfwords in R4 to corresponding halfword in R2,
                      ; saturates to 16 bits, writes to corresponding halfword of R7
UQADD8   R4, R2, R5   ; Adds bytes of R2 to corresponding byte of R5, saturates
                      ; to 8 bits, writes to corresponding bytes of R4
UQSUB16  R6, R3, R0   ; Subtracts halfwords in R0 from corresponding halfword
                      ; in R3, saturates to 16 bits, writes to corresponding
                      ; halfword in R6
UQSUB8   R1, R5, R6   ; Subtracts bytes in R6 from corresponding byte of R5,
                      ; saturates to 8 bits, writes to corresponding byte of R1.
```

11.6.8 Packing and Unpacking Instructions

The table below shows the instructions that operate on packing and unpacking data.

Table 11-23. Packing and Unpacking Instructions

Mnemonic	Description
PKH	Pack Halfword
SXTAB	Extend 8 bits to 32 and add
SXTAB16	Dual extend 8 bits to 16 and add
SXTAH	Extend 16 bits to 32 and add
SXTB	Sign extend a byte
SXTB16	Dual extend 8 bits to 16 and add
SXTH	Sign extend a halfword
UXTAB	Extend 8 bits to 32 and add
UXTAB16	Dual extend 8 bits to 16 and add
UXTAH	Extend 16 bits to 32 and add
UXTB	Zero extend a byte
UXTB16	Dual zero extend 8 bits to 16 and add
UXTH	Zero extend a halfword

11.6.8.1 PKHBT and PKHTB

Pack Halfword

Syntax

```
op{cond} {Rd}, Rn, Rm {, LSL #imm}
op{cond} {Rd}, Rn, Rm {, ASR #imm}
```

where:

op — is one of:

PKHBT Pack Halfword, bottom and top with shift.

PKHTB Pack Halfword, top and bottom with shift.

cond — is an optional condition code, see “Conditional Execution” .

Rd — is the destination register.

Rn — is the first operand register

Rm — is the second operand register holding the value to be optionally shifted.

imm — is the shift length. The type of shift length depends on the instruction:
For PKHBT
LSL a left shift with a shift length from 1 to 31, 0 means no shift.
For PKHTB
ASR an arithmetic shift right with a shift length from 1 to 32,
a shift of 32-bits is encoded as 0b00000.

Operation

The PKHBT instruction:

1. Writes the value of the bottom halfword of the first operand to the bottom halfword of the destination register.
2. If shifted, the shifted value of the second operand is written to the top halfword of the destination register.

The PKHTB instruction:

1. Writes the value of the top halfword of the first operand to the top halfword of the destination register.
2. If shifted, the shifted value of the second operand is written to the bottom halfword of the destination register.

Restrictions

Rd must not be SP and must not be PC.

Condition Flags

This instruction does not change the flags.

Examples

```
PKHBT   R3, R4, R5 LSL #0  ; Writes bottom halfword of R4 to bottom halfword of
                           ; R3, writes top halfword of R5, unshifted, to top
                           ; halfword of R3
PKHTB   R4, R0, R2 ASR #1  ; Writes R2 shifted right by 1 bit to bottom halfword
                           ; of R4, and writes top halfword of R0 to top
                           ; halfword of R4.
```

11.6.8.2 SXT and UXT

Sign extend and Zero extend.

Syntax

```
op{cond} {Rd,} Rm {, ROR #n}
op{cond} {Rd}, Rm {, ROR #n}
```

where:

op is one of:

SXTB Sign extends an 8-bit value to a 32-bit value.

SXTH Sign extends a 16-bit value to a 32-bit value.

SXTB16 Sign extends two 8-bit values to two 16-bit values.

UXTB Zero extends an 8-bit value to a 32-bit value.

UXTH Zero extends a 16-bit value to a 32-bit value.

UXTB16 Zero extends two 8-bit values to two 16-bit values.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

Rm is the register holding the value to extend.

ROR #*n* is one of:

ROR #8 Value from *Rm* is rotated right 8 bits.

Operation

These instructions do the following:

1. Rotate the value from *Rm* right by 0, 8, 16 or 24 bits.
2. Extract bits from the resulting value:
   - SXTB extracts bits[7:0] and sign extends to 32 bits.
   - UXTB extracts bits[7:0] and zero extends to 32 bits.
   - SXTH extracts bits[15:0] and sign extends to 32 bits.
   - UXTH extracts bits[15:0] and zero extends to 32 bits.
   - SXTB16 extracts bits[7:0] and sign extends to 16 bits, and extracts bits [23:16] and sign extends to 16 bits.
   - UXTB16 extracts bits[7:0] and zero extends to 16 bits, and extracts bits [23:16] and zero extends to 16 bits.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the flags.

Examples

```
SXTH  R4, R6, ROR #16  ; Rotates R6 right by 16 bits, obtains bottom halfword of
                       ; of result, sign extends to 32 bits and writes to R4
UXTB  R3, R10          ; Extracts lowest byte of value in R10, zero extends, and
                       ; writes to R3.
```

11.6.8.3 SXTA and UXTA

Signed and Unsigned Extend and Add

Syntax

```
op{cond} {Rd,} Rn, Rm {, ROR #n}
op{cond} {Rd,} Rn, Rm {, ROR #n}
```

where:

op is one of:

SXTAB Sign extends an 8-bit value to a 32-bit value and add.

SXTAH Sign extends a 16-bit value to a 32-bit value and add.

SXTAB16 Sign extends two 8-bit values to two 16-bit values and add.

UXTAB Zero extends an 8-bit value to a 32-bit value and add.

UXTAH Zero extends a 16-bit value to a 32-bit value and add.

UXTAB16 Zero extends two 8-bit values to two 16-bit values and add.

cond is an optional condition code, see “Conditional Execution” .

Rd is the destination register.

Rn is the first operand register.

Rm is the register holding the value to rotate and extend.

ROR *#n* is one of:

ROR #8 Value from *Rm* is rotated right 8 bits.

ROR #16 Value from *Rm* is rotated right 16 bits.

ROR #24 Value from *Rm* is rotated right 24 bits.

If ROR *#n* is omitted, no rotation is performed.

Operation

These instructions do the following:

1. Rotate the value from *Rm* right by 0, 8, 16 or 24 bits.
2. Extract bits from the resulting value:
   - SXTAB extracts bits[7:0] from *Rm* and sign extends to 32 bits.
   - UXTAB extracts bits[7:0] from *Rm* and zero extends to 32 bits.
   - SXTAH extracts bits[15:0] from *Rm* and sign extends to 32 bits.
   - UXTAH extracts bits[15:0] from *Rm* and zero extends to 32 bits.
   - SXTAB16 extracts bits[7:0] from *Rm* and sign extends to 16 bits, and extracts bits [23:16] from *Rm* and sign extends to 16 bits.
   - UXTAB16 extracts bits[7:0] from *Rm* and zero extends to 16 bits, and extracts bits [23:16] from *Rm* and zero extends to 16 bits.
3. Adds the signed or zero extended value to the word or corresponding halfword of *Rn* and writes the result in *Rd*.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the flags.

Examples

```
SXTAH  R4, R8, R6, ROR #16 ; Rotates R6 right by 16 bits, obtains bottom
                           ; halfword, sign extends to 32 bits, adds
                           ; R8,and writes to R4
UXTAB  R3, R4, R10         ; Extracts bottom byte of R10 and zero extends
                           ; to 32 bits, adds R4, and writes to R3.
```

11.6.9 Bitfield Instructions

The table below shows the instructions that operate on adjacent sets of bits in registers or bitfields.

Table 11-24. Packing and Unpacking Instructions

Mnemonic	Description
BFC	Bit Field Clear
BFI	Bit Field Insert
SBFX	Signed Bit Field Extract
SXTB	Sign extend a byte
SXTH	Sign extend a halfword
UBFX	Unsigned Bit Field Extract
UXTB	Zero extend a byte
UXTH	Zero extend a halfword

11.6.9.1 BFC and BFI

Bit Field Clear and Bit Field Insert.

Syntax

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

where:

cond — is an optional condition code, see “Conditional Execution” .

Rd — is the destination register.

Rn — is the source register.

lsb — is the position of the least significant bit of the bitfield. *lsb* must be in the range 0 to 31.

width — is the width of the bitfield and must be in the range 1 to 32-*lsb*.

Operation

BFC clears a bitfield in a register. It clears *width* bits in *Rd*, starting at the low bit position *lsb*. Other bits in *Rd* are unchanged.

BFI copies a bitfield into one register from another register. It replaces *width* bits in *Rd* starting at the low bit position *lsb*, with *width* bits from *Rn* starting at bit[0]. Other bits in *Rd* are unchanged.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the flags.

Examples

```
BFC   R4, #8, #12       ; Clear bit 8 to bit 19 (12 bits) of R4 to 0
BFI   R9, R2, #8, #12   ; Replace bit 8 to bit 19 (12 bits) of R9 with
                        ; bit 0 to bit 11 from R2.
```

11.6.9.2 SBFX and UBFX

Signed Bit Field Extract and Unsigned Bit Field Extract.

Syntax

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

where:

cond — is an optional condition code, see "Conditional Execution" .

Rd — is the destination register.

Rn — is the source register.

lsb — is the position of the least significant bit of the bitfield. *lsb* must be in the range 0 to 31.

width — is the width of the bitfield and must be in the range 1 to 32-*lsb*.

Operation

SBFX extracts a bitfield from one register, sign extends it to 32 bits, and writes the result to the destination register.

UBFX extracts a bitfield from one register, zero extends it to 32 bits, and writes the result to the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the flags.

Examples

```
SBFX  R0, R1, #20, #4  ; Extract bit 20 to bit 23 (4 bits) from R1 and sign
                       ; extend to 32 bits and then write the result to R0.
UBFX  R8, R11, #9, #10 ; Extract bit 9 to bit 18 (10 bits) from R11 and zero
                       ; extend to 32 bits and then write the result to R8.
```

11.6.9.3 SXT and UXT

Sign extend and Zero extend.

Syntax

```
SXTextend{cond} {Rd,} Rm {, ROR #n}
UXTextend{cond} {Rd}, Rm {, ROR #n}
```

where:

extend — is one of:

B Extends an 8-bit value to a 32-bit value.

H Extends a 16-bit value to a 32-bit value.

cond — is an optional condition code, see “Conditional Execution” .

Rd — is the destination register.

Rm — is the register holding the value to extend.

ROR *#n* — is one of:

ROR #8 Value from *Rm* is rotated right 8 bits.

ROR #16 Value from *Rm* is rotated right 16 bits.

ROR #24 Value from *Rm* is rotated right 24 bits.

If ROR *#n* is omitted, no rotation is performed.

Operation

These instructions do the following:

1. Rotate the value from *Rm* right by 0, 8, 16 or 24 bits.
2. Extract bits from the resulting value:
   - SXTB extracts bits[7:0] and sign extends to 32 bits.
   - UXTB extracts bits[7:0] and zero extends to 32 bits.
   - SXTH extracts bits[15:0] and sign extends to 32 bits.
   - UXTH extracts bits[15:0] and zero extends to 32 bits.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the flags.

Examples

```
SXTH  R4, R6, ROR #16  ; Rotate R6 right by 16 bits, then obtain the lower
                       ; halfword of the result and then sign extend to
                       ; 32 bits and write the result to R4.
UXTB  R3, R10          ; Extract lowest byte of the value in R10 and zero
                       ; extend it, and write the result to R3.
```

11.6.10 Branch and Control Instructions

The table below shows the branch and control instructions.

Table 11-25. Branch and Control Instructions

Mnemonic	Description
B	Branch
BL	Branch with Link
BLX	Branch indirect with Link
BX	Branch indirect
CBNZ	Compare and Branch if Non Zero
CBZ	Compare and Branch if Zero
IT	If-Then
TBB	Table Branch Byte
TBH	Table Branch Halfword

11.6.10.1 B, BL, BX, and BLX

Branch instructions.

Syntax

```
B{cond} label
BL{cond} label
BX{cond} Rm
BLX{cond} Rm
```

where:

B	is branch (immediate).
BL	is branch with link (immediate).
BX	is branch indirect (register).
BLX	is branch indirect with link (register).
cond	is an optional condition code, see "Conditional Execution" .
label	is a PC-relative expression. See "PC-relative Expressions" .
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.

Operation

All these instructions cause a branch to *label*, or to the address indicated in *Rm*. In addition:

- The BL and BLX instructions write the address of the next instruction to LR (the link register, R14).
- The BX and BLX instructions result in a UsageFault exception if bit[0] of *Rm* is 0.

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" .

The table below shows the ranges for the various branch instructions.

Table 11-26. 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

The .W suffix might be used to get the maximum branch range. See "Instruction Width Selection" .

Restrictions

The restrictions are:

- Do not use PC in the BLX instruction
- For BX and BLX, bit[0] of *Rm* must be 1 for correct execution but a branch occurs to the target address created by changing bit[0] to 0
- When any of these instructions is inside an IT block, it must be the last instruction of the IT block.

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.

Condition Flags

These instructions do not change the flags.

Examples

```
B       loopA    ; Branch to loopA
BLE     ng       ; Conditionally branch to label ng
B.W     target   ; Branch to target within 16MB range
BEQ     target   ; Conditionally branch to target
BEQ.W   target   ; Conditionally branch to target within 1MB
BL      funC     ; Branch with link (Call) to function funC, return address
                 ; stored in LR
BX      LR       ; Return from function call
BXNE    R0       ; Conditionally branch to address stored in R0
BLX     R0       ; Branch with link and exchange (Call) to a address stored in R0.
```

11.6.10.2 CBZ and CBNZ

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

Syntax

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

where:

Rn is the register holding the operand.

label is the branch destination.

Operation

Use the CBZ or CBNZ instructions to avoid changing the condition code flags and to reduce the number of instructions.

CBZ Rn, label does not change condition flags but is otherwise equivalent to:

```
CMP     Rn, #0
BEQ     label
```

CBNZ Rn, label does not change condition flags but is otherwise equivalent to:

```
CMP     Rn, #0
BNE     label
```

Restrictions

The restrictions are:

- *Rn* must be in the range of R0 to R7
- The branch destination must be within 4 to 130 bytes after the instruction
- These instructions must not be used inside an IT block.

Condition Flags

These instructions do not change the flags.

Examples

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

11.6.10.3 IT

If-Then condition instruction.

Syntax

```
IT{x{y{z}}} cond
```

where:

x specifies the condition switch for the second instruction in the IT block.

y specifies the condition switch for the third instruction in the IT block.

z specifies the condition switch for the fourth instruction in the IT block.

cond specifies the condition for the first instruction in the IT block.

The condition switch for the second, third and fourth instruction in the IT block can be either:

T Then. Applies the condition *cond* to the instruction.

E Else. Applies the inverse condition of *cond* to the instruction.

It is possible to use AL (the *always* condition) for *cond* in an IT instruction. If this is done, all of the instructions in the IT block must be unconditional, and each of *x*, *y*, and *z* must be T or omitted but not E.

Operation

The IT instruction makes up to four following instructions conditional. The conditions can be all the same, or some of them can be the logical inverse of the others. The conditional instructions following the IT instruction form the *IT block*.

The instructions in the IT block, including any branches, must specify the condition in the {*cond*} part of their syntax.

The assembler might be able to generate the required IT instructions for conditional instructions automatically, so that the user does not have to write them. See the assembler documentation for details.

A BKPT instruction in an IT block is always executed, even if its condition fails.

Exceptions can be taken between an IT instruction and the corresponding IT block, or within an IT block. Such an exception results in entry to the appropriate exception handler, with suitable return information in LR and stacked PSR.

Instructions designed for use for exception returns can be used as normal to return from the exception, and execution of the IT block resumes correctly. This is the only way that a PC-modifying instruction is permitted to branch to an instruction in an IT block.

Restrictions

The following instructions are not permitted in an IT block:

- IT
- CBZ and CBNZ
- CPSID and CPSIE.

Other restrictions when using an IT block are:

- A branch or any instruction that modifies the PC must either be outside an IT block or must be the last instruction inside the IT block. These are:
  - ADD PC, PC, Rm
  - MOV PC, Rm
  - B, BL, BX, BLX
  - Any LDM, LDR, or POP instruction that writes to the PC
  - TBB and TBH
- Do not branch to any instruction inside an IT block, except when returning from an exception handler

- All conditional instructions except B*cond* must be inside an IT block. B*cond* can be either outside or inside an IT block but has a larger branch range if it is inside one
- Each instruction inside the IT block must specify a condition code suffix that is either the same or logical inverse as for the other instructions in the block.

Your assembler might place extra restrictions on the use of IT blocks, such as prohibiting the use of assembler directives within them.

Condition Flags

This instruction does not change the flags.

Example

```
ITTE   NE           ; Next 3 instructions are conditional
ANDNE  R0, R0, R1   ; ANDNE does not update condition flags
ADDSNE R2, R2, #1   ; ADDSNE updates condition flags
MOVEQ  R2, R3       ; Conditional move


CMP    R0, #9       ; Convert R0 hex value (0 to 15) into ASCII
                    ; ('0'-'9', 'A'-'F')
ITE    GT           ; Next 2 instructions are conditional
ADDGT  R1, R0, #55  ; Convert 0xA -> 'A'
ADDLE  R1, R0, #48  ; Convert 0x0 -> '0'


IT     GT           ; IT block with only one conditional instruction
ADDGT  R1, R1, #1   ; Increment R1 conditionally


ITTEE  EQ           ; Next 4 instructions are conditional
MOVEQ  R0, R1       ; Conditional move
ADDEQ  R2, R2, #10  ; Conditional add
ANDNE  R3, R3, #1   ; Conditional AND
BNE.W  dloop        ; Branch instruction can only be used in the last
                    ; instruction of an IT block


IT     NE           ; Next instruction is conditional
ADD    R0, R0, R1   ; Syntax error: no condition code used in IT block
```

11.6.10.4 TBB and TBH

Table Branch Byte and Table Branch Halfword.

Syntax

```
TBB [Rn, Rm]
TBH [Rn, Rm, LSL #1]
```

where:

Rn is the register containing the address of the table of branch lengths.

If *Rn* is PC, then the address of the table is the address of the byte immediately following the TBB or TBH instruction.

Rm is the index register. This contains an index into the table. For halfword tables, LSL #1 doubles the value in *Rm* to form the right offset into the table.

Operation

These instructions cause a PC-relative forward branch using a table of single byte offsets for TBB, or halfword offsets for TBH. *Rn* provides a pointer to the table, and *Rm* supplies an index into the table. For TBB the branch offset is twice the unsigned value of the byte returned from the table. and for TBH the branch offset is twice the unsigned value of the halfword returned from the table. The branch occurs to the address at that offset from the address of the byte immediately after the TBB or TBH instruction.

Restrictions

The restrictions are:

- *Rn* must not be SP
- *Rm* must not be SP and must not be PC
- When any of these instructions is used inside an IT block, it must be the last instruction of the IT block.

Condition Flags

These instructions do not change the flags.

Examples

```
        ADR.W  R0, BranchTable_Byte
        TBB    [R0, R1]      ; R1 is the index, R0 is the base address of the
                             ; branch table
    Case1
    ; an instruction sequence follows
    Case2
    ; an instruction sequence follows
    Case3
    ; an instruction sequence follows
    BranchTable_Byte
        DCB    0                  ; Case1 offset calculation
        DCB    ((Case2-Case1)/2)  ; Case2 offset calculation
        DCB    ((Case3-Case1)/2)  ; Case3 offset calculation


        TBH    [PC, R1, LSL #1]   ; R1 is the index, PC is used as base of the
                                  ; branch table
    BranchTable_H
        DCI    ((CaseA - BranchTable_H)/2)  ; CaseA offset calculation
        DCI    ((CaseB - BranchTable_H)/2)  ; CaseB offset calculation
        DCI    ((CaseC - BranchTable_H)/2)  ; CaseC offset calculation

    CaseA
    ; an instruction sequence follows
    CaseB
    ; an instruction sequence follows
    CaseC
    ; an instruction sequence follows
```

11.6.11 Floating-point Instructions

The table below shows the floating-point instructions.

These instructions are only available if the FPU is included, and enabled, in the system. See “Enabling the FPU” for information about enabling the floating-point unit.

Table 11-27. Floating-point Instructions

Mnemonic	Description
VABS	Floating-point Absolute
VADD	Floating-point Add
VCMP	Compare two floating-point registers, or one floating-point register and zero
VCMPE	Compare two floating-point registers, or one floating-point register and zero with Invalid Operation check
VCVT	Convert between floating-point and integer
VCVT	Convert between floating-point and fixed point
VCVTR	Convert between floating-point and integer with rounding
VCVTB	Converts half-precision value to single-precision
VCVTT	Converts single-precision register to half-precision
VDIV	Floating-point Divide
VFMA	Floating-point Fused Multiply Accumulate
VFNMA	Floating-point Fused Negate Multiply Accumulate
VFMS	Floating-point Fused Multiply Subtract
VFNMS	Floating-point Fused Negate Multiply Subtract
VLDM	Load Multiple extension registers
VLDR	Loads an extension register from memory
VLMA	Floating-point Multiply Accumulate
VLMS	Floating-point Multiply Subtract
VMOV	Floating-point Move Immediate
VMOV	Floating-point Move Register
VMOV	Copy ARM core register to single precision
VMOV	Copy 2 ARM core registers to 2 single precision
VMOV	Copies between ARM core register to scalar
VMOV	Copies between Scalar to ARM core register
VMRS	Move to ARM core register from floating-point System Register
VMSR	Move to floating-point System Register from ARM Core register
VMUL	Multiply floating-point
VNEG	Floating-point negate
VNMLA	Floating-point multiply and add
VNMLS	Floating-point multiply and subtract
VNMUL	Floating-point multiply
VPOP	Pop extension registers

Table 11-27. Floating-point Instructions (Continued)

Mnemonic	Description
VPUSH	Push extension registers
VSQRT	Floating-point square root
VSTM	Store Multiple extension registers
VSTR	Stores an extension register to memory
VSUB	Floating-point Subtract

11.6.11.1 VABS

Floating-point Absolute.

Syntax

```
VABS{cond}.F32 Sd, Sm
```

where:

cond is an optional condition code, see “Conditional Execution” .

Sd, Sm are the destination floating-point value and the operand floating-point value.

Operation

This instruction:

1. Takes the absolute value of the operand floating-point register.
2. Places the results in the destination floating-point register.

Restrictions

There are no restrictions.

Condition Flags

The floating-point instruction clears the sign bit.

Examples

```
VABS.F32 S4, S6
```

11.6.11.2 VADD

Floating-point Add

Syntax

```
VADD{cond}.F32 {Sd,} Sn, Sm
```

where:

cond is an optional condition code, see “Conditional Execution” .

Sd, is the destination floating-point value.

Sn, Sm are the operand floating-point values.

Operation

This instruction:

1. Adds the values in the two floating-point operand registers.
2. Places the results in the destination floating-point register.

Restrictions

There are no restrictions.

Condition Flags

This instruction does not change the flags.

Examples

```
VADD.F32 S4, S6, S7
```

11.6.11.3 VCMP, VCMPE

Compares two floating-point registers, or one floating-point register and zero.

Syntax

```
VCMP{E}{cond}.F32 Sd, Sm
VCMP{E}{cond}.F32 Sd, #0.0
```

where:

cond — is an optional condition code, see "Conditional Execution" .

E — If present, any NaN operand causes an Invalid Operation exception. Otherwise, only a signaling NaN causes the exception.

Sd — is the floating-point operand to compare.

Sm — is the floating-point operand that is compared with.

Operation

This instruction:

1. Compares:
   - Two floating-point registers.
   - One floating-point register and zero.
2. Writes the result to the FPSCR flags.

Restrictions

This instruction can optionally raise an Invalid Operation exception if either operand is any type of NaN. It always raises an Invalid Operation exception if either operand is a signaling NaN.

Condition Flags

When this instruction writes the result to the FPSCR flags, the values are normally transferred to the ARM flags by a subsequent VMRS instruction, see "" .

Examples

```
VCMP.F32    S4, #0.0
VCMP.F32    S4, S2
```

11.6.11.4 VCVT, VCVTR between Floating-point and Integer

Converts a value in a register from floating-point to a 32-bit integer.

Syntax

```
VCVT{R}{cond}.Tm.F32 Sd, Sm
VCVT{cond}.F32.Tm Sd, Sm
```

where:

R — If *R* is specified, the operation uses the rounding mode specified by the FPSCR. If *R* is omitted. the operation uses the Round towards Zero rounding mode.

cond — is an optional condition code, see “Conditional Execution” .

Tm — is the data type for the operand. It must be one of:

S32 signed 32-bit value. — U32 unsigned 32-bit value.

Sd, Sm — are the destination register and the operand register.

Operation

These instructions:

1. Either
   - Converts a value in a register from floating-point value to a 32-bit integer.
   - Converts from a 32-bit integer to floating-point value.
2. Places the result in a second register.

The floating-point to integer operation normally uses the *Round towards Zero* rounding mode, but can optionally use the rounding mode specified by the FPSCR.

The integer to floating-point operation uses the rounding mode specified by the FPSCR.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

11.6.11.5 VCVT between Floating-point and Fixed-point

Converts a value in a register from floating-point to and from fixed-point.

Syntax

```
VCVT{cond}.Td.F32 Sd, Sd, #fbits
VCVT{cond}.F32.Td Sd, Sd, #fbits
```

where:

cond is an optional condition code, see "Conditional Execution" .

Td is the data type for the fixed-point number. It must be one of:

S16 signed 16-bit value.
U16 unsigned 16-bit value.

S32 signed 32-bit value.
U32 unsigned 32-bit value.

Sd is the destination register and the operand register.

fbits is the number of fraction bits in the fixed-point number:

If *Td* is S16 or U16, *fbits* must be in the range 0–16.
If *Td* is S32 or U32, *fbits* must be in the range 1–32.

Operation

These instructions:

1. Either
   - Converts a value in a register from floating-point to fixed-point.
   - Converts a value in a register from fixed-point to floating-point.
2. Places the result in a second register.

The floating-point values are single-precision.

The fixed-point value can be 16-bit or 32-bit. Conversions from fixed-point values take their operand from the low-order bits of the source register and ignore any remaining bits.

Signed conversions to fixed-point values sign-extend the result value to the destination register width.

Unsigned conversions to fixed-point values zero-extend the result value to the destination register width.

The floating-point to fixed-point operation uses the *Round towards Zero* rounding mode. The fixed-point to floating-point operation uses the *Round to Nearest* rounding mode.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

11.6.11.6 VCVTB, VCVTT

Converts between a half-precision value and a single-precision value.

Syntax

```
VCVT{y}{cond}.F32.F16 Sd, Sm
VCVT{y}{cond}.F16.F32 Sd, Sm
```

where:

y — Specifies which half of the operand register *Sm* or destination register *Sd* is used for the operand or destination:

- If y is B, then the bottom half, bits [15:0], of *Sm* or *Sd* is used.
- If *y* is T, then the top half, bits [31:16], of *Sm* or *Sd* is used.

cond — is an optional condition code, see “Conditional Execution” .

Sd — is the destination register.

Sm — is the operand register.

Operation

This instruction with the.F16.32 suffix:

1. Converts the half-precision value in the top or bottom half of a single-precision. register to single-precision.
2. Writes the result to a single-precision register.

This instruction with the.F32.F16 suffix:

1. Converts the value in a single-precision register to half-precision.
2. Writes the result into the top or bottom half of a single-precision register, preserving the other half of the target register.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

11.6.11.7 VDIV

Divides floating-point values.

Syntax

```
VDIV{cond}.F32 {Sd,} Sn, Sm
```

where:

cond is an optional condition code, see “Conditional Execution” .

Sd is the destination register.

Sn, Sm are the operand registers.

Operation

This instruction:

1. Divides one floating-point value by another floating-point value.
2. Writes the result to the floating-point destination register.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

11.6.11.8 VFMA, VFMS

Floating-point Fused Multiply Accumulate and Subtract.

Syntax

```
VFMA{cond}.F32 {Sd,} Sn, Sm
VFMS{cond}.F32 {Sd,} Sn, Sm
```

where:

cond is an optional condition code, see “Conditional Execution” .

Sd is the destination register.

Sn, Sm are the operand registers.

Operation

The VFMA instruction:

1. Multiplies the floating-point values in the operand registers.
2. Accumulates the results into the destination register.

The result of the multiply is not rounded before the accumulation.

The VFMS instruction:

1. Negates the first operand register.
2. Multiplies the floating-point values of the first and second operand registers.
3. Adds the products to the destination register.
4. Places the results in the destination register.

The result of the multiply is not rounded before the addition.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

11.6.11.9 VFNMA, VFNMS

Floating-point Fused Negate Multiply Accumulate and Subtract.

Syntax

```
VFNMA{cond}.F32 {Sd,} Sn, Sm
VFNMS{cond}.F32 {Sd,} Sn, Sm
```

where:

cond is an optional condition code, see “Conditional Execution” .

Sd is the destination register.

Sn, Sm are the operand registers.

Operation

The VFNMA instruction:

1. Negates the first floating-point operand register.
2. Multiplies the first floating-point operand with second floating-point operand.
3. Adds the negation of the floating -point destination register to the product
4. Places the result into the destination register.

The result of the multiply is not rounded before the addition.

The VFNMS instruction:

1. Multiplies the first floating-point operand with second floating-point operand.
2. Adds the negation of the floating-point value in the destination register to the product.
3. Places the result in the destination register.

The result of the multiply is not rounded before the addition.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

11.6.11.10 VLDM

Floating-point Load Multiple

Syntax

```
VLDM{mode}{cond}{.size} Rn{!}, list
```

where:

mode — is the addressing mode:
- *IA* Increment After. The consecutive addresses start at the address specified in *Rn*.
- *DB* Decrement Before. The consecutive addresses end just before the address specified in *Rn*.

cond — is an optional condition code, see “Conditional Execution” .

size — is an optional data size specifier.

Rn — is the base register. The SP can be used

! — is the command to the instruction to write a modified value back to *Rn*. This is required if mode == DB, and is optional if mode == IA.

list — is the list of extension registers to be loaded, as a list of consecutively numbered doubleword or singleword registers, separated by commas and surrounded by brackets.

Operation

This instruction loads:

- Multiple extension registers from consecutive memory locations using an address from an ARM core register as the base address.

Restrictions

The restrictions are:

- If *size* is present, it must be equal to the size in bits, 32 or 64, of the registers in *list*.
- For the base address, the SP can be used.
  In the ARM instruction set, if *!* is not specified the PC can be used.
- list must contain at least one register. If it contains doubleword registers, it must not contain more than 16 registers.
- If using the *Decrement Before addressing* mode, the write back flag, *!*, must be appended to the base register specification.

Condition Flags

These instructions do not change the flags.

11.6.11.11 VLDR

Loads a single extension register from memory

Syntax

```
VLDR{cond}{.64} Dd, [Rn{#imm}]
VLDR{cond}{.64} Dd, label
VLDR{cond}{.64} Dd, [PC, #imm}]
VLDR{cond}{.32} Sd, [Rn {, #imm}]
VLDR{cond}{.32} Sd, label
VLDR{cond}{.32} Sd, [PC, #imm]
```

where:

cond	is an optional condition code, see “Conditional Execution” .
64, 32	are the optional data size specifiers.
Dd	is the destination register for a doubleword load.
Sd	is the destination register for a singleword load.
Rn	is the base register. The SP can be used.
imm	is the + or - immediate offset used to form the address. Permitted address values are multiples of 4 in the range 0 to 1020.
label	is the label of the literal data item to be loaded.

Operation

This instruction:

- Loads a single extension register from memory, using a base address from an ARM core register, with an optional offset.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

11.6.11.12 VLMA, VLMS

Multiplies two floating-point values, and accumulates or subtracts the results.

Syntax

```
VLMA{cond}.F32 Sd, Sn, Sm
VLMS{cond}.F32 Sd, Sn, Sm
```

where:

cond is an optional condition code, see “Conditional Execution” .

Sd is the destination floating-point value.

Sn, Sm are the operand floating-point values.

Operation

The floating-point Multiply Accumulate instruction:

1. Multiplies two floating-point values.
2. Adds the results to the destination floating-point value.

The floating-point Multiply Subtract instruction:

1. Multiplies two floating-point values.
2. Subtracts the products from the destination floating-point value.
3. Places the results in the destination register.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

11.6.11.13 VMOV Immediate

Move floating-point Immediate

Syntax

```
VMOV{cond}.F32 Sd, #imm
```

where:

cond is an optional condition code, see “Conditional Execution” .

Sd is the branch destination.

imm is a floating-point constant.

Operation

This instruction copies a constant value to a floating-point register.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

11.6.11.14 VMOV Register

Copies the contents of one register to another.

Syntax

```
VMOV{cond}.F64 Dd, Dm
VMOV{cond}.F32 Sd, Sm
```

where:

cond is an optional condition code, see “Conditional Execution” .

Dd is the destination register, for a doubleword operation.

Dm is the source register, for a doubleword operation.

Sd is the destination register, for a singleword operation.

Sm is the source register, for a singleword operation.

Operation

This instruction copies the contents of one floating-point register to another.

Restrictions

There are no restrictions

Condition Flags

These instructions do not change the flags.

11.6.11.15 VMOV Scalar to ARM Core Register

Transfers one word of a doubleword floating-point register to an ARM core register.

Syntax

```
VMOV{cond} Rt, Dn[x]
```

where:

cond — is an optional condition code, see “Conditional Execution” .

Rt — is the destination ARM core register.

Dn — is the 64-bit doubleword register.

x — Specifies which half of the doubleword register to use:
- If *x* is 0, use lower half of doubleword register
- If *x* is 1, use upper half of doubleword register.

Operation

This instruction transfers:

- One word from the upper or lower half of a doubleword floating-point register to an ARM core register.

Restrictions

Rt cannot be PC or SP.

Condition Flags

These instructions do not change the flags.

11.6.11.16 VMOV ARM Core Register to Single Precision

Transfers a single-precision register to and from an ARM core register.

Syntax

```
VMOV{cond} Sn, Rt
VMOV{cond} Rt, Sn
```

where:

cond is an optional condition code, see “Conditional Execution” .

Sn is the single-precision floating-point register.

Rt is the ARM core register.

Operation

This instruction transfers:

- The contents of a single-precision register to an ARM core register.
- The contents of an ARM core register to a single-precision register.

Restrictions

Rt cannot be PC or SP.

Condition Flags

These instructions do not change the flags.

11.6.11.17 VMOV Two ARM Core Registers to Two Single Precision

Transfers two consecutively numbered single-precision registers to and from two ARM core registers.

Syntax

```
VMOV{cond} Sm, Sm1, Rt, Rt2
VMOV{cond} Rt, Rt2, Sm, Sm
```

where:

cond	is an optional condition code, see “Conditional Execution” .
Sm	is the first single-precision register.
Sm1	is the second single-precision register. This is the next single-precision register after *Sm*.
Rt	is the ARM core register that *Sm* is transferred to or from.
Rt2	is the The ARM core register that *Sm1* is transferred to or from.

Operation

This instruction transfers:

- The contents of two consecutively numbered single-precision registers to two ARM core registers.
- The contents of two ARM core registers to a pair of single-precision registers.

Restrictions

- The restrictions are:
- The floating-point registers must be contiguous, one after the other.
- The ARM core registers do not have to be contiguous.
- *Rt* cannot be PC or SP.

Condition Flags

These instructions do not change the flags.

11.6.11.18 VMOV ARM Core Register to Scalar

Transfers one word to a floating-point register from an ARM core register.

Syntax

```
VMOV{cond}{.32} Dd[x], Rt
```

where:

cond — is an optional condition code, see "Conditional Execution" .

32 — is an optional data size specifier.

Dd[x] — is the destination, where [x] defines which half of the doubleword is transferred, as follows:
If *x* is 0, the lower half is extracted
If *x* is 1, the upper half is extracted.

Rt — is the source ARM core register.

Operation

This instruction transfers one word to the upper or lower half of a doubleword floating-point register from an ARM core register.

Restrictions

Rt cannot be PC or SP.

Condition Flags

These instructions do not change the flags.

11.6.11.19 VMRS

Move to ARM Core register from floating-point System Register.

Syntax

```
VMRS{cond} Rt, FPSCR
VMRS{cond} APSR_nzcv, FPSCR
```

where:

cond is an optional condition code, see “Conditional Execution” .

Rt is the destination ARM core register. This register can be R0–R14.

APSR_nzcv Transfer floating-point flags to the APSR flags.

Operation

This instruction performs one of the following actions:

- Copies the value of the FPSCR to a general-purpose register.
- Copies the value of the FPSCR flag bits to the APSR N, Z, C, and V flags.

Restrictions

Rt cannot be PC or SP.

Condition Flags

These instructions optionally change the flags: N, Z, C, V

11.6.11.20 VMSR

Move to floating-point System Register from ARM Core register.

Syntax

```
VMSR{cond} FPSCR, Rt
```

where:

cond is an optional condition code, see “Conditional Execution” .

Rt is the general-purpose register to be transferred to the FPSCR.

Operation

This instruction moves the value of a general-purpose register to the FPSCR. See “Floating-point Status Control Register” for more information.

Restrictions

The restrictions are:

- *Rt* cannot be PC or SP.

Condition Flags

This instruction updates the FPSCR.

11.6.11.21 VMUL

Floating-point Multiply.

Syntax

```
VMUL{cond}.F32 {Sd,} Sn, Sm
```

where:

cond is an optional condition code, see “Conditional Execution” .

Sd is the destination floating-point value.

Sn, Sm are the operand floating-point values.

Operation

This instruction:

1. Multiplies two floating-point values.
2. Places the results in the destination register.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

11.6.11.22 VNEG

Floating-point Negate.

Syntax

```
VNEG{cond}.F32 Sd, Sm
```

where:

cond is an optional condition code, see “Conditional Execution” .

Sd is the destination floating-point value.

Sm is the operand floating-point value.

Operation

This instruction:

1. Negates a floating-point value.
2. Places the results in a second floating-point register.

The floating-point instruction inverts the sign bit.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

11.6.11.23 VNMLA, VNMLS, VNMUL

Floating-point multiply with negation followed by add or subtract.

Syntax

```
VNMLA{cond}.F32 Sd, Sn, Sm
VNMLS{cond}.F32 Sd, Sn, Sm
VNMUL{cond}.F32 {Sd,} Sn, Sm
```

where:

cond is an optional condition code, see “Conditional Execution” .

Sd is the destination floating-point register.

Sn, Sm are the operand floating-point registers.

Operation

The VNMLA instruction:

1. Multiplies two floating-point register values.
2. Adds the negation of the floating-point value in the destination register to the negation of the product.
3. Writes the result back to the destination register.

The VNMLS instruction:

1. Multiplies two floating-point register values.
2. Adds the negation of the floating-point value in the destination register to the product.
3. Writes the result back to the destination register.

The VNMUL instruction:

1. Multiplies together two floating-point register values.
2. Writes the negation of the result to the destination register.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

11.6.11.24 VPOP

Floating-point extension register Pop.

Syntax

```
VPOP{cond}{.size} list
```

where:

cond — is an optional condition code, see “Conditional Execution” .

size — is an optional data size specifier.
If present, it must be equal to the size in bits, 32 or 64, of the registers in *list*.

list — is the list of extension registers to be loaded, as a list of consecutively numbered doubleword or singleword registers, separated by commas and surrounded by brackets.

Operation

This instruction loads multiple consecutive extension registers from the stack.

Restrictions

The list must contain at least one register, and not more than sixteen registers.

Condition Flags

These instructions do not change the flags.

11.6.11.25 VPUSH

Floating-point extension register Push.

Syntax

```
VPUSH{cond}{.size} list
```

where:

cond — is an optional condition code, see “Conditional Execution” .

size — is an optional data size specifier.
If present, it must be equal to the size in bits, 32 or 64, of the registers in *list*.

list — is a list of the extension registers to be stored, as a list of consecutively numbered doubleword or singleword registers, separated by commas and surrounded by brackets.

Operation

This instruction:

- Stores multiple consecutive extension registers to the stack.

Restrictions

The restrictions are:

- list must contain at least one register, and not more than sixteen.

Condition Flags

These instructions do not change the flags.

11.6.11.26 VSQRT

Floating-point Square Root.

Syntax

```
VSQRT{cond}.F32 Sd, Sm
```

where:

cond is an optional condition code, see “Conditional Execution” .

Sd is the destination floating-point value.

Sm is the operand floating-point value.

Operation

This instruction:

- Calculates the square root of the value in a floating-point register.
- Writes the result to another floating-point register.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

11.6.11.27 VSTM

Floating-point Store Multiple.

Syntax

```
VSTM{mode}{cond}{.size} Rn{!}, list
```

where:

mode — is the addressing mode:
- *IA* Increment After. The consecutive addresses start at the address specified in *Rn*.
This is the default and can be omitted.
- *DB* Decrement Before. The consecutive addresses end just before the address specified in *Rn*.

cond — is an optional condition code, see “Conditional Execution” .

size — is an optional data size specifier.
If present, it must be equal to the size in bits, 32 or 64, of the registers in *list*.

Rn — is the base register. The SP can be used

! — is the function that causes the instruction to write a modified value back to *Rn*.
Required if mode == DB.

list — is a list of the extension registers to be stored, as a list of consecutively numbered doubleword or singleword registers, separated by commas and surrounded by brackets.

Operation

This instruction:

- Stores multiple extension registers to consecutive memory locations using a base address from an ARM core register.

Restrictions

The restrictions are:

- list must contain at least one register.
If it contains doubleword registers it must not contain more than 16 registers.
- Use of the PC as *Rn* is deprecated.

Condition Flags

These instructions do not change the flags.

11.6.11.28 VSTR

Floating-point Store.

Syntax

```
VSTR{cond}{.32} Sd, [Rn{, #imm}]
VSTR{cond}{.64} Dd, [Rn{, #imm}]
```

where

cond	is an optional condition code, see “Conditional Execution” .
32, 64	are the optional data size specifiers.
Sd	is the source register for a singleword store.
Dd	is the source register for a doubleword store.
Rn	is the base register. The SP can be used.
imm	is the + or - immediate offset used to form the address. Values are multiples of 4 in the range 0–1020. *imm* can be omitted, meaning an offset of +0.

Operation

This instruction:

- Stores a single extension register to memory, using an address from an ARM core register, with an optional offset, defined in *imm*.

Restrictions

The restrictions are:

- The use of PC for *Rn* is deprecated.

Condition Flags

These instructions do not change the flags.

11.6.11.29 VSUB

Floating-point Subtract.

Syntax

```
VSUB{cond}.F32 {Sd,} Sn, Sm
```

where:

cond is an optional condition code, see “Conditional Execution” .

Sd is the destination floating-point value.

Sn, Sm are the operand floating-point value.

Operation

This instruction:

1. Subtracts one floating-point value from another floating-point value.
2. Places the results in the destination floating-point register.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

11.6.12 Miscellaneous Instructions

The table below shows the remaining Cortex-M4 instructions.

Table 11-28. Miscellaneous Instructions

Mnemonic	Description
BKPT	Breakpoint
CPSID	Change Processor State, Disable Interrupts
CPSIE	Change Processor State, Enable Interrupts
DMB	Data Memory Barrier
DSB	Data Synchronization Barrier
ISB	Instruction Synchronization Barrier
MRS	Move from special register to register
MSR	Move from register to special register
NOP	No Operation
SEV	Send Event
SVC	Supervisor Call
WFE	Wait For Event
WFI	Wait For Interrupt

11.6.12.1 BKPT

Breakpoint.

Syntax

```
BKPT #imm
```

where:

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

Operation

The BKPT instruction causes the processor to enter Debug state. Debug tools can use this to investigate system state when the instruction at a particular address is reached.

imm is ignored by the processor. If required, a debugger can use it to store additional information about the breakpoint.

The BKPT instruction can be placed inside an IT block, but it executes unconditionally, unaffected by the condition specified by the IT instruction.

Condition Flags

This instruction does not change the flags.

Examples

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

Note: ARM does not recommend the use of the BKPT instruction with an immediate value set to 0xAB for any purpose other than Semi-hosting.

11.6.12.2 CPS

Change Processor State.

Syntax

```
CPSeffect iflags
```

where:

effect is one of:

- IE Clears the special purpose register.
- ID Sets the special purpose register.

iflags is a sequence of one or more flags:

- i Set or clear PRIMASK.
- f Set or clear FAULTMASK.

Operation

CPS changes the PRIMASK and FAULTMASK special register values. See "Exception Mask Registers" for more information about these registers.

Restrictions

The restrictions are:

- Use CPS only from privileged software, it has no effect if used in unprivileged software
- CPS cannot be conditional and so must not be used inside an IT block.

Condition Flags

This instruction does not change the condition flags.

Examples

```
CPSID i  ; Disable interrupts and configurable fault handlers (set PRIMASK)
CPSID f  ; Disable interrupts and all fault handlers (set FAULTMASK)
CPSIE i  ; Enable interrupts and configurable fault handlers (clear PRIMASK)
CPSIE f  ; Enable interrupts and fault handlers (clear FAULTMASK)
```

11.6.12.3 DMB

Data Memory Barrier.

Syntax

```
DMB{cond}
```

where:

cond is an optional condition code, see “Conditional Execution” .

Operation

DMB acts as a data memory barrier. It ensures that all explicit memory accesses that appear, in program order, before the DMB instruction are completed before any explicit memory accesses that appear, in program order, after the DMB instruction. DMB does not affect the ordering or execution of instructions that do not access memory.

Condition Flags

This instruction does not change the flags.

Examples

```
DMB  ; Data Memory Barrier
```

11.6.12.4 DSB

Data Synchronization Barrier.

Syntax

```
DSB{cond}
```

where:

cond is an optional condition code, see “Conditional Execution” .

Operation

DSB acts as a special data synchronization memory barrier. Instructions that come after the DSB, in program order, do not execute until the DSB instruction completes. The DSB instruction completes when all explicit memory accesses before it complete.

Condition Flags

This instruction does not change the flags.

Examples

```
DSB ; Data Synchronisation Barrier
```

11.6.12.5 ISB

Instruction Synchronization Barrier.

Syntax

```
ISB{cond}
```

where:

cond is an optional condition code, see “Conditional Execution” .

Operation

ISB acts as an instruction synchronization barrier. It flushes the pipeline of the processor, so that all instructions following the ISB are fetched from cache or memory again, after the ISB instruction has been completed.

Condition Flags

This instruction does not change the flags.

Examples

```
ISB  ; Instruction Synchronisation Barrier
```

11.6.12.6 MRS

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

Syntax

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

where:

cond — is an optional condition code, see "Conditional Execution" .

Rd — is the destination register.

spec_reg — can be any of: APSR, IPSR, EPSR, IEPSR, IAPSR, EAPSR, PSR, MSP, PSP, PRIMASK, BASEPRI, BASEPRI_MAX, FAULTMASK, or CONTROL.

Operation

Use MRS in combination with MSR as part of a read-modify-write sequence for updating a PSR, for example to clear the Q flag.

In process swap code, the programmers model state of the process being swapped out must be saved, including relevant PSR contents. Similarly, the state of the process being swapped in must also be restored. These operations use MRS in the state-saving instruction sequence and MSR in the state-restoring instruction sequence.

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

See "MSR" .

Restrictions

Rd must not be SP and must not be PC.

Condition Flags

This instruction does not change the flags.

Examples

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

11.6.12.7 MSR

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

Syntax

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

where:

cond — is an optional condition code, see “Conditional Execution” .

Rn — is the source register.

spec_reg — can be any of: APSR, IPSR, EPSR, IEPSR, IAPSR, EAPSR, PSR, MSP, PSP, PRIMASK, BASEPRI, BASEPRI_MAX, FAULTMASK, or CONTROL.

Operation

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

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

Note: When the user writes to BASEPRI_MAX, the instruction writes to BASEPRI only if either:
Rn is non-zero and the current BASEPRI value is 0
Rn is non-zero and less than the current BASEPRI value.

See “MRS” .

Restrictions

Rn must not be SP and must not be PC.

Condition Flags

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

Examples

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

11.6.12.8 NOP

No Operation.

Syntax

```
NOP{cond}
```

where:

cond is an optional condition code, see “Conditional Execution” .

Operation

NOP does nothing. NOP is not necessarily a time-consuming NOP. The processor might remove it from the pipeline before it reaches the execution stage.

Use NOP for padding, for example to place the following instruction on a 64-bit boundary.

Condition Flags

This instruction does not change the flags.

Examples

```
NOP  ; No operation
```

11.6.12.9 SEV

Send Event.

Syntax

```
SEV{cond}
```

where:

cond is an optional condition code, see “Conditional Execution” .

Operation

SEV is a hint instruction that causes an event to be signaled to all processors within a multiprocessor system. It also sets the local event register to 1, see “Power Management” .

Condition Flags

This instruction does not change the flags.

Examples

```
SEV ; Send Event
```

11.6.12.10 SVC

Supervisor Call.

Syntax

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

where:

cond is an optional condition code, see “Conditional Execution” .

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

Operation

The SVC instruction causes the SVC exception.

imm is ignored by the processor. If required, it can be retrieved by the exception handler to determine what service is being requested.

Condition Flags

This instruction does not change the flags.

Examples

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

11.6.12.11 WFE

Wait For Event.

Syntax

```
WFE{cond}
```

where:

cond is an optional condition code, see “Conditional Execution” .

Operation

WFE is a hint instruction.

If the event register is 0, WFE suspends execution until one of the following events occurs:

- An exception, unless masked by the exception mask registers or the current priority level
- An exception enters the Pending state, if SEVONPEND in the System Control Register is set
- A Debug Entry request, if Debug is enabled
- An event signaled by a peripheral or another processor in a multiprocessor system using the SEV instruction.

If the event register is 1, WFE clears it to 0 and returns immediately.

For more information, see “Power Management” .

Condition Flags

This instruction does not change the flags.

Examples

```
WFE  ; Wait for event
```

11.6.12.12 WFI

Wait for Interrupt.

Syntax

```
WFI{cond}
```

where:

cond is an optional condition code, see “Conditional Execution” .

Operation

WFI is a hint instruction that suspends execution until one of the following events occurs:

- An exception
- A Debug Entry request, regardless of whether Debug is enabled.

Condition Flags

This instruction does not change the flags.

Examples

```
WFI ; Wait for interrupt
```

11.7 Cortex-M4 Core Peripherals

11.7.1 Peripherals

- Nested Vectored Interrupt Controller (NVIC)
  The Nested Vectored Interrupt Controller (NVIC) is an embedded interrupt controller that supports low latency interrupt processing. See Section 11.8 ”Nested Vectored Interrupt Controller (NVIC)”.
- System Control Block (SCB)
  The System Control Block (SCB) is the programmers model interface to the processor. It provides system implementation information and system control, including configuration, control, and reporting of system exceptions. See Section 11.9 ”System Control Block (SCB)”.
- System Timer (SysTick)
  The System Timer, SysTick, is a 24-bit count-down timer. Use this as a Real Time Operating System (RTOS) tick timer or as a simple counter. See Section 11.10 ”System Timer (SysTick)”.
- Memory Protection Unit (MPU)
  The Memory Protection Unit (MPU) improves system reliability by defining the memory attributes for different memory regions. It provides up to eight different regions, and an optional predefined background region. See Section 11.11 ”Memory Protection Unit (MPU)”.
- Floating-point Unit (FPU)
  The Floating-point Unit (FPU) provides IEEE754-compliant operations on single-precision, 32-bit, floating-point values. See Section 11.12 ”Floating Point Unit (FPU)”.

11.7.2 Address Map

The address map of the *Private peripheral bus* (PPB) is given in the following table.

Table 11-29. Core Peripheral Register Regions

Address	Core Peripheral
0xE000E008–0xE000E00F	System Control Block
0xE000E010–0xE000E01F	System Timer
0xE000E100–0xE000E4EF	Nested Vectored Interrupt Controller
0xE000ED00–0xE000ED3F	System control block
0xE000ED90–0xE000EDB8	Memory Protection Unit
0xE000EF00–0xE000EF03	Nested Vectored Interrupt Controller
0xE000EF30–0xE000EF44	Floating-point Unit

In register descriptions:

- The *required privilege* gives the privilege level required to access the register, as follows:
  - Privileged: Only privileged software can access the register.
  - Unprivileged: Both unprivileged and privileged software can access the register.

11.8 Nested Vectored Interrupt Controller (NVIC)

This section describes the NVIC and the registers it uses. The NVIC supports:

- Up to 47 interrupts
- A programmable priority level of 0–15 for each interrupt. A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority.
- Level detection of interrupt signals
- Dynamic reprioritization of interrupts
- Grouping of priority values into group priority and subpriority fields
- Interrupt tail-chaining
- An external *Non-maskable interrupt* (NMI)

The processor automatically stacks its state on exception entry and unstacks this state on exception exit, with no instruction overhead. This provides low latency exception handling.

11.8.1 Level-sensitive Interrupts

The processor supports level-sensitive interrupts. A level-sensitive interrupt is held asserted until the peripheral deasserts the interrupt signal. Typically, this happens because the ISR accesses the peripheral, causing it to clear the interrupt request.

When the processor enters the ISR, it automatically removes the pending state from the interrupt (see “Hardware and Software Control of Interrupts”). For a level-sensitive interrupt, if the signal is not deasserted before the processor returns from the ISR, the interrupt becomes pending again, and the processor must execute its ISR again. This means that the peripheral can hold the interrupt signal asserted until it no longer requires servicing.

11.8.1.1 Hardware and Software Control of Interrupts

The Cortex-M4 latches all interrupts. A peripheral interrupt becomes pending for one of the following reasons:

- The NVIC detects that the interrupt signal is HIGH and the interrupt is not active
- The NVIC detects a rising edge on the interrupt signal
- A software writes to the corresponding interrupt set-pending register bit, see “Interrupt Set-pending Registers” , or to the NVIC_STIR to make an interrupt pending, see “Software Trigger Interrupt Register” .

A pending interrupt remains pending until one of the following:

- The processor enters the ISR for the interrupt. This changes the state of the interrupt from pending to active. Then:
  - For a level-sensitive interrupt, when the processor returns from the ISR, the NVIC samples the interrupt signal. If the signal is asserted, the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR. Otherwise, the state of the interrupt changes to inactive.
- Software writes to the corresponding interrupt clear-pending register bit.
  For a level-sensitive interrupt, if the interrupt signal is still asserted, the state of the interrupt does not change. Otherwise, the state of the interrupt changes to inactive.

11.8.2 NVIC Design Hints and Tips

Ensure that the software uses correctly aligned register accesses. The processor does not support unaligned accesses to NVIC registers. See the individual register descriptions for the supported access sizes.

A interrupt can enter a pending state even if it is disabled. Disabling an interrupt only prevents the processor from taking that interrupt. Before programming SCB_VTOR to relocate the vector table, ensure that the vector table entries of the new vector table are set up for fault handlers, NMI and all enabled exception like interrupts. For more information, see the “Vector Table Offset Register” .

11.8.2.1 NVIC Programming Hints

The software uses the CPSIE I and CPSID I instructions to enable and disable the interrupts. The CMSIS provides the following intrinsic functions for these instructions:

void __disable_irq(void) // Disable Interrupts

void __enable_irq(void) // Enable Interrupts

In addition, the CMSIS provides a number of functions for NVIC control, including:

Table 11-30. CMSIS Functions for NVIC Control

CMSIS Interrupt Control Function	Description
void NVIC_SetPriorityGrouping(uint32_t priority_grouping)	Set the priority grouping
void NVIC_EnableIRQ(IRQn_t IRQn)	Enable IRQn
void NVIC_DisableIRQ(IRQn_t IRQn)	Disable IRQn
uint32_t NVIC_GetPendingIRQ (IRQn_t IRQn)	Return true (IRQ-Number) if IRQn is pending
void NVIC_SetPendingIRQ (IRQn_t IRQn)	Set IRQn pending
void NVIC_ClearPendingIRQ (IRQn_t IRQn)	Clear IRQn pending status
uint32_t NVIC_GetActive (IRQn_t IRQn)	Return the IRQ number of the active interrupt
void NVIC_SetPriority (IRQn_t IRQn, uint32_t priority)	Set priority for IRQn
uint32_t NVIC_GetPriority (IRQn_t IRQn)	Read priority of IRQn
void NVIC_SystemReset (void)	Reset the system

The input parameter IRQn is the IRQ number. For more information about these functions, see the CMSIS documentation.

To improve software efficiency, the CMSIS simplifies the NVIC register presentation. In the CMSIS:

- The Set-enable, Clear-enable, Set-pending, Clear-pending and Active Bit registers map to arrays of 32-bit integers, so that:
  - The array ISER[0] 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 Interrupt Priority Registers (IPR0–IPR12) provide an 8-bit priority field for each interrupt and each register holds four priority fields.

The CMSIS provides thread-safe code that gives atomic access to the Interrupt Priority Registers. Table 11-31 shows how the interrupts, or IRQ numbers, map onto the interrupt registers and corresponding CMSIS variables that have one bit per interrupt.

Table 11-31. Mapping of Interrupts

	CMSIS Array Elements[1]				
Interrupts	**Set-enable**	**Clear-enable**	**Set-pending**	**Clear-pending**	**Active Bit**
0–31	ISER[0]	ICER[0]	ISPR[0]	ICPR[0]	IABR[0]
32–47	ISER[1]	ICER[1]	ISPR[1]	ICPR[1]	IABR[1]

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

11.8.3 Nested Vectored Interrupt Controller (NVIC) User Interface

Table 11-32. Nested Vectored Interrupt Controller (NVIC) Register Mapping

Offset	Register	Name	Access	Reset
0xE000E100	Interrupt Set-enable Register 0	NVIC_ISER0	Read/Write	0x00000000
...	...	...	...	...
0xE000E11C	Interrupt Set-enable Register 7	NVIC_ISER7	Read/Write	0x00000000
0XE000E180	Interrupt Clear-enable Register 0	NVIC_ICER0	Read/Write	0x00000000
...	...	...	...	...
0xE000E19C	Interrupt Clear-enable Register 7	NVIC_ICER7	Read/Write	0x00000000
0XE000E200	Interrupt Set-pending Register 0	NVIC_ISPR0	Read/Write	0x00000000
...	...	...	...	...
0xE000E21C	Interrupt Set-pending Register 7	NVIC_ISPR7	Read/Write	0x00000000
0XE000E280	Interrupt Clear-pending Register 0	NVIC_ICPR0	Read/Write	0x00000000
...	...	...	...	...
0xE000E29C	Interrupt Clear-pending Register 7	NVIC_ICPR7	Read/Write	0x00000000
0xE000E300	Interrupt Active Bit Register 0	NVIC_IABR0	Read/Write	0x00000000
...	...	...	...	...
0xE000E31C	Interrupt Active Bit Register 7	NVIC_IABR7	Read/Write	0x00000000
0xE000E400	Interrupt Priority Register 0	NVIC_IPR0	Read/Write	0x00000000
...	...	...	...	...
0XE000E42C	Interrupt Priority Register 12	NVIC_IPR12	Read/Write	0x00000000
0xE000EF00	Software Trigger Interrupt Register	NVIC_STIR	Write-only	0x00000000

11.8.3.1 Interrupt Set-enable Registers

Name: NVIC_ISERx [x=0..7]

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
SETENA							

23	22	21	20	19	18	17	16
SETENA							

15	14	13	12	11	10	9	8
SETENA							

7	6	5	4	3	2	1	0
SETENA							

These registers enable interrupts and show which interrupts are enabled.

- **SETENA: Interrupt Set-enable**

Write:

0: No effect.

1: Enables the interrupt.

Read:

0: Interrupt disabled.

1: Interrupt enabled.

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

11.8.3.2 Interrupt Clear-enable Registers

Name: NVIC_ICERx [x=0..7]

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
CLRENA							

23	22	21	20	19	18	17	16
CLRENA							

15	14	13	12	11	10	9	8
CLRENA							

7	6	5	4	3	2	1	0
CLRENA							

These registers disable interrupts, and show which interrupts are enabled.

- **CLRENA: Interrupt Clear-enable**

Write:

0: No effect.

1: Disables the interrupt.

Read:

0: Interrupt disabled.

1: Interrupt enabled.

11.8.3.3 Interrupt Set-pending Registers

Name: NVIC_ISPRx [x=0..7]

Access: Read/Write

Reset: 0x000000000

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							

These registers force interrupts into the pending state, and show which interrupts are pending.

- **SETPEND: Interrupt Set-pending**

Write:

0: No effect.

1: Changes the interrupt state to pending.

Read:

0: Interrupt is not pending.

1: Interrupt is pending.

Notes:
1. Writing a 1 to an ISPR bit corresponding to an interrupt that is pending has no effect.
2. Writing a 1 to an ISPR bit corresponding to a disabled interrupt sets the state of that interrupt to pending.

11.8.3.4 Interrupt Clear-pending Registers

Name: NVIC_ICPRx [x=0..7]

Access: Read/Write

Reset: 0x000000000

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							

These registers remove the pending state from interrupts, and show which interrupts are pending.

- **CLRPEND: Interrupt Clear-pending**

Write:

0: No effect.

1: Removes the pending state from an interrupt.

Read:

0: Interrupt is not pending.

1: Interrupt is pending.

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

11.8.3.5 Interrupt Active Bit Registers

Name: NVIC_IABRx [x=0..7]

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
ACTIVE							

23	22	21	20	19	18	17	16
ACTIVE							

15	14	13	12	11	10	9	8
ACTIVE							

7	6	5	4	3	2	1	0
ACTIVE							

These registers indicate which interrupts are active.

- **ACTIVE: Interrupt Active Flags**

0: Interrupt is not active.

1: Interrupt is active.

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

11.8.3.6 Interrupt Priority Registers

Name: NVIC_IPRx [x=0..12]

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
PRI3							

23	22	21	20	19	18	17	16
PRI2							

15	14	13	12	11	10	9	8
PRI1							

7	6	5	4	3	2	1	0
PRI0							

The NVIC_IPR0–NVIC_IPR12 registers provide a 8-bit priority field for each interrupt. These registers are byte-accessible. Each register holds four priority fields that map up to four elements in the CMSIS interrupt priority array IP[0] to IP[46].

- **PRI3: Priority (4m+3)**

Priority, Byte Offset 3, refers to register bits [31:24].

- **PRI2: Priority (4m+2)**

Priority, Byte Offset 2, refers to register bits [23:16].

- **PRI1: Priority (4m+1)**

Priority, Byte Offset 1, refers to register bits [15:8].

- **PRI0: Priority (4m)**

Priority, Byte Offset 0, refers to register bits [7:0].

Notes:
1. Each priority field holds a priority value, 0–15. The lower the value, the greater the priority of the corresponding interrupt. The processor implements only bits[7:4] of each field; bits[3:0] read as zero and ignore writes.
2. For more information about the IP[0] to IP[46] interrupt priority array, that provides the software view of the interrupt priorities, see Table 11-30, “CMSIS Functions for NVIC Control” .
3. The corresponding IPR number *n* is given by *n* = *m* DIV 4.
4. The byte offset of the required Priority field in this register is *m* MOD 4.

11.8.3.7 Software Trigger Interrupt Register

Name: NVIC_STIR

Access: Write-only

Reset: 0x000000000

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

Write to this register to generate an interrupt from the software.

- **INTID: Interrupt ID**

Interrupt ID of the interrupt to trigger, in the range 0–239. For example, a value of 0x03 specifies interrupt IRQ3.

11.9 System Control Block (SCB)

The System Control Block (SCB) provides system implementation information, and system control. This includes configuration, control, and reporting of the system exceptions.

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

- Except for the SCB_CFSR and SCB_SHPR1–SCB_SHPR3 registers, it must use aligned word accesses
- For the SCB_CFSR and SCB_SHPR1–SCB_SHPR3 registers, it can use byte or aligned halfword or word accesses.

The processor does not support unaligned accesses to system control block registers.

In a fault handler, to determine the true faulting address:

1. Read and save the MMFAR or SCB_BFAR value.
2. Read the MMARVALID bit in the MMFSR subregister, or the BFARVALID bit in the BFSR subregister. The SCB_MMFAR or SCB_BFAR address is valid only if this bit is 1.

The software must follow this sequence because another higher priority exception might change the SCB_MMFAR or SCB_BFAR value. For example, if a higher priority handler preempts the current fault handler, the other fault might change the SCB_MMFAR or SCB_BFAR value.

11.9.1 System Control Block (SCB) User Interface

Table 11-33. System Control Block (SCB) Register Mapping

Offset	Register	Name	Access	Reset
0xE000E008	Auxiliary Control Register	SCB_ACTLR	Read/Write	0x00000000
0xE000ED00	CPUID Base Register	SCB_CPUID	Read-only	0x410FC240
0xE000ED04	Interrupt Control and State Register	SCB_ICSR	Read/Write(1)	0x00000000
0xE000ED08	Vector Table Offset Register	SCB_VTOR	Read/Write	0x00000000
0xE000ED0C	Application Interrupt and Reset Control Register	SCB_AIRCR	Read/Write	0xFA050000
0xE000ED10	System Control Register	SCB_SCR	Read/Write	0x00000000
0xE000ED14	Configuration and Control Register	SCB_CCR	Read/Write	0x00000200
0xE000ED18	System Handler Priority Register 1	SCB_SHPR1	Read/Write	0x00000000
0xE000ED1C	System Handler Priority Register 2	SCB_SHPR2	Read/Write	0x00000000
0xE000ED20	System Handler Priority Register 3	SCB_SHPR3	Read/Write	0x00000000
0xE000ED24	System Handler Control and State Register	SCB_SHCSR	Read/Write	0x00000000
0xE000ED28	Configurable Fault Status Register	SCB_CFSR(2)	Read/Write	0x00000000
0xE000ED2C	HardFault Status Register	SCB_HFSR	Read/Write	0x00000000
0xE000ED34	MemManage Fault Address Register	SCB_MMFAR	Read/Write	Unknown
0xE000ED38	BusFault Address Register	SCB_BFAR	Read/Write	Unknown

Notes:
1. See the register description for more information.
2. This register contains the subregisters: "MMFSR: Memory Management Fault Status Subregister" (0xE000ED28 - 8 bits), "BFSR: Bus Fault Status Subregister" (0xE000ED29 - 8 bits), "UFSR: Usage Fault Status Subregister" (0xE000ED2A - 16 bits).

11.9.1.1 Auxiliary Control Register

Name: SCB_ACTLR

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
–	–	–	–	–	–	DISOOFP	DISFPCA
7	6	5	4	3	2	1	0
–	–	–	–	–	DISFOLD	DISDEFWBUF	DISMCYCINT

The SCB_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.

By default, this register is set to provide optimum performance from the Cortex-M4 processor, and does not normally require modification.

- **DISOOFP: Disable Out Of Order Floating Point**

Disables floating point instructions that complete out of order with respect to integer instructions.

- **DISFPCA: Disable FPCA**

Disables an automatic update of CONTROL.FPCA.

- **DISFOLD: Disable Folding**

When set to 1, disables the IT folding.

Note: In some situations, the processor can start executing the first instruction in an IT block while it is still executing the IT instruction. This behavior is called IT folding, and it improves the performance. However, IT folding can cause jitter in looping. If a task must avoid jitter, set the DISFOLD bit to 1 before executing the task, to disable the IT folding.

- **DISDEFWBUF: Disable Default Write Buffer**

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

This bit only affects write buffers implemented in the Cortex-M4 processor.

- **DISMCYCINT: Disable Multiple Cycle Interruption**

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

11.9.1.2 CPUID Base Register

Name: SCB_CPUID

Access: Read/Write

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			

The SCB_CPUID register contains the processor part number, version, and implementation information.

- **Implementer: Implementer Code**

0x41: ARM.

- **Variant: Variant Number**

It is the r value in the rnpn product revision identifier:

0x0: Revision 0.

- **Constant: Reads as 0xF**

Reads as 0xF.

- **PartNo: Part Number of the Processor**

0xC24 = Cortex-M4.

- **Revision: Revision Number**

It is the p value in the rnpn product revision identifier:

0x0: Patch 0.

11.9.1.3 Interrupt Control and State Register

Name: SCB_ICSR

Access: Read/Write

31	30	29	28	27	26	25	24
NMIPENDSET	–		PENDSVSET	PENDSVCLR	PENDSTSET	PENDSTCLR	–
23	22	21	20	19	18	17	16
–	ISRPENDING	VECTPENDING					
15	14	13	12	11	10	9	8
VECTPENDING				RETTOBASE	–	–	VECTACTIVE
7	6	5	4	3	2	1	0
VECTACTIVE							

The SCB_ICSR provides a set-pending bit for the Non-Maskable Interrupt (NMI) exception, and set-pending and clear-pending bits for the PendSV and SysTick exceptions.

It indicates:

- The exception number of the exception being processed, and whether there are preempted active exceptions,
- The exception number of the highest priority pending exception, and whether any interrupts are pending.

- **NMIPENDSET: NMI Set-pending**

Write:

PendSV set-pending bit.

Write:

0: No effect.

1: Changes NMI exception state to pending.

Read:

0: NMI exception is not pending.

1: NMI exception is pending.

As NMI is the highest-priority exception, the processor normally enters the NMI exception handler as soon as it registers a write of 1 to this bit. Entering the handler clears this bit to 0. A read of this bit by the NMI exception handler returns 1 only if the NMI signal is reasserted while the processor is executing that handler.

- **PENDSVSET: PendSV Set-pending**

Write:

0: No effect.

1: Changes PendSV exception state to pending.

Read:

0: PendSV exception is not pending.

1: PendSV exception is pending.

Writing a 1 to this bit is the only way to set the PendSV exception state to pending.

- **PENDSVCLR: PendSV Clear-pending**

Write:

0: No effect.

1: Removes the pending state from the PendSV exception.

- **PENDSTSET: SysTick Exception Set-pending**

Write:

0: No effect.

1: Changes SysTick exception state to pending.

Read:

0: SysTick exception is not pending.

1: SysTick exception is pending.

- **PENDSTCLR: SysTick Exception Clear-pending**

Write:

0: No effect.

1: Removes the pending state from the SysTick exception.

This bit is Write-only. On a register read, its value is Unknown.

- **ISRPENDING: Interrupt Pending Flag (Excluding NMI and Faults)**

0: Interrupt not pending.

1: Interrupt pending.

- **VECTPENDING: Exception Number of the Highest Priority Pending Enabled Exception**

0: No pending exceptions.

Nonzero: The exception number of the highest priority pending enabled exception.

The value indicated by this field includes the effect of the BASEPRI and FAULTMASK registers, but not any effect of the PRIMASK register.

- **RETTOBASE: Preempted Active Exceptions Present or Not**

0: There are preempted active exceptions to execute.

1: There are no active exceptions, or the currently-executing exception is the only active exception.

- **VECTACTIVE: Active Exception Number Contained**

0: Thread mode.

Nonzero: The exception number of the currently active exception. The value is the same as IPSR bits [8:0]. See “Interrupt Program Status Register” .

Subtract 16 from this value to obtain the IRQ number required to index into the Interrupt Clear-Enable, Set-Enable, Clear-Pending, Set-Pending, or Priority Registers, see “Interrupt Program Status Register” .

Note: When the user writes to the SCB_ICSR, the effect is unpredictable if:
- Writing a 1 to the PENDSVSET bit and writing a 1 to the PENDSVCLR bit
- Writing a 1 to the PENDSTSET bit and writing a 1 to the PENDSTCLR bit.

11.9.1.4 Vector Table Offset Register

Name: SCB_VTOR

Access: Read/Write

31	30	29	28	27	26	25	24
TBLOFF							

23	22	21	20	19	18	17	16
TBLOFF							

15	14	13	12	11	10	9	8
TBLOFF							

7	6	5	4	3	2	1	0
TBLOFF	–	–	–	–	–	–	–

The SCB_VTOR indicates the offset of the vector table base address from memory address 0x00000000.

- **TBLOFF: Vector Table Base Offset**

It contains bits [29:7] of the offset of the table base from the bottom of the memory map.

Bit [29] determines whether the vector table is in the code or SRAM memory region:

0: Code.

1: SRAM.

It is sometimes called the TBLBASE bit.

Note: When setting TBLOFF, the offset must be aligned to the number of exception entries in the vector table. Configure the next statement to give the information required for your implementation; the statement reminds the user of how to determine the alignment requirement. The minimum alignment is 32 words, enough for up to 16 interrupts. For more interrupts, adjust the alignment by rounding up to the next power of two. For example, if 21 interrupts are required, the alignment must be on a 64-word boundary because the required table size is 37 words, and the next power of two is 64.

Table alignment requirements mean that bits[6:0] of the table offset are always zero.

11.9.1.5 Application Interrupt and Reset Control Register

Name: SCB_AIRCR

Access: Read/Write

31	30	29	28	27	26	25	24
VECTKEYSTAT/VECTKEY							
23	22	21	20	19	18	17	16
VECTKEYSTAT/VECTKEY							
15	14	13	12	11	10	9	8
ENDIANNESS	–	–	–	–	PRIGROUP		
7	6	5	4	3	2	1	0
–	–	–	–	–	SYSRESETREQ	VECTCLRACTIVE	VECTRESET

The SCB_AIRCR provides priority grouping control for the exception model, endian status for data accesses, and reset control of the system. To write to this register, write 0x5FA to the VECTKEY field, otherwise the processor ignores the write.

- **VECTKEYSTAT: Register Key (Read)**

Reads as 0xFA05.

- **VECTKEY: Register Key (Write)**

Writes 0x5FA to VECTKEY, otherwise the write is ignored.

- **ENDIANNESS: Data Endianness**

0: Little-endian.

1: Big-endian.

- **PRIGROUP: Interrupt Priority Grouping**

This field determines the split of group priority from subpriority. It shows the position of the binary point that splits the PRI_*n* fields in the Interrupt Priority Registers into separate *group priority* and *subpriority* fields. The table below shows how the PRIGROUP value controls this split.

	Interrupt Priority Level Value, PRI_*N*[7:0]			Number of	
PRIGROUP	**Binary Point**[1]	**Group Priority Bits**	**Subpriority Bits**	**Group Priorities**	**Subpriorities**
0b000	bxxxxxxx.y	[7:1]	None	128	2
0b001	bxxxxxx.yy	[7:2]	[4:0]	64	4
0b010	bxxxxx.yyy	[7:3]	[4:0]	32	8
0b011	bxxxx.yyyy	[7:4]	[4:0]	16	16
0b100	bxxx.yyyyy	[7:5]	[4:0]	8	32
0b101	bxx.yyyyyy	[7:6]	[5:0]	4	64
0b110	bx.yyyyyyy	[7]	[6:0]	2	128
0b111	b.yyyyyyy	None	[7:0]	1	256

Note: 1. PRI_n[7:0] field showing the binary point. x denotes a group priority field bit, and y denotes a subpriority field bit.

Determining preemption of an exception uses only the group priority field.

- **SYSRESETREQ: System Reset Request**

0: No system reset request.

1: Asserts a signal to the outer system that requests a reset.

This is intended to force a large system reset of all major components except for debug. This bit reads as 0.

- **VECTCLRACTIVE: Reserved for Debug use**

This bit reads as 0. When writing to the register, write a 0 to this bit, otherwise the behavior is unpredictable.

- **VECTRESET: Reserved for Debug use**

This bit reads as 0. When writing to the register, write a 0 to this bit, otherwise the behavior is unpredictable.

11.9.1.6 System Control Register

Name: SCB_SCR

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
–	–	–	SEVONPEND	–	SLEEPDEEP	SLEEPONEXIT	–

- **SEVONPEND: Send Event on Pending Bit**

0: Only enabled interrupts or events can wake up the processor; disabled interrupts are excluded.

1: Enabled events and all interrupts, including disabled interrupts, can wake up the processor.

When an event or an interrupt enters the pending state, the event signal wakes up the processor from WFE. If the processor is not waiting for an event, the event is registered and affects the next WFE.

The processor also wakes up on execution of an SEV instruction or an external event.

- **SLEEPDEEP: Sleep or Deep Sleep**

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

0: Sleep.

1: Deep sleep.

- **SLEEPONEXIT: Sleep-on-exit**

Indicates sleep-on-exit when returning from the Handler mode to the Thread mode:

0: Do not sleep when returning to Thread mode.

1: Enter sleep, or deep sleep, on return from an ISR.

Setting this bit to 1 enables an interrupt-driven application to avoid returning to an empty main application.

11.9.1.7 Configuration and Control Register

Name: SCB_CCR

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
–	–	–	–	–	–	STKALIGN	BFHFNMIGN

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

The SCB_CCR controls the entry to the Thread mode and enables the handlers for NMI, hard fault and faults escalated by FAULTMASK to ignore BusFaults. It also enables the division by zero and unaligned access trapping, and the access to the NVIC_STIR by unprivileged software (see “Software Trigger Interrupt Register”).

- **STKALIGN: Stack Alignment**

Indicates the stack alignment on exception entry:

0: 4-byte aligned.

1: 8-byte aligned.

On exception entry, the processor uses bit [9] of the stacked PSR to indicate the stack alignment. On return from the exception, it uses this stacked bit to restore the correct stack alignment.

- **BFHFNMIGN: Bus Faults Ignored**

Enables handlers with priority -1 or -2 to ignore data bus faults caused by load and store instructions. This applies to the hard fault and FAULTMASK escalated handlers:

0: Data bus faults caused by load and store instructions cause a lock-up.

1: Handlers running at priority -1 and -2 ignore data bus faults caused by load and store instructions.

Set this bit to 1 only when the handler and its data are in absolutely safe memory. The normal use of this bit is to probe system devices and bridges to detect control path problems and fix them.

- **DIV_0_TRP: Division by Zero Trap**

Enables faulting or halting when the processor executes an SDIV or UDIV instruction with a divisor of 0:

0: Do not trap divide by 0.

1: Trap divide by 0.

When this bit is set to 0, a divide by zero returns a quotient of 0.

- **UNALIGN_TRP: Unaligned Access Trap**

Enables unaligned access traps:

0: Do not trap unaligned halfword and word accesses.

1: Trap unaligned halfword and word accesses.

If this bit is set to 1, an unaligned access generates a usage fault.

Unaligned LDM, STM, LDRD, and STRD instructions always fault irrespective of whether UNALIGN_TRP is set to 1.

- **USERSETMPEND: Unprivileged Software Access**

Enables unprivileged software access to the NVIC_STIR, see “Software Trigger Interrupt Register” :

0: Disable.

1: Enable.

- **NONBASETHRDENA: Thread Mode Enable**

Indicates how the processor enters Thread mode:

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

1: The processor can enter the Thread mode from any level under the control of an EXC_RETURN value, see “Exception Return” .

11.9.1.8 System Handler Priority Registers

The SCB_SHPR1–SCB_SHPR3 registers set the priority level, 0 to 15 of the exception handlers that have configurable priority. They are byte-accessible.

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

Table 11-34. System Fault Handler Priority Fields

Handler	Field	Register Description
Memory management fault (MemManage)	PRI_4	System Handler Priority Register 1
Bus fault (BusFault)	PRI_5	
Usage fault (UsageFault)	PRI_6	
SVCall	PRI_11	System Handler Priority Register 2
PendSV	PRI_14	System Handler Priority Register 3
SysTick	PRI_15	

Each PRI_N field is 8 bits wide, but the processor implements only bits [7:4] of each field, and bits [3:0] read as zero and ignore writes.

11.9.1.9 System Handler Priority Register 1

Name: SCB_SHPR1

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

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_6: Priority**

Priority of system handler 6, UsageFault.

- **PRI_5: Priority**

Priority of system handler 5, BusFault.

- **PRI_4: Priority**

Priority of system handler 4, MemManage.

11.9.1.10 System Handler Priority Register 2

Name: SCB_SHPR2

Access: Read/Write

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

- **PRI_11: Priority**

Priority of system handler 11, SVCall.

11.9.1.11 System Handler Priority Register 3

Name: SCB_SHPR3

Access: Read/Write

31	30	29	28	27	26	25	24
PRI_15							
23	22	21	20	19	18	17	16
PRI_14							
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

- **PRI_15: Priority**

Priority of system handler 15, SysTick exception.

- **PRI_14: Priority**

Priority of system handler 14, PendSV.

11.9.1.12 System Handler Control and State Register

Name: SCB_SHCSR

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

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

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

7	6	5	4	3	2	1	0
SVCALLACT	–	–	–	USGFAULTACT	–	BUSFAULTACT	MEMFAULTACT

The SHCSR enables the system handlers, and indicates the pending status of the bus fault, memory management fault, and SVC exceptions; it also indicates the active status of the system handlers.

- **USGFAULTENA: Usage Fault Enable**

0: Disables the exception.

1: Enables the exception.

- **BUSFAULTENA: Bus Fault Enable**

0: Disables the exception.

1: Enables the exception.

- **MEMFAULTENA: Memory Management Fault Enable**

0: Disables the exception.

1: Enables the exception.

- **SVCALLPENDED: SVC Call Pending**

Read:

0: The exception is not pending.

1: The exception is pending.

Note: The user can write to these bits to change the pending status of the exceptions.

- **BUSFAULTPENDED: Bus Fault Exception Pending**

Read:

0: The exception is not pending.

1: The exception is pending.

Note: The user can write to these bits to change the pending status of the exceptions.

- **MEMFAULTPENDED: Memory Management Fault Exception Pending**

Read:

0: The exception is not pending.

1: The exception is pending.

Note: The user can write to these bits to change the pending status of the exceptions.

- **USGFAULTPENDED: Usage Fault Exception Pending**

Read:

0: The exception is not pending.

1: The exception is pending.

Note: The user can write to these bits to change the pending status of the exceptions.

- **SYSTICKACT: SysTick Exception Active**

Read:

0: The exception is not active.

1: The exception is active.

Note: The user can write to these bits to change the active status of the exceptions.
- Caution: A software that changes the value of an active bit in this register without a correct adjustment to the stacked content can cause the processor to generate a fault exception. Ensure that the software writing to this register retains and subsequently restores the current active status.
- Caution: After enabling the system handlers, to change the value of a bit in this register, the user must use a read-modify-write procedure to ensure that only the required bit is changed.

- **PENDSVACT: PendSV Exception Active**

0: The exception is not active.

1: The exception is active.

- **MONITORACT: Debug Monitor Active**

0: Debug monitor is not active.

1: Debug monitor is active.

- **SVCALLACT: SVC Call Active**

0: SVC call is not active.

1: SVC call is active.

- **USGFAULTACT: Usage Fault Exception Active**

0: Usage fault exception is not active.

1: Usage fault exception is active.

- **BUSFAULTACT: Bus Fault Exception Active**

0: Bus fault exception is not active.

1: Bus fault exception is active.

- **MEMFAULTACT: Memory Management Fault Exception Active**

0: Memory management fault exception is not active.

1: Memory management fault exception is active.

If the user disables a system handler and the corresponding fault occurs, the processor treats the fault as a hard fault.

The user can write to this register to change the pending or active status of system exceptions. An OS kernel can write to the active bits to perform a context switch that changes the current exception type.

11.9.1.13 Configurable Fault Status Register

Name: SCB_CFSR

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	DIVBYZERO	UNALIGNED
23	22	21	20	19	18	17	16
–	–	–	–	NOCP	INVPC	INVSTATE	UNDEFINSTR
15	14	13	12	11	10	9	8
BFARVALID	–	LSPERR	STKERR	UNSTKERR	IMPRECISERR	PRECISERR	IBUSERR
7	6	5	4	3	2	1	0
MMARVALID	–	MLSPERR	MSTKERR	MUNSTKERR	–	DACCVIOL	IACCVIOL

- **IACCVIOL: Instruction Access Violation Flag**

This is part of "MMFSR: Memory Management Fault Status Subregister" .

0: No instruction access violation fault.

1: The processor attempted an instruction fetch from a location that does not permit execution.

This fault occurs on any access to an XN region, even when the MPU is disabled or not present.

When this bit is 1, the PC value stacked for the exception return points to the faulting instruction. The processor has not written a fault address to the SCB_MMFAR.

- **DACCVIOL: Data Access Violation Flag**

This is part of "MMFSR: Memory Management Fault Status Subregister" .

0: No data access violation fault.

1: The processor attempted a load or store at a location that does not permit the operation.

When this bit is 1, the PC value stacked for the exception return points to the faulting instruction. The processor has loaded the SCB_MMFAR with the address of the attempted access.

- **MUNSTKERR: Memory Manager Fault on Unstacking for a Return From Exception**

This is part of "MMFSR: Memory Management Fault Status Subregister" .

0: No unstacking fault.

1: Unstack for an exception return has caused one or more access violations.

This fault is chained to the handler. This means that when this bit is 1, the original return stack is still present. The processor has not adjusted the SP from the failing return, and has not performed a new save. The processor has not written a fault address to the SCB_MMFAR.

- **MSTKERR: Memory Manager Fault on Stacking for Exception Entry**

This is part of "MMFSR: Memory Management Fault Status Subregister" .

0: No stacking fault.

1: Stacking for an exception entry has caused one or more access violations.

When this bit is 1, the SP is still adjusted but the values in the context area on the stack might be incorrect. The processor has not written a fault address to SCB_MMFAR.

- **MLSPERR: MemManage During Lazy State Preservation**

This is part of “MMFSR: Memory Management Fault Status Subregister” .

0: No MemManage fault occurred during the floating-point lazy state preservation.

1: A MemManage fault occurred during the floating-point lazy state preservation.

- **MMARVALID: Memory Management Fault Address Register (SCB_MMFAR) Valid Flag**

This is part of “MMFSR: Memory Management Fault Status Subregister” .

0: The value in SCB_MMFAR is not a valid fault address.

1: SCB_MMFAR holds a valid fault address.

If a memory management fault occurs and is escalated to a hard fault because of priority, the hard fault handler must set this bit to 0. This prevents problems on return to a stacked active memory management fault handler whose SCB_MMFAR value has been overwritten.

- **IBUSERR: Instruction Bus Error**

This is part of “BFSR: Bus Fault Status Subregister” .

0: No instruction bus error.

1: Instruction bus error.

The processor detects the instruction bus error on prefetching an instruction, but it sets the IBUSERR flag to 1 only if it attempts to issue the faulting instruction.

When the processor sets this bit to 1, it does not write a fault address to the BFAR.

- **PRECISERR: Precise Data Bus Error**

This is part of “BFSR: Bus Fault Status Subregister” .

0: No precise data bus error.

1: A data bus error has occurred, and the PC value stacked for the exception return points to the instruction that caused the fault.

When the processor sets this bit to 1, it writes the faulting address to the SCB_BFAR.

- **IMPRECISERR: Imprecise Data Bus Error**

This is part of “BFSR: Bus Fault Status Subregister” .

0: No imprecise data bus error.

1: A data bus error has occurred, but the return address in the stack frame is not related to the instruction that caused the error.

When the processor sets this bit to 1, it does not write a fault address to the SCB_BFAR.

This is an asynchronous fault. Therefore, if it is detected when the priority of the current process is higher than the bus fault priority, the bus fault becomes pending and becomes active only when the processor returns from all higher priority processes. If a precise fault occurs before the processor enters the handler for the imprecise bus fault, the handler detects that both this bit and one of the precise fault status bits are set to 1.

- **UNSTKERR: Bus Fault on Unstacking for a Return From Exception**

This is part of “BFSR: Bus Fault Status Subregister” .

0: No unstacking fault.

1: Unstack for an exception return has caused one or more bus faults.

This fault is chained to the handler. This means that when the processor sets this bit to 1, the original return stack is still present. The processor does not adjust the SP from the failing return, does not performed a new save, and does not write a fault address to the BFAR.

- **STKERR: Bus Fault on Stacking for Exception Entry**

This is part of “BFSR: Bus Fault Status Subregister” .

0: No stacking fault.

1: Stacking for an exception entry has caused one or more bus faults.

When the processor sets this bit to 1, the SP is still adjusted but the values in the context area on the stack might be incorrect. The processor does not write a fault address to the SCB_BFAR.

- **LSPERR: Bus Error During Lazy Floating-point State Preservation**

This is part of “BFSR: Bus Fault Status Subregister” .

0: No bus fault occurred during floating-point lazy state preservation

1: A bus fault occurred during floating-point lazy state preservation.

- **BFARVALID: Bus Fault Address Register (BFAR) Valid flag**

This is part of “BFSR: Bus Fault Status Subregister” .

0: The value in SCB_BFAR is not a valid fault address.

1: SCB_BFAR holds a valid fault address.

The processor sets this bit to 1 after a bus fault where the address is known. Other faults can set this bit to 0, such as a memory management fault occurring later.

If a bus fault occurs and is escalated to a hard fault because of priority, the hard fault handler must set this bit to 0. This prevents problems if returning to a stacked active bus fault handler whose SCB_BFAR value has been overwritten.

- **UNDEFINSTR: Undefined Instruction Usage Fault**

This is part of “UFSR: Usage Fault Status Subregister” .

0: No undefined instruction usage fault.

1: The processor has attempted to execute an undefined instruction.

When this bit is set to 1, the PC value stacked for the exception return points to the undefined instruction.

An undefined instruction is an instruction that the processor cannot decode.

- **INVSTATE: Invalid State Usage Fault**

This is part of “UFSR: Usage Fault Status Subregister” .

0: No invalid state usage fault.

1: The processor has attempted to execute an instruction that makes illegal use of the EPSR.

When this bit is set to 1, the PC value stacked for the exception return points to the instruction that attempted the illegal use of the EPSR.

This bit is not set to 1 if an undefined instruction uses the EPSR.

- **INVPC: Invalid PC Load Usage Fault**

This is part of “UFSR: Usage Fault Status Subregister” . It is caused by an invalid PC load by EXC_RETURN:

0: No invalid PC load usage fault.

1: The processor has attempted an illegal load of EXC_RETURN to the PC, as a result of an invalid context, or an invalid EXC_RETURN value.

When this bit is set to 1, the PC value stacked for the exception return points to the instruction that tried to perform the illegal load of the PC.

- **NOCP: No Coprocessor Usage Fault**

This is part of “UFSR: Usage Fault Status Subregister” . The processor does not support coprocessor instructions:

0: No usage fault caused by attempting to access a coprocessor.

1: The processor has attempted to access a coprocessor.

- **UNALIGNED: Unaligned Access Usage Fault**

This is part of “UFSR: Usage Fault Status Subregister” .

0: No unaligned access fault, or unaligned access trapping not enabled.

1: The processor has made an unaligned memory access.

Enable trapping of unaligned accesses by setting the UNALIGN_TRP bit in the SCB_CCR to 1. See “Configuration and Control Register” . Unaligned LDM, STM, LDRD, and STRD instructions always fault irrespective of the setting of UNALIGN_TRP.

- **DIVBYZERO: Divide by Zero Usage Fault**

This is part of “UFSR: Usage Fault Status Subregister” .

0: No divide by zero fault, or divide by zero trapping not enabled.

1: The processor has executed an SDIV or UDIV instruction with a divisor of 0.

When the processor sets this bit to 1, the PC value stacked for the exception return points to the instruction that performed the divide by zero. Enable trapping of divide by zero by setting the DIV_0_TRP bit in the SCB_CCR to 1. See “Configuration and Control Register” .

11.9.1.14 Configurable Fault Status Register (Byte Access)

Name: SCB_CFSR (BYTE)

Access: Read/Write

31	30	29	28	27	26	25	24
UFSR							

23	22	21	20	19	18	17	16
UFSR							

15	14	13	12	11	10	9	8
BFSR							

7	6	5	4	3	2	1	0
MMFSR							

- **MMFSR: Memory Management Fault Status Subregister**

The flags in the MMFSR subregister indicate the cause of memory access faults. See bitfield [7..0] description in Section 11.9.1.13.

- **BFSR: Bus Fault Status Subregister**

The flags in the BFSR subregister indicate the cause of a bus access fault. See bitfield [14..8] description in Section 11.9.1.13.

- **UFSR: Usage Fault Status Subregister**

The flags in the UFSR subregister indicate the cause of a usage fault. See bitfield [31..15] description in Section 11.9.1.13.

Note: The UFSR bits are sticky. This means that as one or more fault occurs, the associated bits are set to 1. A bit that is set to 1 is cleared to 0 only by wrting a 1 to that bit, or by a reset.

The SCB_CFSR indicates the cause of a memory management fault, bus fault, or usage fault. It is byte accessible. The user can access the SCB_CFSR or its subregisters as follows:

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

11.9.1.15 Hard Fault Status Register

Name: SCB_HFSR

Access: Read/Write

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

The SCB_HFSR gives information about events that activate the hard fault handler. This register is read, write to clear. This means that bits in the register read normally, but wrting a 1 to any bit clears that bit to 0.

- **DEBUGEVT: Reserved for Debug Use**

When writing to the register, write a 0 to this bit, otherwise the behavior is unpredictable.

- **FORCED: Forced Hard Fault**

It indicates a forced hard fault, generated by escalation of a fault with configurable priority that cannot be handles, either because of priority or because it is disabled:

0: No forced hard fault.

1: Forced hard fault.

When this bit is set to 1, the hard fault handler must read the other fault status registers to find the cause of the fault.

- **VECTTBL: Bus Fault on a Vector Table**

It indicates a bus fault on a vector table read during an exception processing:

0: No bus fault on vector table read.

1: Bus fault on vector table read.

This error is always handled by the hard fault handler.

When this bit is set to 1, the PC value stacked for the exception return points to the instruction that was preempted by the exception.

Note: The HFSR bits are sticky. This means that, as one or more fault occurs, the associated bits are set to 1. A bit that is set to 1 is cleared to 0 only by wrting a 1 to that bit, or by a reset.

11.9.1.16 MemManage Fault Address Register

Name: SCB_MMFAR

Access: Read/Write

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							

The SCB_MMFAR contains the address of the location that generated a memory management fault.

- **ADDRESS: Memory Management Fault Generation Location Address**

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

Notes:
1. When an unaligned access faults, the address is the actual address that faulted. Because a single read or write instruction can be split into multiple aligned accesses, the fault address can be any address in the range of the requested access size.
2. Flags in the MMFSR subregister indicate the cause of the fault, and whether the value in the SCB_MMFAR is valid. See “MMFSR: Memory Management Fault Status Subregister” .

11.9.1.17 Bus Fault Address Register

Name: SCB_BFAR

Access: Read/Write

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							

The SCB_BFAR contains the address of the location that generated a bus fault.

- **ADDRESS: Bus Fault Generation Location Address**

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

Notes:
1. When an unaligned access faults, the address in the SCB_BFAR is the one requested by the instruction, even if it is not the address of the fault.
2. Flags in the BFSR indicate the cause of the fault, and whether the value in the SCB_BFAR is valid. See “BFSR: Bus Fault Status Subregister” .

11.10 System Timer (SysTick)

The processor has a 24-bit system timer, SysTick, that counts down from the reload value to zero, reloads (wraps to) the value in the SYST_RVR on the next clock edge, then counts down on subsequent clocks.

When the processor is halted for debugging, the counter does not decrement.

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

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

The SysTick counter reload and current value are undefined at reset; the correct initialization sequence for the SysTick counter is:

1. Program the reload value.
2. Clear the current value.
3. Program the Control and Status register.

11.10.1 System Timer (SysTick) User Interface

Table 11-35. System Timer (SYST) Register Mapping

Offset	Register	Name	Access	Reset
0xE000E010	SysTick Control and Status Register	SYST_CSR	Read/Write	0x00000000
0xE000E014	SysTick Reload Value Register	SYST_RVR	Read/Write	Unknown
0xE000E018	SysTick Current Value Register	SYST_CVR	Read/Write	Unknown
0xE000E01C	SysTick Calibration Value Register	SYST_CALIB	Read-only	0x00003A98

11.10.1.1 SysTick Control and Status Register

Name: SYST_CSR

Access: Read/Write

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

The SysTick SYST_CSR enables the SysTick features.

- **COUNTFLAG: Count Flag**

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

- **CLKSOURCE: Clock Source**

Indicates the clock source:

0: External Clock.

1: Processor Clock.

- **TICKINT: SysTick Exception Request Enable**

Enables a SysTick exception request:

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

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

The software can use COUNTFLAG to determine if SysTick has ever counted to zero.

- **ENABLE: Counter Enable**

Enables the counter:

0: Counter disabled.

1: Counter enabled.

When ENABLE is set to 1, the counter loads the RELOAD value from the SYST_RVR 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.

11.10.1.2 SysTick Reload Value Registers

Name: SYST_RVR

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
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							

The SYST_RVR specifies the start value to load into the SYST_CVR.

- **RELOAD: SYST_CVR Load Value**

Value to load into the SYST_CVR when the counter is enabled and when it reaches 0.

The RELOAD value can be any value in the range 0x00000001–0x00FFFFFF. A start value of 0 is possible, but has no effect because the SysTick exception request and COUNTFLAG are activated when counting from 1 to 0.

The RELOAD value is calculated according to its use: For example, to generate a multi-shot timer with a period of N processor clock cycles, use a RELOAD value of N-1. If the SysTick interrupt is required every 100 clock pulses, set RELOAD to 99.

11.10.1.3 SysTick Current Value Register

Name: SYST_CVR

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

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							

The SysTick SYST_CVR contains the current value of the SysTick counter.

- **CURRENT: SysTick Counter Current Value**

Reads return the current value of the SysTick counter.

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

11.10.1.4 SysTick Calibration Value Register

Name: SYST_CALIB

Access: Read/Write

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

The SysTick SYST_CSR indicates the SysTick calibration properties.

- **NOREF: No Reference Clock**

It indicates whether the device provides a reference clock to the processor:

0: Reference clock provided.

1: No reference clock provided.

If your device does not provide a reference clock, the SYST_CSR.CLKSOURCE bit reads-as-one and ignores writes.

- **SKEW: TENMS Value Verification**

It indicates whether the TENMS value is exact:

0: TENMS value is exact.

1: TENMS value is inexact, or not given.

An inexact TENMS value can affect the suitability of SysTick as a software real time clock.

- **TENMS: Ten Milliseconds**

The reload value for 10 ms (100 Hz) timing is subject to system clock skew errors. If the value reads as zero, the calibration value is not known.

The TENMS field default value is 0x00003A98 (15000 decimal).

In order to achieve a 1 ms timebase on SystTick, the TENMS field must be programmed to a value corresponding to the processor clock frequency (in kHz) divided by 8.

For example, for devices running the processor clock at 48 MHz, the TENMS field value must be 0x0001770 (48000 kHz/8).

11.11 Memory Protection Unit (MPU)

The MPU divides the memory map into a number of regions, and defines the location, size, access permissions, and memory attributes of each region. It supports:

- Independent attribute settings for each region
- Overlapping regions
- Export of memory attributes to the system.

The memory attributes affect the behavior of memory accesses to the region. The Cortex-M4 MPU defines:

- Eight separate memory regions, 0–7
- A background region.

When memory regions overlap, a memory access is affected by the attributes of the region with the highest number. For example, the attributes for region 7 take precedence over the attributes of any region that overlaps region 7.

The background region has the same memory access attributes as the default memory map, but is accessible from privileged software only.

The Cortex-M4 MPU memory map is unified. This means that instruction accesses and data accesses have the same region settings.

If a program accesses a memory location that is prohibited by the MPU, the processor generates a memory management fault. This causes a fault exception, and might cause the termination of the process in an OS environment.

In an OS environment, the kernel can update the MPU region setting dynamically based on the process to be executed. Typically, an embedded OS uses the MPU for memory protection.

The configuration of MPU regions is based on memory types (see "Memory Regions, Types and Attributes").

Table 11-36 shows the possible MPU region attributes. These include Share ability and cache behavior attributes that are not relevant to most microcontroller implementations. See "MPU Configuration for a Microcontroller" for guidelines for programming such an implementation.

Table 11-36. Memory Attributes Summary

Memory Type	Shareability	Other Attributes	Description
Strongly-ordered	–	–	All accesses to Strongly-ordered memory occur in program order. All Strongly-ordered regions are assumed to be shared.
Device	Shared	–	Memory-mapped peripherals that several processors share.
	Non-shared	–	Memory-mapped peripherals that only a single processor uses.
Normal	Shared	Non-cacheable Write-through Cacheable Write-back Cacheable	Normal memory that is shared between several processors.
	Non-shared	Non-cacheable Write-through Cacheable Write-back Cacheable	Normal memory that only a single processor uses.

11.11.1 MPU Access Permission Attributes

This section describes the MPU access permission attributes. The access permission bits (TEX, C, B, S, AP, and XN) of the MPU_RASR control the access to the corresponding memory region. If an access is made to an area of memory without the required permissions, then the MPU generates a permission fault.

The table below shows the encodings for the TEX, C, B, and S access permission bits.

Table 11-37. 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	Outer and inner noncacheable.
			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	Cached memory BB = outer policy, AA = inner policy.
			1		Shareable	

Note: 1. The MPU ignores the value of this bit.

Table 11-38 shows the cache policy for memory attribute encodings with a TEX value is in the range 4–7.

Table 11-38. 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 11-39 shows the AP encodings that define the access permissions for privileged and unprivileged software.

Table 11-39. 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

11.11.1.1 MPU Mismatch

When an access violates the MPU permissions, the processor generates a memory management fault, see "Exceptions and Interrupts" . The MMFSR indicates the cause of the fault. See "MMFSR: Memory Management Fault Status Subregister" for more information.

11.11.1.2 Updating an MPU Region

To update the attributes for an MPU region, update the MPU_RNR, MPU_RBAR and MPU_RASRs. Each register can be programed separately, or a multiple-word write can be used to program all of these registers. MPU_RBAR and MPU_RASR aliases can be used to program up to four regions simultaneously using an STM instruction.

11.11.1.3 Updating an MPU Region Using Separate Words

Simple code to configure one region:

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
LDR R0,=MPU_RNR         ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]      ; Region Number
STR R4, [R0, #0x4]      ; Region Base Address
STRH R2, [R0, #0x8]     ; Region Size and Enable
STRH R3, [R0, #0xA]     ; Region Attribute
```

Disable a region before writing new region settings to the MPU, if the region being changed was previously enabled. For example:

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
LDR R0,=MPU_RNR          ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]       ; Region Number
BIC R2, R2, #1           ; Disable
STRH R2, [R0, #0x8]      ; Region Size and Enable
STR R4, [R0, #0x4]       ; Region Base Address
STRH R3, [R0, #0xA]      ; Region Attribute
ORR R2, #1               ; Enable
```

```
STRH R2, [R0, #0x8]       ; Region Size and Enable
```

The software must use memory barrier instructions:

- Before the MPU setup, if there might be outstanding memory transfers, such as buffered writes, that might be affected by the change in MPU settings
- After the MPU setup, if it includes memory transfers that must use the new MPU settings.

However, memory barrier instructions are not required if the MPU setup process starts by entering an exception handler, or is followed by an exception return, because the exception entry and exception return mechanisms cause memory barrier behavior.

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

For example, if the user wants all of the memory access behavior to take effect immediately after the programming sequence, a DSB instruction and an ISB instruction must be used. A DSB is required after changing MPU settings, such as at the end of a context switch. An ISB is required if the code that programs the MPU region or regions is entered using a branch or call. If the programming sequence is entered using a return from exception, or by taking an exception, then an ISB is not required.

11.11.1.4 Updating an MPU Region Using Multi-word Writes

The user can program directly using multi-word writes, depending on how the information is divided. Consider the following reprogramming:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
LDR R0, =MPU_RNR    ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]  ; Region Number
STR R2, [R0, #0x4]  ; Region Base Address
STR R3, [R0, #0x8]  ; Region Attribute, Size and Enable
```

Use an STM instruction to optimize this:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
LDR R0, =MPU_RNR    ; 0xE000ED98, MPU region number register
STM R0, {R1-R3}     ; Region Number, address, attribute, size and enable
```

This can be done in two words for pre-packed information. This means that the MPU_RBAR contains the required region number and had the VALID bit set to 1. See “MPU Region Base Address Register” . Use this when the data is statically packed, for example in a boot loader:

```
; R1 = address and region number in one
; R2 = size and attributes in one
LDR R0, =MPU_RBAR   ; 0xE000ED9C, MPU Region Base register
STR R1, [R0, #0x0]  ; Region base address and
                    ; region number combined with VALID (bit 4) set to 1
STR R2, [R0, #0x4]  ; Region Attribute, Size and Enable
```

Use an STM instruction to optimize this:

```
; R1 = address and region number in one
; R2 = size and attributes in one
LDR R0,=MPU_RBAR    ; 0xE000ED9C, MPU Region Base register
```

```
STM R0, {R1-R2}     ; Region base address, region number and VALID bit,
                    ; and Region Attribute, Size and Enable
```

11.11.1.5 Subregions

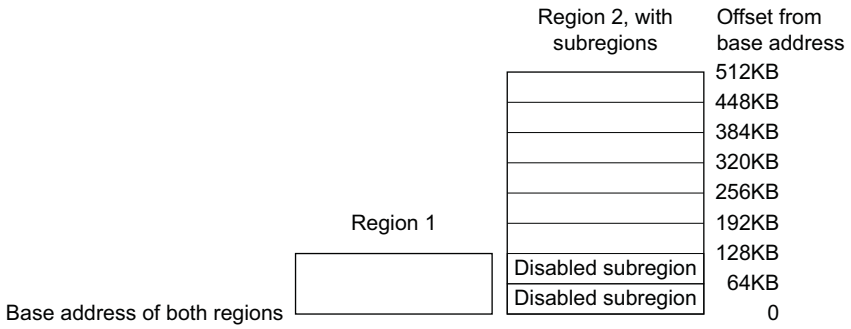
Regions of 256 bytes or more are divided into eight equal-sized subregions. Set the corresponding bit in the SRD field of the MPU_RASR field to disable a subregion. See "MPU Region Attribute and Size Register" . The least significant bit of SRD controls the first subregion, and the most significant bit controls the last subregion. Disabling a subregion means another region overlapping the disabled range matches instead. If no other enabled region overlaps the disabled subregion, the MPU issues a fault.

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

11.11.1.6 Example of SRD Use

Two regions with the same base address overlap. Region 1 is 128 KB, and region 2 is 512 KB. To ensure the attributes from region 1 apply to the first 128 KB region, set the SRD field for region 2 to b00000011 to disable the first two subregions, as in Figure 11-13 below:

Figure 11-13. SRD Use



11.11.1.7 MPU Design Hints And Tips

To avoid unexpected behavior, disable the interrupts before updating the attributes of a region that the interrupt handlers might access.

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

- Except for the MPU_RASR, it must use aligned word accesses
- For the MPU_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.

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 11-40. 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 shareability and cache policy attributes do not affect the system behavior. However, using these settings for the MPU regions can make the application code more portable. The values given are for typical situations. In special systems, such as multiprocessor designs or designs with a separate DMA engine, the shareability attribute might be important. In these cases, refer to the recommendations of the memory device manufacturer.

11.11.2 Memory Protection Unit (MPU) User Interface

Table 11-41. Memory Protection Unit (MPU) Register Mapping

Offset	Register	Name	Access	Reset
0xE000ED90	MPU Type Register	MPU_TYPE	Read-only	0x00000800
0xE000ED94	MPU Control Register	MPU_CTRL	Read/Write	0x00000000
0xE000ED98	MPU Region Number Register	MPU_RNR	Read/Write	0x00000000
0xE000ED9C	MPU Region Base Address Register	MPU_RBAR	Read/Write	0x00000000
0xE000EDA0	MPU Region Attribute and Size Register	MPU_RASR	Read/Write	0x00000000
0xE000EDA4	MPU Region Base Address Register Alias 1	MPU_RBAR_A1	Read/Write	0x00000000
0xE000EDA8	MPU Region Attribute and Size Register Alias 1	MPU_RASR_A1	Read/Write	0x00000000
0xE000EDAC	MPU Region Base Address Register Alias 2	MPU_RBAR_A2	Read/Write	0x00000000
0xE000EDB0	MPU Region Attribute and Size Register Alias 2	MPU_RASR_A2	Read/Write	0x00000000
0xE000EDB4	MPU Region Base Address Register Alias 3	MPU_RBAR_A3	Read/Write	0x00000000
0xE000EDB8	MPU Region Attribute and Size Register Alias 3	MPU_RASR_A3	Read/Write	0x00000000

11.11.2.1 MPU Type Register

Name: MPU_TYPE

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
IREGION							

15	14	13	12	11	10	9	8
DREGION							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	SEPARATE

The MPU_TYPE register indicates whether the MPU is present, and if so, how many regions it supports.

- **IREGION: Instruction Region**

Indicates the number of supported MPU instruction regions.

Always contains 0x00. The MPU memory map is unified and is described by the DREGION field.

- **DREGION: Data Region**

Indicates the number of supported MPU data regions:

0x08 = Eight MPU regions.

- **SEPARATE: Separate Instruction**

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

0: Unified.

11.11.2.2 MPU Control Register

Name: MPU_CTRL

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
–	–	–	–	–	PRIVDEFENA	HFNMIENA	ENABLE

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

- **PRIVDEFENA: Privileged Default Memory Map Enable**

Enables privileged software access to the default memory map:

0: If the MPU is enabled, disables the use of the default memory map. Any memory access to a location not covered by any enabled region causes a fault.

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

When enabled, the background region acts as a region number -1. Any region that is defined and enabled has priority over this default map.

If the MPU is disabled, the processor ignores this bit.

- **HFNMIENA: Hard Fault and NMI Enable**

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: MPU Enable**

Enables the MPU:

0: MPU disabled.

1: MPU enabled.

When ENABLE and PRIVDEFENA are both set to 1:

- For privileged accesses, the *default memory map* is as described in “Memory Model” . Any access by privileged software that does not address an enabled memory region behaves as defined by the default memory map.
- Any access by unprivileged software that does not address an enabled memory region causes a memory management fault.

XN and Strongly-ordered rules always apply to the System Control Space regardless of the value of the ENABLE bit.

When the ENABLE bit is set to 1, at least one region of the memory map must be enabled for the system to function unless the PRIVDEFENA bit is set to 1. If the PRIVDEFENA bit is set to 1 and no regions are enabled, then only privileged software can operate.

When the ENABLE bit is set to 0, the system uses the default memory map. This has the same memory attributes as if the MPU is not implemented. The default memory map applies to accesses from both privileged and unprivileged software.

When the MPU is enabled, accesses to the System Control Space and vector table are always permitted. Other areas are accessible based on regions and whether PRIVDEFENA is set to 1.

Unless 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.

11.11.2.3 MPU Region Number Register

Name: MPU_RNR

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
REGION							

The MPU_RNR selects which memory region is referenced by the MPU_RBAR and MPU_RASRs.

- **REGION: MPU Region Referenced by the MPU_RBAR and MPU_RASRs**

Indicates the MPU region referenced by the MPU_RBAR and MPU_RASRs.

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

Normally, the required region number is written to this register before accessing the MPU_RBAR or MPU_RASR. However, the region number can be changed by writing to the MPU_RBAR with the VALID bit set to 1; see “MPU Region Base Address Register” . This write updates the value of the REGION field.

11.11.2.4 MPU Region Base Address Register

Name: MPU_RBAR

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			VALID	REGION			

The MPU_RBAR defines the base address of the MPU region selected by the MPU_RNR, and can update the value of the MPU_RNR.

Write MPU_RBAR with the VALID bit set to 1 to change the current region number and update the MPU_RNR.

- **ADDR: Region Base Address**

Software must ensure that the value written to the ADDR field aligns with the size of the selected region (SIZE field in the MPU_RASR).

If the region size is configured to 4 GB, in the MPU_RASR, there is no valid ADDR field. In this case, the region occupies the complete memory map, and the base address is 0x00000000.

The base address is aligned to the size of the region. For example, a 64 KB region must be aligned on a multiple of 64 KB, for example, at 0x00010000 or 0x00020000.

- **VALID: MPU Region Number Valid**

Write:

0: MPU_RNR not changed, and the processor updates the base address for the region specified in the MPU_RNR, and ignores the value of the REGION field.

1: The processor updates the value of the MPU_RNR to the value of the REGION field, and updates the base address for the region specified in the REGION field.

Always reads as zero.

- **REGION: MPU Region**

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

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

11.11.2.5 MPU Region Attribute and Size Register

Name: MPU_RASR

Access: Read/Write

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

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

15	14	13	12	11	10	9	8
SRD							

7	6	5	4	3	2	1	0
–	–	SIZE					ENABLE

The MPU_RASR defines the region size and memory attributes of the MPU region specified by the MPU_RNR, and enables that region and any subregions.

MPU_RASR is accessible using word or halfword accesses:

- The most significant halfword holds the region attributes.
- The least significant halfword holds the region size, and the region and subregion enable bits.

- **XN: Instruction Access Disable**

0: Instruction fetches enabled.

1: Instruction fetches disabled.

- **AP: Access Permission**

See Table 11-39.

- **TEX, C, B: Memory Access Attributes**

See Table 11-37.

- **S: Shareable**

See Table 11-37.

- **SRD: Subregion Disable**

For each bit in this field:

0: Corresponding subregion is enabled.

1: Corresponding subregion is disabled.

See “Subregions” for more information.

Region sizes of 128 bytes and less do not support subregions. When writing the attributes for such a region, write the SRD field as 0x00.

- **SIZE: Size of the MPU Protection Region**

The minimum permitted value is 3 (b00010).

The SIZE field defines the size of the MPU memory region specified by the MPU_RNR. as follows:

(Region size in bytes) = $2^{(SIZE+1)}$

The smallest permitted region size is 32B, corresponding to a SIZE value of 4. The table below gives an example of SIZE values, with the corresponding region size and value of N in the MPU_RBAR.

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

Note: 1. In the MPU_RBAR; see “MPU Region Base Address Register”

- **ENABLE: Region Enable**

Note: For information about access permission, see “MPU Access Permission Attributes” .

11.11.2.6 MPU Region Base Address Register Alias 1

Name: MPU_RBAR_A1

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			VALID	REGION			

The MPU_RBAR defines the base address of the MPU region selected by the MPU_RNR, and can update the value of the MPU_RNR.

Write MPU_RBAR with the VALID bit set to 1 to change the current region number and update the MPU_RNR.

- **ADDR: Region Base Address**

Software must ensure that the value written to the ADDR field aligns with the size of the selected region.

The value of N depends on the region size. The ADDR field is bits[31:N] of the MPU_RBAR. The region size, as specified by the SIZE field in the MPU_RASR, defines the value of N:

N = Log2(Region size in bytes),

If the region size is configured to 4 GB, in the MPU_RASR, there is no valid ADDR field. In this case, the region occupies the complete memory map, and the base address is 0x00000000.

The base address is aligned to the size of the region. For example, a 64 KB region must be aligned on a multiple of 64 KB, for example, at 0x00010000 or 0x00020000.

- **VALID: MPU Region Number Valid**

Write:

0: MPU_RNR not changed, and the processor updates the base address for the region specified in the MPU_RNR, and ignores the value of the REGION field.

1: The processor updates the value of the MPU_RNR to the value of the REGION field, and updates the base address for the region specified in the REGION field.

Always reads as zero.

- **REGION: MPU Region**

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

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

11.11.2.7 MPU Region Attribute and Size Register Alias 1

Name: MPU_RASR_A1

Access: Read/Write

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

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

15	14	13	12	11	10	9	8
SRD							

7	6	5	4	3	2	1	0
–	–	SIZE					ENABLE

The MPU_RASR defines the region size and memory attributes of the MPU region specified by the MPU_RNR, and enables that region and any subregions.

MPU_RASR is accessible using word or halfword accesses:

- The most significant halfword holds the region attributes.
- The least significant halfword holds the region size, and the region and subregion enable bits.

- **XN: Instruction Access Disable**

0: Instruction fetches enabled.

1: Instruction fetches disabled.

- **AP: Access Permission**

See Table 11-39.

- **TEX, C, B: Memory Access Attributes**

See Table 11-37.

- **S: Shareable**

See Table 11-37.

- **SRD: Subregion Disable**

For each bit in this field:

0: Corresponding subregion is enabled.

1: Corresponding subregion is disabled.

See “Subregions” for more information.

Region sizes of 128 bytes and less do not support subregions. When writing the attributes for such a region, write the SRD field as 0x00.

- **SIZE: Size of the MPU Protection Region**

The minimum permitted value is 3 (b00010).

The SIZE field defines the size of the MPU memory region specified by the MPU_RNR. as follows:

(Region size in bytes) = $2^{(SIZE+1)}$

The smallest permitted region size is 32B, corresponding to a SIZE value of 4. The table below gives an example of SIZE values, with the corresponding region size and value of N in the MPU_RBAR.

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

Note: 1. In the MPU_RBAR; see “MPU Region Base Address Register”

- **ENABLE: Region Enable**

Note: For information about access permission, see “MPU Access Permission Attributes” .

11.11.2.8 MPU Region Base Address Register Alias 2

Name: MPU_RBAR_A2

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			VALID	REGION			

The MPU_RBAR defines the base address of the MPU region selected by the MPU_RNR, and can update the value of the MPU_RNR.

Write MPU_RBAR with the VALID bit set to 1 to change the current region number and update the MPU_RNR.

- **ADDR: Region Base Address**

Software must ensure that the value written to the ADDR field aligns with the size of the selected region.

The value of N depends on the region size. The ADDR field is bits[31:N] of the MPU_RBAR. The region size, as specified by the SIZE field in the MPU_RASR, defines the value of N:

N = Log2(Region size in bytes),

If the region size is configured to 4 GB, in the MPU_RASR, there is no valid ADDR field. In this case, the region occupies the complete memory map, and the base address is 0x00000000.

The base address is aligned to the size of the region. For example, a 64 KB region must be aligned on a multiple of 64 KB, for example, at 0x00010000 or 0x00020000.

- **VALID: MPU Region Number Valid**

Write:

0: MPU_RNR not changed, and the processor updates the base address for the region specified in the MPU_RNR, and ignores the value of the REGION field.

1: The processor updates the value of the MPU_RNR to the value of the REGION field, and updates the base address for the region specified in the REGION field.

Always reads as zero.

- **REGION: MPU Region**

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

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

11.11.2.9 MPU Region Attribute and Size Register Alias 2

Name: MPU_RASR_A2

Access: Read/Write

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

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

15	14	13	12	11	10	9	8
SRD							

7	6	5	4	3	2	1	0
–	–	SIZE					ENABLE

The MPU_RASR defines the region size and memory attributes of the MPU region specified by the MPU_RNR, and enables that region and any subregions.

MPU_RASR is accessible using word or halfword accesses:

- The most significant halfword holds the region attributes.
- The least significant halfword holds the region size, and the region and subregion enable bits.

- **XN: Instruction Access Disable**

0: Instruction fetches enabled.

1: Instruction fetches disabled.

- **AP: Access Permission**

See Table 11-39.

- **TEX, C, B: Memory Access Attributes**

See Table 11-37.

- **S: Shareable**

See Table 11-37.

- **SRD: Subregion Disable**

For each bit in this field:

0: Corresponding subregion is enabled.

1: Corresponding subregion is disabled.

See “Subregions” for more information.

Region sizes of 128 bytes and less do not support subregions. When writing the attributes for such a region, write the SRD field as 0x00.

- **SIZE: Size of the MPU Protection Region**

The minimum permitted value is 3 (b00010).

The SIZE field defines the size of the MPU memory region specified by the MPU_RNR. as follows:

(Region size in bytes) = $2^{(SIZE+1)}$

The smallest permitted region size is 32B, corresponding to a SIZE value of 4. The table below gives an example of SIZE values, with the corresponding region size and value of N in the MPU_RBAR.

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

Note: 1. In the MPU_RBAR; see “MPU Region Base Address Register”

- **ENABLE: Region Enable**

Note: For information about access permission, see “MPU Access Permission Attributes” .

11.11.2.10 MPU Region Base Address Register Alias 3

Name: MPU_RBAR_A3

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			VALID	REGION			

The MPU_RBAR defines the base address of the MPU region selected by the MPU_RNR, and can update the value of the MPU_RNR.

Write MPU_RBAR with the VALID bit set to 1 to change the current region number and update the MPU_RNR.

- **ADDR: Region Base Address**

Software must ensure that the value written to the ADDR field aligns with the size of the selected region.

The value of N depends on the region size. The ADDR field is bits[31:N] of the MPU_RBAR. The region size, as specified by the SIZE field in the MPU_RASR, defines the value of N:

N = Log2(Region size in bytes),

If the region size is configured to 4 GB, in the MPU_RASR, there is no valid ADDR field. In this case, the region occupies the complete memory map, and the base address is 0x00000000.

The base address is aligned to the size of the region. For example, a 64 KB region must be aligned on a multiple of 64 KB, for example, at 0x00010000 or 0x00020000.

- **VALID: MPU Region Number Valid**

Write:

0: MPU_RNR not changed, and the processor updates the base address for the region specified in the MPU_RNR, and ignores the value of the REGION field.

1: The processor updates the value of the MPU_RNR to the value of the REGION field, and updates the base address for the region specified in the REGION field.

Always reads as zero.

- **REGION: MPU Region**

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

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

11.11.2.11 MPU Region Attribute and Size Register Alias 3

Name: MPU_RASR_A3

Access: Read/Write

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

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

15	14	13	12	11	10	9	8
SRD							

7	6	5	4	3	2	1	0
–	–	SIZE					ENABLE

The MPU_RASR defines the region size and memory attributes of the MPU region specified by the MPU_RNR, and enables that region and any subregions.

MPU_RASR is accessible using word or halfword accesses:

- The most significant halfword holds the region attributes.
- The least significant halfword holds the region size, and the region and subregion enable bits.

- **XN: Instruction Access Disable**

0: Instruction fetches enabled.

1: Instruction fetches disabled.

- **AP: Access Permission**

See Table 11-39.

- **TEX, C, B: Memory Access Attributes**

See Table 11-37.

- **S: Shareable**

See Table 11-37.

- **SRD: Subregion Disable**

For each bit in this field:

0: Corresponding subregion is enabled.

1: Corresponding subregion is disabled.

See “Subregions” for more information.

Region sizes of 128 bytes and less do not support subregions. When writing the attributes for such a region, write the SRD field as 0x00.

- **SIZE: Size of the MPU Protection Region**

The minimum permitted value is 3 (b00010).

The SIZE field defines the size of the MPU memory region specified by the MPU_RNR. as follows:

(Region size in bytes) = $2^{(SIZE+1)}$

The smallest permitted region size is 32B, corresponding to a SIZE value of 4. The table below gives an example of SIZE values, with the corresponding region size and value of N in the MPU_RBAR.

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

Note: 1. In the MPU_RBAR; see “MPU Region Base Address Register”

- **ENABLE: Region Enable**

Note: For information about access permission, see “MPU Access Permission Attributes” .

11.12 Floating Point Unit (FPU)

The Cortex-M4F FPU implements the FPv4-SP floating-point extension.

The FPU fully supports single-precision add, subtract, multiply, divide, multiply and accumulate, and square root operations. It also provides conversions between fixed-point and floating-point data formats, and floating-point constant instructions.

The FPU provides floating-point computation functionality that is compliant with the ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic, referred to as the IEEE 754 standard.

The FPU contains 32 single-precision extension registers, which can also be accessed as 16 doubleword registers for load, store, and move operations.

11.12.1 Enabling the FPU

The FPU is disabled from reset. It must be enabled before any floating-point instructions can be used. Example 4-1 shows an example code sequence for enabling the FPU in both privileged and user modes. The processor must be in privileged mode to read from and write to the CPACR.

Example of Enabling the FPU:

```
; CPACR is located at address 0xE000ED88
LDR.W R0, =0xE000ED88
; Read CPACR
LDR R1, [R0]
; Set bits 20-23 to enable CP10 and CP11 coprocessors
ORR R1, R1, #(0xF << 20)
; Write back the modified value to the CPACR
STR R1, [R0]; wait for store to complete
DSB
;reset pipeline now the FPU is enabled
ISB
```

11.12.2 Floating Point Unit (FPU) User Interface

Table 11-42. Floating Point Unit (FPU) Register Mapping

Offset	Register	Name	Access	Reset
0xE000ED88	Coprocessor Access Control Register	CPACR	Read/Write	0x00000000
0xE000EF34	Floating-point Context Control Register	FPCCR	Read/Write	0xC0000000
0xE000EF38	Floating-point Context Address Register	FPCAR	Read/Write	–
–	Floating-point Status Control Register	FPSCR	Read/Write	–
0xE000E01C	Floating-point Default Status Control Register	FPDSCR	Read/Write	0x00000000

11.12.2.1 Coprocessor Access Control Register

Name: CPACR

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
CP11		CP10		–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

The CPACR specifies the access privileges for coprocessors.

- **CP10: Access Privileges for Coprocessor 10**

The possible values of each field are:

0: Access denied. Any attempted access generates a NOCP UsageFault.

1: Privileged access only. An unprivileged access generates a NOCP fault.

2: Reserved. The result of any access is unpredictable.

3: Full access.

- **CP11: Access Privileges for Coprocessor 11**

The possible values of each field are:

0: Access denied. Any attempted access generates a NOCP UsageFault.

1: Privileged access only. An unprivileged access generates a NOCP fault.

2: Reserved. The result of any access is unpredictable.

3: Full access.

11.12.2.2 Floating-point Context Control Register

Name: FPCCR

Access: Read/Write

31	30	29	28	27	26	25	24
ASPEN	LSPEN	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	MONRDY
7	6	5	4	3	2	1	0
–	BFRDY	MMRDY	HFRDY	THREAD	–	USER	LSPACT

The FPCCR sets or returns FPU control data.

- **ASPEN: Automatic Hardware State Preservation And Restoration**

Enables CONTROL bit [2] setting on execution of a floating-point instruction. This results in an automatic hardware state preservation and restoration, for floating-point context, on exception entry and exit.

0: Disable CONTROL bit [2] setting on execution of a floating-point instruction.

1: Enable CONTROL bit [2] setting on execution of a floating-point instruction.

- **LSPEN: Automatic Lazy State Preservation**

0: Disable automatic lazy state preservation for floating-point context.

1: Enable automatic lazy state preservation for floating-point context.

- **MONRDY: Debug Monitor Ready**

0: DebugMonitor is disabled or the priority did not permit to set MON_PEND when the floating-point stack frame was allocated.

1: DebugMonitor is enabled and the priority permitted to set MON_PEND when the floating-point stack frame was allocated.

- **BFRDY: Bus Fault Ready**

0: BusFault is disabled or the priority did not permit to set the BusFault handler to the pending state when the floating-point stack frame was allocated.

1: BusFault is enabled and the priority permitted to set the BusFault handler to the pending state when the floating-point stack frame was allocated.

- **MMRDY: Memory Management Ready**

0: MemManage is disabled or the priority did not permit to set the MemManage handler to the pending state when the floating-point stack frame was allocated.

1: MemManage is enabled and the priority permitted to set the MemManage handler to the pending state when the floating-point stack frame was allocated.

- **HFRDY: Hard Fault Ready**

0: The priority did not permit to set the HardFault handler to the pending state when the floating-point stack frame was allocated.

1: The priority permitted to set the HardFault handler to the pending state when the floating-point stack frame was allocated.

- **THREAD: Thread Mode**

0: The mode was not the Thread Mode when the floating-point stack frame was allocated.

1: The mode was the Thread Mode when the floating-point stack frame was allocated.

- **USER: User Privilege Level**

0: The privilege level was not User when the floating-point stack frame was allocated.

1: The privilege level was User when the floating-point stack frame was allocated.

- **LSPACT: Lazy State Preservation Active**

0: The lazy state preservation is not active.

1: The lazy state preservation is active. The floating-point stack frame has been allocated but saving the state to it has been deferred.

11.12.2.3 Floating-point Context Address Register

Name: FPCAR

Access: Read/Write

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					–	–	–

The FPCAR holds the location of the unpopulated floating-point register space allocated on an exception stack frame.

- **ADDRESS: Location of Unpopulated Floating-point Register Space Allocated on an Exception Stack Frame**

The location of the unpopulated floating-point register space allocated on an exception stack frame.

11.12.2.4 Floating-point Status Control Register

Name: FPSCR

Access: Read/Write

31	30	29	28	27	26	25	24
N	Z	C	V	–	AHP	DN	FZ

23	22	21	20	19	18	17	16
RMode		–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
IDC	–	–	IXC	UFC	OFC	DZC	IOC

The FPSCR provides all necessary User level control of the floating-point system.

- **N: Negative Condition Code Flag**

Floating-point comparison operations update this flag.

- **Z: Zero Condition Code Flag**

Floating-point comparison operations update this flag.

- **C: Carry Condition Code Flag**

Floating-point comparison operations update this flag.

- **V: Overflow Condition Code Flag**

Floating-point comparison operations update this flag.

- **AHP: Alternative Half-precision Control**

0: IEEE half-precision format selected.

1: Alternative half-precision format selected.

- **DN: Default NaN Mode Control**

0: NaN operands propagate through to the output of a floating-point operation.

1: Any operation involving one or more NaNs returns the Default NaN.

- **FZ: Flush-to-zero Mode Control**

0: Flush-to-zero mode disabled. The behavior of the floating-point system is fully compliant with the IEEE 754 standard.

1: Flush-to-zero mode enabled.

- **RMode: Rounding Mode Control**

The encoding of this field is:

0b00: Round to Nearest (RN) mode

0b01: Round towards Plus Infinity (RP) mode.

0b10: Round towards Minus Infinity (RM) mode.

0b11: Round towards Zero (RZ) mode.

The specified rounding mode is used by almost all floating-point instructions.

- **IDC: Input Denormal Cumulative Exception**

IDC is a cumulative exception bit for floating-point exception; see also bits [4:0].

This bit is set to 1 to indicate that the corresponding exception has occurred since 0 was last written to it.

- **IXC: Inexact Cumulative Exception**

IXC is a cumulative exception bit for floating-point exception; see also bit [7].

This bit is set to 1 to indicate that the corresponding exception has occurred since 0 was last written to it.

- **UFC: Underflow Cumulative Exception**

UFC is a cumulative exception bit for floating-point exception; see also bit [7].

This bit is set to 1 to indicate that the corresponding exception has occurred since 0 was last written to it.

- **OFC: Overflow Cumulative Exception**

OFC is a cumulative exception bit for floating-point exception; see also bit [7].

This bit is set to 1 to indicate that the corresponding exception has occurred since 0 was last written to it.

- **DZC: Division by Zero Cumulative Exception**

DZC is a cumulative exception bit for floating-point exception; see also bit [7].

This bit is set to 1 to indicate that the corresponding exception has occurred since 0 was last written to it.

- **IOC: Invalid Operation Cumulative Exception**

IOC is a cumulative exception bit for floating-point exception; see also bit [7].

This bit is set to 1 to indicate that the corresponding exception has occurred since 0 was last written to it.

11.12.2.5 Floating-point Default Status Control Register

Name: FPDSCR

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	AHP	DN	FZ

23	22	21	20	19	18	17	16
RMode		–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

The FPDSCR holds the default values for the floating-point status control data.

- **AHP: FPSCR.AHP Default Value**

Default value for FPSCR.AHP.

- **DN: FPSCR.DN Default Value**

Default value for FPSCR.DN.

- **FZ: FPSCR.FZ Default Value**

Default value for FPSCR.FZ.

- **RMode: FPSCR.RMode Default Value**

Default value for FPSCR.RMode.

11.13 Glossary

This glossary describes some of the terms used in technical documents from ARM.

Abort	A mechanism that indicates to a processor that the value associated with a memory access is invalid. An abort can be caused by the external or internal memory system as a result of attempting to access invalid instruction or data memory.
Aligned	A data item stored at an address that is divisible by the number of bytes that defines the data size is said to be aligned. Aligned words and halfwords have addresses that are divisible by four and two respectively. The terms word-aligned and halfword-aligned therefore stipulate addresses that are divisible by four and two respectively.
Banked register	A register that has multiple physical copies, where the state of the processor determines which copy is used. The Stack Pointer, SP (R13) is a banked register.
Base register	In instruction descriptions, a register specified by a load or store instruction that is used to hold the base value for the instruction's address calculation. Depending on the instruction and its addressing mode, an offset can be added to or subtracted from the base register value to form the address that is sent to memory. *See also* "Index register" .
Big-endian (BE)	Byte ordering scheme in which bytes of decreasing significance in a data word are stored at increasing addresses in memory. *See also* "Byte-invariant" , "Endianness" , "Little-endian (LE)" .
Big-endian memory	Memory in which: a byte or halfword at a word-aligned address is the most significant byte or halfword within the word at that address, a byte at a halfword-aligned address is the most significant byte within the halfword at that address. *See also* "Little-endian memory" .
Breakpoint	A breakpoint is a mechanism provided by debuggers to identify an instruction at which program execution is to be halted. Breakpoints are inserted by the programmer to enable inspection of register contents, memory locations, variable values at fixed points in the program execution to test that the program is operating correctly. Breakpoints are removed after the program is successfully tested.

Byte-invariant	In a byte-invariant system, the address of each byte of memory remains unchanged when switching between little-endian and big-endian operation. When a data item larger than a byte is loaded from or stored to memory, the bytes making up that data item are arranged into the correct order depending on the endianness of the memory access. An ARM byte-invariant implementation also supports unaligned halfword and word memory accesses. It expects multi-word accesses to be word-aligned.
Cache	A block of on-chip or off-chip fast access memory locations, situated between the processor and main memory, used for storing and retrieving copies of often used instructions, data, or instructions and data. This is done to greatly increase the average speed of memory accesses and so improve processor performance.
Condition field	A four-bit field in an instruction that specifies a condition under which the instruction can execute.
Conditional execution	If the condition code flags indicate that the corresponding condition is true when the instruction starts executing, it executes normally. Otherwise, the instruction does nothing.
Context	The environment that each process operates in for a multitasking operating system. In ARM processors, this is limited to mean the physical address range that it can access in memory and the associated memory access permissions.
Coprocessor	A processor that supplements the main processor. Cortex-M4 does not support any coprocessors.
Debugger	A debugging system that includes a program, used to detect, locate, and correct software faults, together with custom hardware that supports software debugging.
Direct Memory Access (DMA)	An operation that accesses main memory directly, without the processor performing any accesses to the data concerned.
Doubleword	A 64-bit data item. The contents are taken as being an unsigned integer unless otherwise stated.
Doubleword-aligned	A data item having a memory address that is divisible by eight.
Endianness	Byte ordering. The scheme that determines the order that successive bytes of a data word are stored in memory. An aspect of the system's memory mapping. *See also* "Little-endian (LE)" and "Big-endian (BE)" .

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* "Big-endian (BE)" , "Byte-invariant" , "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. *See also* "Big-endian memory" .
Load/store architecture	A processor architecture where data-processing operations only operate on register contents, not directly on memory contents.
Memory Protection Unit (MPU)	Hardware that controls access permissions to blocks of memory. An MPU does not perform any address translation.
Prefetching	In pipelined processors, the process of fetching instructions from memory to fill up the pipeline before the preceding instructions have finished executing. Prefetching an instruction does not mean that the instruction has to be executed.
Preserved	Preserved by writing the same value back that has been previously read from the same field on the same processor.
Read	Reads are defined as memory operations that have the semantics of a load. Reads include the Thumb instructions LDM, LDR, LDRSH, LDRH, LDRSB, LDRB, and POP.
Region	A partition of memory space.
Reserved	A field in a control register or instruction format is reserved if the field is to be defined by the implementation, or produces Unpredictable results if the contents of the field are not zero. These fields are reserved for use in future extensions of the architecture or are implementation-specific. All reserved bits not used by the implementation must be written as 0 and read as 0.
Thread-safe	In a multi-tasking environment, thread-safe functions use safeguard mechanisms when accessing shared resources, to ensure correct operation without the risk of shared access conflicts.
Thumb instruction	One or two halfwords that specify an operation for a processor to perform. Thumb instructions must be halfword-aligned.

Unaligned	A data item stored at an address that is not divisible by the number of bytes that defines the data size is said to be unaligned. For example, a word stored at an address that is not divisible by four.
Undefined	Indicates an instruction that generates an Undefined instruction exception.
Unpredictable	One cannot rely on the behavior. Unpredictable behavior must not represent security holes. Unpredictable behavior must not halt or hang the processor, or any parts of the system.
Warm reset	Also known as a core reset. Initializes the majority of the processor excluding the debug controller and debug logic. This type of reset is useful if debugging features of a processor.
WA	*See* “Write-allocate (WA)” .
WB	*See* “Write-back (WB)” .
Word	A 32-bit data item.
Write	Writes are defined as operations that have the semantics of a store. Writes include the Thumb instructions STM, STR, STRH, STRB, and PUSH.
Write-allocate (WA)	In a write-allocate cache, a cache miss on storing data causes a cache line to be allocated into the cache.
Write-back (WB)	In a write-back cache, data is only written to main memory when it is forced out of the cache on line replacement following a cache miss. Otherwise, writes by the processor only update the cache. This is also known as copyback.
Write buffer	A block of high-speed memory, arranged as a FIFO buffer, between the data cache and main memory, whose purpose is to optimize stores to main memory.
Write-through (WT)	In a write-through cache, data is written to main memory at the same time as the cache is updated.

12. Debug and Test Features

12.1 Description

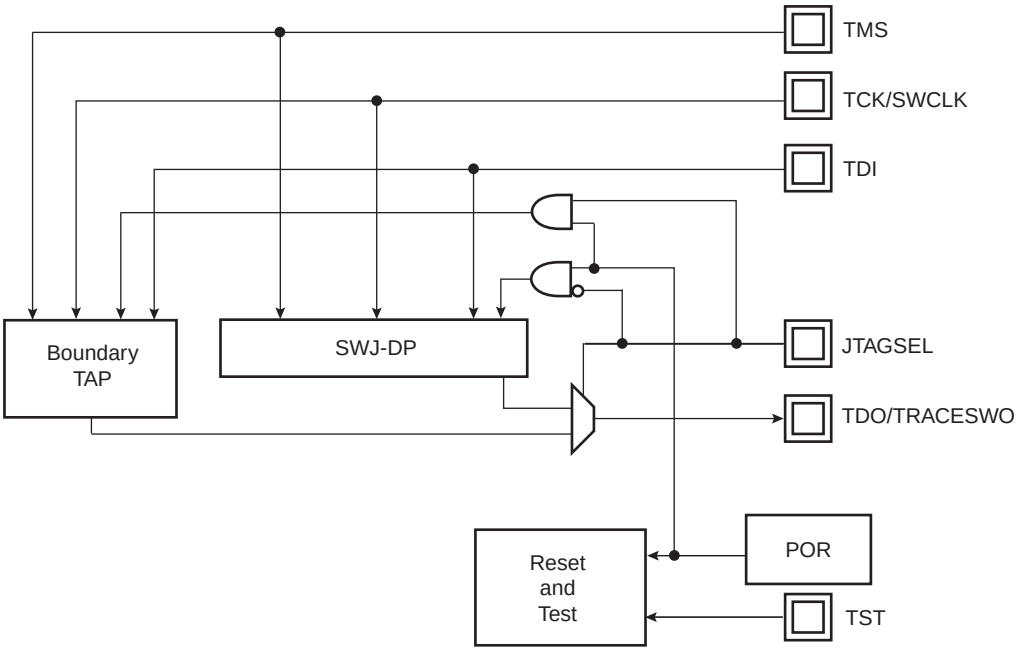
The SAM4 Series Microcontrollers feature a number of complementary debug and test capabilities. The Serial Wire/JTAG Debug Port (SWJ-DP) combining a Serial Wire Debug Port (SW-DP) and JTAG Debug (JTAG-DP) port is used for standard debugging functions, such as downloading code and single-stepping through programs. It also embeds a serial wire trace.

12.2 Embedded Characteristics

- Debug access to all memory and registers in the system, including Cortex-M4 register bank when the core is running, halted, or held in reset.
- Serial Wire Debug Port (SW-DP) and Serial Wire JTAG Debug Port (SWJ-DP) debug access
- Flash Patch and Breakpoint (FPB) unit for implementing breakpoints and code patches
- Data Watchpoint and Trace (DWT) unit for implementing watchpoints, data tracing, and system profiling
- Instrumentation Trace Macrocell (ITM) for support of **printf** style debugging
- IEEE1149.1 JTAG Boundary-scan on all digital pins

12.3 Debug and Test Block Diagram

Figure 12-1. Debug and Test Block Diagram

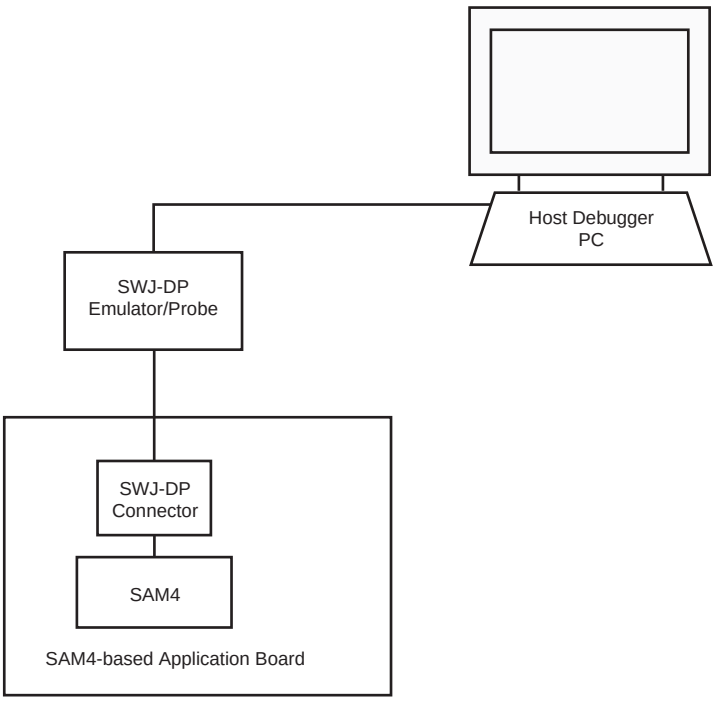


12.4 Application Examples

12.4.1 Debug Environment

Figure 12-2 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.

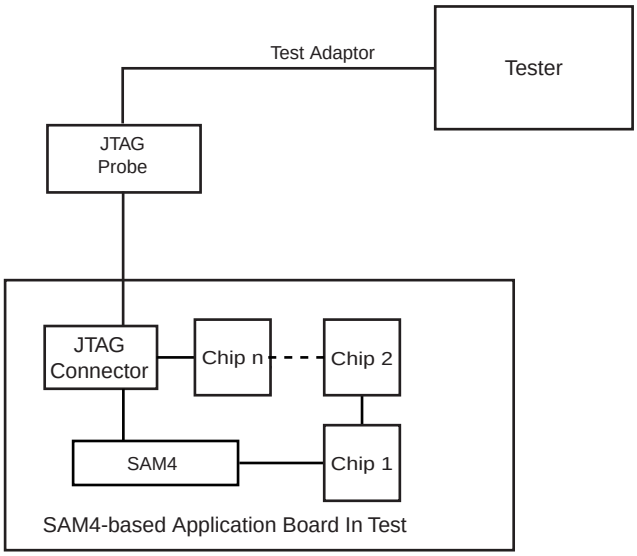
Figure 12-2. Application Debug Environment Example



12.4.2 Test Environment

Figure 12-3 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 12-3. Application Test Environment Example



12.5 Debug and Test Pin Description

Table 12-1. 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	
TMS/SWDIO	Test Mode Select/Serial Wire Input/Output	Input	
JTAGSEL	JTAG Selection	Input	High

12.6 Functional Description

12.6.1 Test Pin

The TST pin is used for JTAG Boundary Scan Manufacturing Test or Fast Flash programming mode of the SAM4E series. The TST pin integrates a permanent pull-down resistor of about 15 kΩ to GND, so that it can be left unconnected for normal operations. To enter fast programming mode, see Section 21. “Fast Flash Programming Interface (FFPI)”.

12.6.2 NRST Pin

The NRST pin is bidirectional. It is handled by the on-chip reset controller and can be driven low to provide a reset signal to the external components or asserted low externally to reset the microcontroller. It will reset the Core and the peripherals except the Backup region (RTC, RTT and Supply Controller). There is no constraint on the length of the reset pulse and the reset controller can guarantee a minimum pulse length. The NRST pin integrates a permanent pull-up resistor to VDDIO of about 100 kΩ. By default, the NRST pin is configured as an input.

12.6.3 ERASE Pin

The ERASE pin is used to reinitialize the Flash content (and some of its NVM bits) to an erased state (all bits read as logic level 1). 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. To avoid unexpected erase at power-up, a minimum ERASE pin assertion time is required. This time is defined in Table 46-68, “AC Flash Characteristics,” on page 1407.

The ERASE pin is a system I/O pin and can be used as a standard I/O. At startup, the ERASE pin is not configured as a PIO pin. If the ERASE pin is used as a standard I/O, start-up level of this pin must be low to prevent unwanted erasing. Also, if the ERASE pin is used as a standard I/O output, asserting the pin to low does not erase the Flash. For details, please refer to Section 10.2 “Peripheral Signal Multiplexing on I/O Lines”.

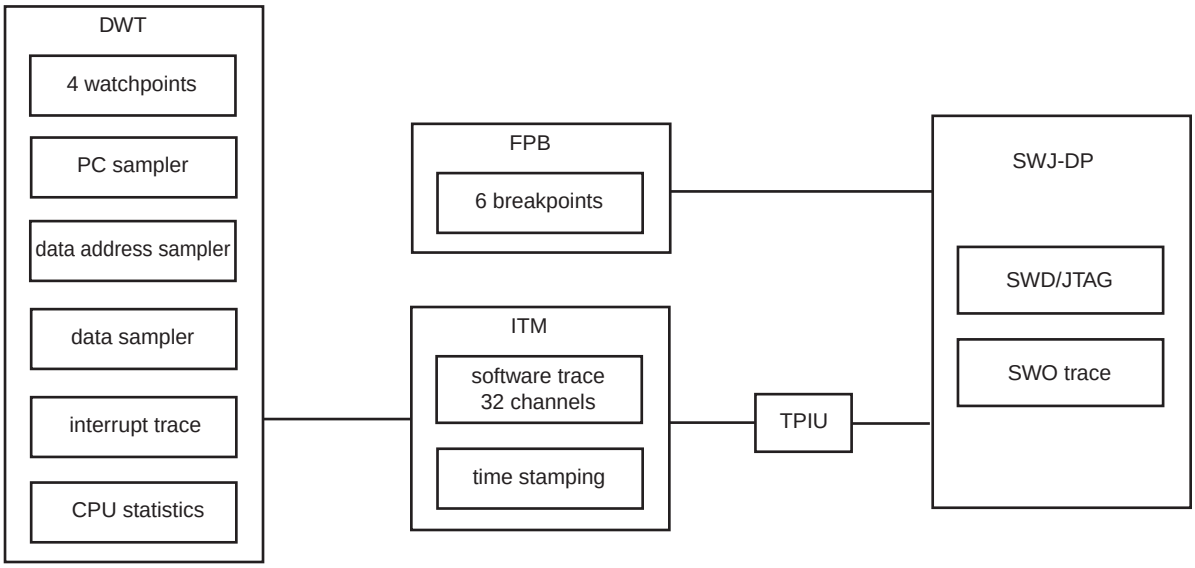
12.6.4 Debug Architecture

Figure 12-4 shows the Debug Architecture used in the SAM4. The Cortex-M4 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-M4 based microcontrollers. For further details on SWJ-DP see the Cortex-M4 technical reference manual.

Figure 12-4. Debug Architecture



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

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

At start-up, SWJ-DP pins are configured in SWJ-DP mode to allow connection with debugging probe.

SWJ-DP pins can be used as standard I/Os to provide users more general input/output pins when the debug port is not needed in the end application. Mode selection between SWJ-DP mode (System IO mode) and general IO mode is performed through the AHB Matrix Special Function Registers (MATRIX_SFR). Configuration of the pad for pull-up, triggers, debouncing and glitch filters is possible regardless of the mode.

The JTAGSEL and PA7 pins are used to select the JTAG boundary scan when JTAGSEL is at high level and PA7 at low 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. The asynchronous trace can only be used with SW-DP, not JTAG-DP.

Table 12-2. 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.

12.6.5.1 SW-DP and JTAG-DP Selection Mechanism

Debug port selection mechanism is done by sending specific **SWDIOTMS** sequence. The JTAG-DP is selected by default after reset.

- Switch from JTAG-DP to SW-DP. The sequence is:
  - Send more than 50 **SWCLKTCK** cycles with **SWDIOTMS** = 1
  - Send the 16-bit sequence on **SWDIOTMS** = 0111100111100111 (0x79E7 MSB first)
  - Send more than 50 **SWCLKTCK** cycles with **SWDIOTMS** = 1
- Switch from SWD to JTAG. The sequence is:
  - Send more than 50 **SWCLKTCK** cycles with **SWDIOTMS** = 1
  - Send the 16-bit sequence on **SWDIOTMS** = 0011110011100111 (0x3CE7 MSB first)
  - Send more than 50 **SWCLKTCK** cycles with **SWDIOTMS** = 1

12.6.6 FPB (Flash Patch Breakpoint)

The FPB:

- Implements hardware breakpoints
- Patches code and data from code space to system space.

The FPB unit contains:

- Two literal comparators for matching against literal loads from Code space, and remapping to a corresponding area in System space.
- Six instruction comparators for matching against instruction fetches from Code space and remapping to a corresponding area in System space.
- Alternatively, comparators can also be configured to generate a Breakpoint instruction to the processor core on a match.

12.6.7 DWT (Data Watchpoint and Trace)

The DWT contains four comparators which can be configured to generate the following:

- PC sampling packets at set intervals
- PC or Data watchpoint packets
- Watchpoint event to halt core

The DWT contains counters for the items that follow:

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

12.6.8 ITM (Instrumentation Trace Macrocell)

The ITM is an application driven trace source that supports **printf** style debugging to trace Operating System (OS) and application events, and emits diagnostic system information. The ITM emits trace information as packets which can be generated by three different sources with several priority levels:

- **Software trace**: Software can write directly to ITM stimulus registers. This can be done using the **printf** function. For more information, refer to Section 12.6.8.1 “How to Configure the ITM”.
- **Hardware trace**: The ITM emits packets generated by the DWT.

- **Time stamping**: Timestamps are emitted relative to packets. The ITM contains a 21-bit counter to generate the timestamp.

12.6.8.1 How to Configure the ITM

The following example describes how to output trace data in asynchronous trace mode.

- Configure the TPIU for asynchronous trace mode (refer to Section 12.6.8.3 “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.

12.6.8.2 Asynchronous Mode

The TPIU is configured in asynchronous mode, trace data are output using the single TRACESWO pin. The TRACESWO signal is multiplexed with the TDO signal of the JTAG Debug Port. As a consequence, asynchronous trace mode is only available when the Serial Wire Debug mode is selected since TDO signal is used in JTAG debug mode.

Two encoding formats are available for the single pin output:

- Manchester encoded stream. This is the reset value.
- NRZ_based UART byte structure

12.6.8.3 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).

12.6.9 IEEE® 1149.1 JTAG Boundary Scan

IEEE 1149.1 JTAG Boundary Scan allows pin-level access independent of the device packaging technology.

IEEE1149.1 JTAG Boundary Scan is enabled when TST is tied to high, PA7 tied low, and JTAGSEL tied to high during power-up. These pins must be maintained in their respective states for the duration of the boundary scan operation.

The SAMPLE, EXTEST and BYPASS functions are implemented. In SWD/JTAG debug mode, the ARM processor responds with a non-JTAG chip ID that identifies the processor. This is not IEEE 1149.1 JTAG-compliant.

It is not possible to switch directly between JTAG Boundary Scan and SWJ Debug Port operations. A chip reset must be performed after JTAGSEL is changed.

A Boundary-scan Descriptor Language (BSDL) file to set up the test is provided on www.atmel.com.

12.6.9.1 JTAG Boundary-scan Register

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

Each SAM4 input/output pin corresponds to a 3-bit register in the BSR. The OUTPUT bit contains data that can be forced on the pad. The INPUT bit facilitates the observability of data applied to the pad. The CONTROL bit selects the direction of the pad.

For more information, please refer to BDSL files available for the SAM4 Series.

12.6.10 ID Code Register

Access: Read-only

31	30	29	28	27	26	25	24
VERSION				PART NUMBER			

23	22	21	20	19	18	17	16
PART NUMBER							

15	14	13	12	11	10	9	8
PART NUMBER				MANUFACTURER IDENTITY			

7	6	5	4	3	2	1	0
MANUFACTURER IDENTITY							1

- **VERSION[31:28]: Product Version Number**

Set to 0x0.

- **PART NUMBER[27:12]: Product Part Number**

Chip Name	Chip ID
SAM4E	0xA3CC_0CE0

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

Set to 0x01F.

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

Set to 0x1.

Chip Name	JTAG ID Code
SAM4E	0x05B3_703F

13. Reset Controller (RSTC)

13.1 Description

The Reset Controller (RSTC), driven by power-on reset (POR) cells, Software, external reset pin and peripheral events, handles all the resets of the system without any external components. It reports which reset occurred last.

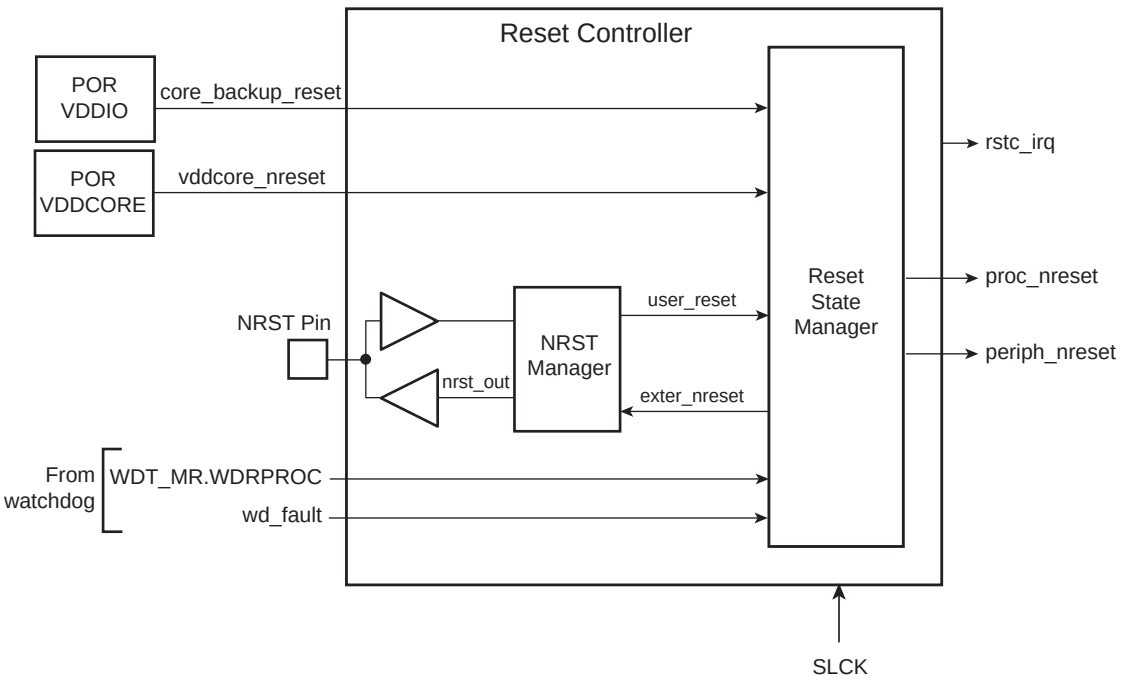
The RSTC also drives independently or simultaneously the external reset and the peripheral and processor resets.

13.2 Embedded Characteristics

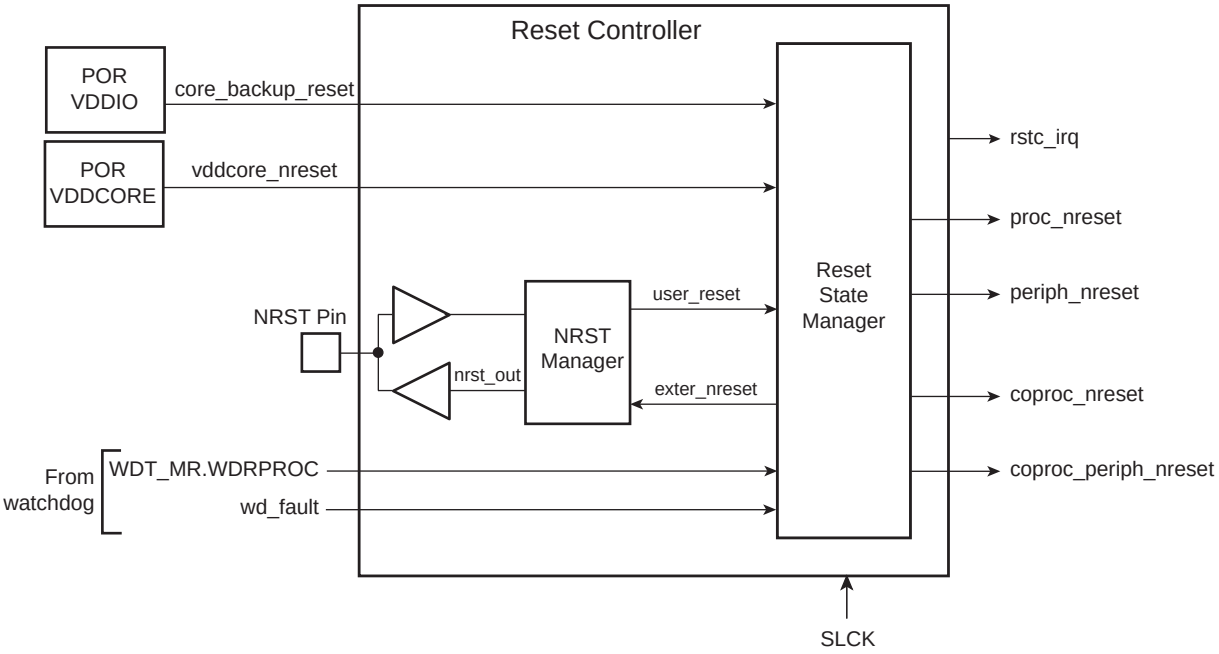
- Driven by Embedded Power-on Reset, Software, External Reset Pin and Peripheral Events
- Management of All System Resets, Including
  - External Devices through the NRST Pin
  - Processor
  - Peripheral Set
- Reset Source Status
  - Status of the Last Reset
  - Either VDDCORE and VDDIO POR Reset, Software Reset, User Reset, Watchdog Reset
- External Reset Signal Control and Shaping

13.3 Block Diagram

Figure 13-1. Reset Controller Block Diagram



13.4 Functional Description



13.4.1 Reset Controller Overview

The Reset Controller is made up of an NRST manager and a reset state manager. It runs at slow clock frequency and generates the following reset signals:

- proc_nreset: processor reset line (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 events generated by peripherals, events on NRST pin, 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.

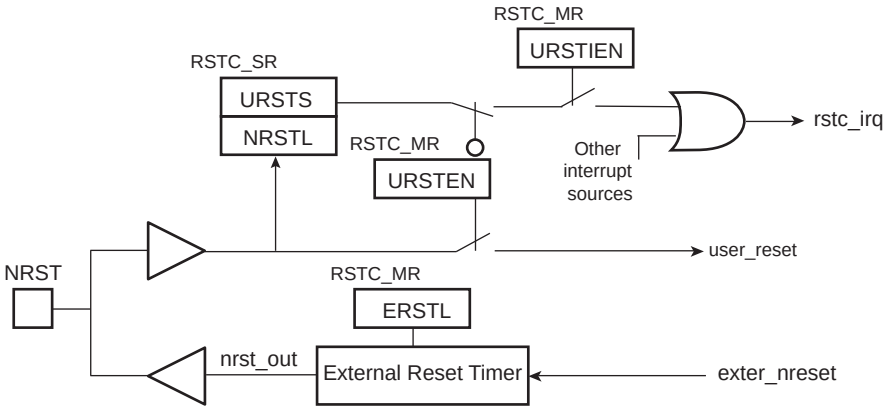
The NRST manager shapes the NRST assertion during a programmable time, thus controlling external device resets.

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

13.4.2 NRST Manager

The NRST manager samples the NRST input pin and drives this pin low when required by the reset state manager. Figure 13-2 shows the block diagram of the NRST manager.

Figure 13-2. NRST Manager



13.4.2.1 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. The NRST pin must be asserted for at least 1 SLCK clock cycle to ensure execution of a user reset.

However, the NRST manager can be programmed to not trigger a reset when an assertion of NRST occurs. Writing a 0 to the URSTEN bit in the 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 the Reset Controller Status Register (RSTC_SR). As soon as the NRST pin is asserted, bit URSTS in the RSTC_SR is written to 1. This bit is cleared only when the RSTC_SR is read.

The Reset Controller can also be programmed to generate an interrupt instead of generating a reset. To do so, set the URSTIEN bit in the RSTC_MR.

13.4.2.2 NRST External Reset Control

The reset state manager asserts the signal exter_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 field ERSTL in the 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.

RSTC_MR is backed up, making it possible to use the ERSTL field to shape the system power-up reset for devices requiring a longer startup time than that of the slow clock oscillator.

13.4.3 Reset States

The reset state manager handles the different reset sources and generates the internal reset signals. It reports the reset status in field RSTTYP of the Status Register (RSTC_SR). The update of RSTC_SR.RSTTYP is performed when the processor reset is released.

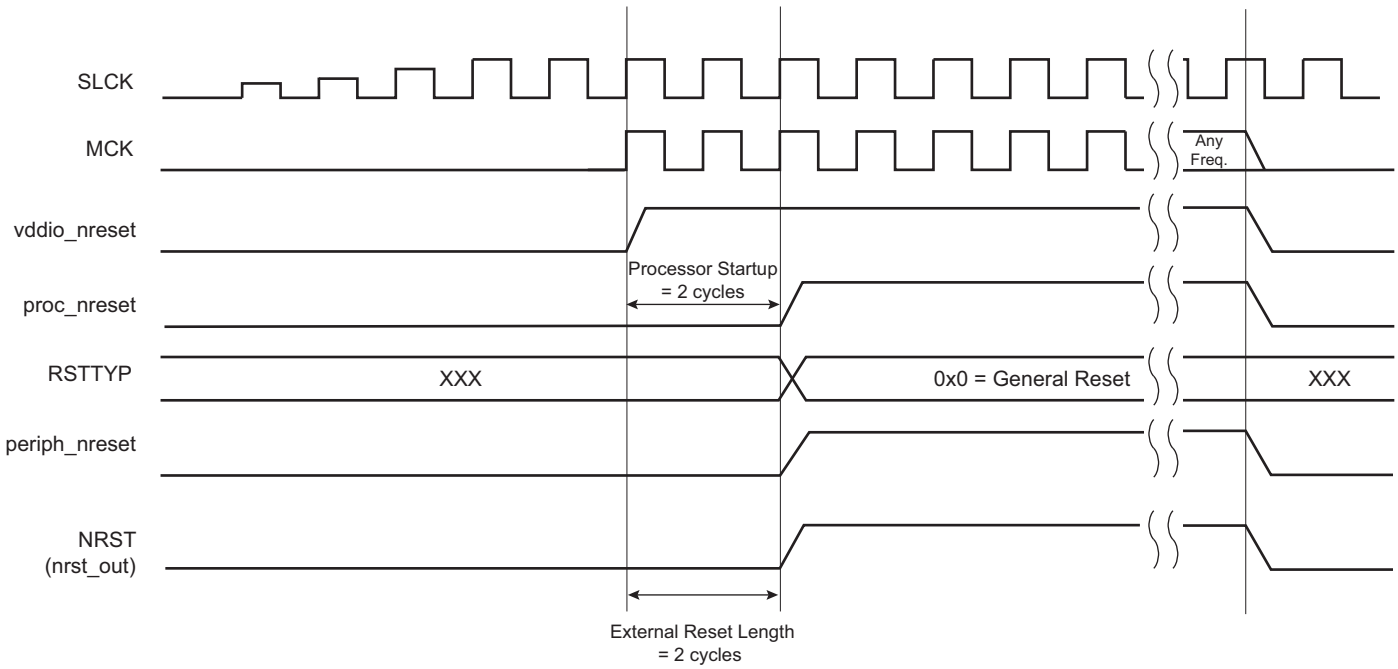
13.4.3.1 General Reset

A general reset occurs when a VDDIO 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 field RSTC_SR.RSTTYP reports a general reset. As the RSTC_MR is written to 0, the NRST line rises two cycles after the vddcore_nreset, as ERSTL defaults at value 0x0.

Figure 13-3 shows how the general reset affects the reset signals.

Figure 13-3. General Reset State



13.4.3.2 Backup Reset

A backup reset occurs when the chip exits from Backup mode. While exiting Backup mode, the vddcore_nreset signal is asserted by the Supply Controller.

Field RSTC_SR.RSTTYP is updated to report a backup reset.

13.4.3.3 Watchdog Reset

The watchdog reset is entered when a watchdog fault occurs. This reset lasts three slow clock cycles.

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

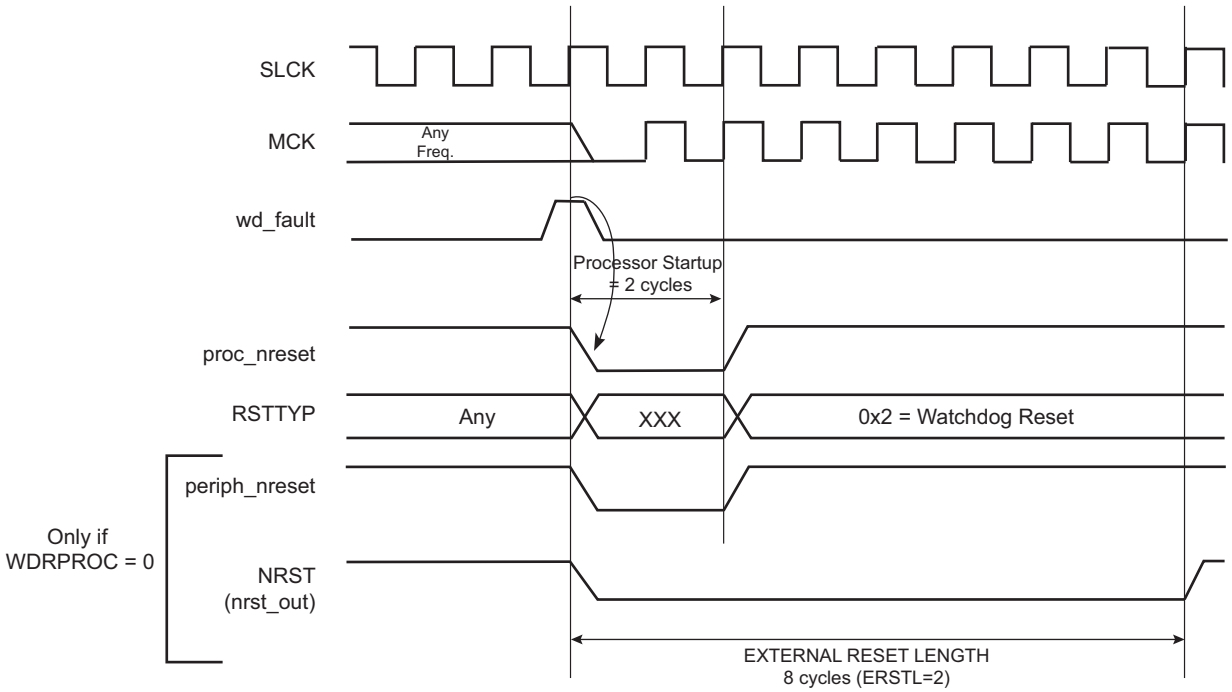
- If WDRPROC = 0, the processor reset and the peripheral reset are asserted. The NRST line is also asserted, depending on how field RSTC_MR.ERSTL is programmed. However, the resulting low level on NRST does not result in a user reset state.
- If WDRPROC = 1, only the processor reset is asserted.

The Watchdog Timer is reset by the proc_nreset signal. As the watchdog fault always causes a processor reset if WDRSTEN in the WDT_MR is written to 1, 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 bit WDT_MR.WDRSTEN is written to 0, the watchdog fault has no impact on the Reset Controller.

After a watchdog overflow occurs, the report on the RSTC_SR.RSTTYP field may differ (either WDT_RST or USER_RST) depending on the external components driving the NRST pin. For example, if the NRST line is driven through a resistor and a capacitor (NRST pin debouncer), the reported value is USER_RST if the low to high transition is greater than one SLCK cycle.

Figure 13-4. Watchdog Reset



13.4.3.4 Software Reset

The Reset Controller offers commands to assert the different reset signals. These commands are performed by writing the Control Register (RSTC_CR) with the following bits at 1:

- RSTC_CR.PROCRST: Writing a 1 to PROCRST resets the processor and the watchdog timer.
- RSTC_CR.PERRST: Writing a 1 to PERRST 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).
- RSTC_CR.EXTRST: Writing a 1 to EXTRST asserts low the NRST pin during a time defined by the field RSTC_MR.ERSTL.

The software reset is entered if at least one of these bits is written to 1 by the software. All these commands can be performed independently or simultaneously. The software reset lasts three slow clock cycles.

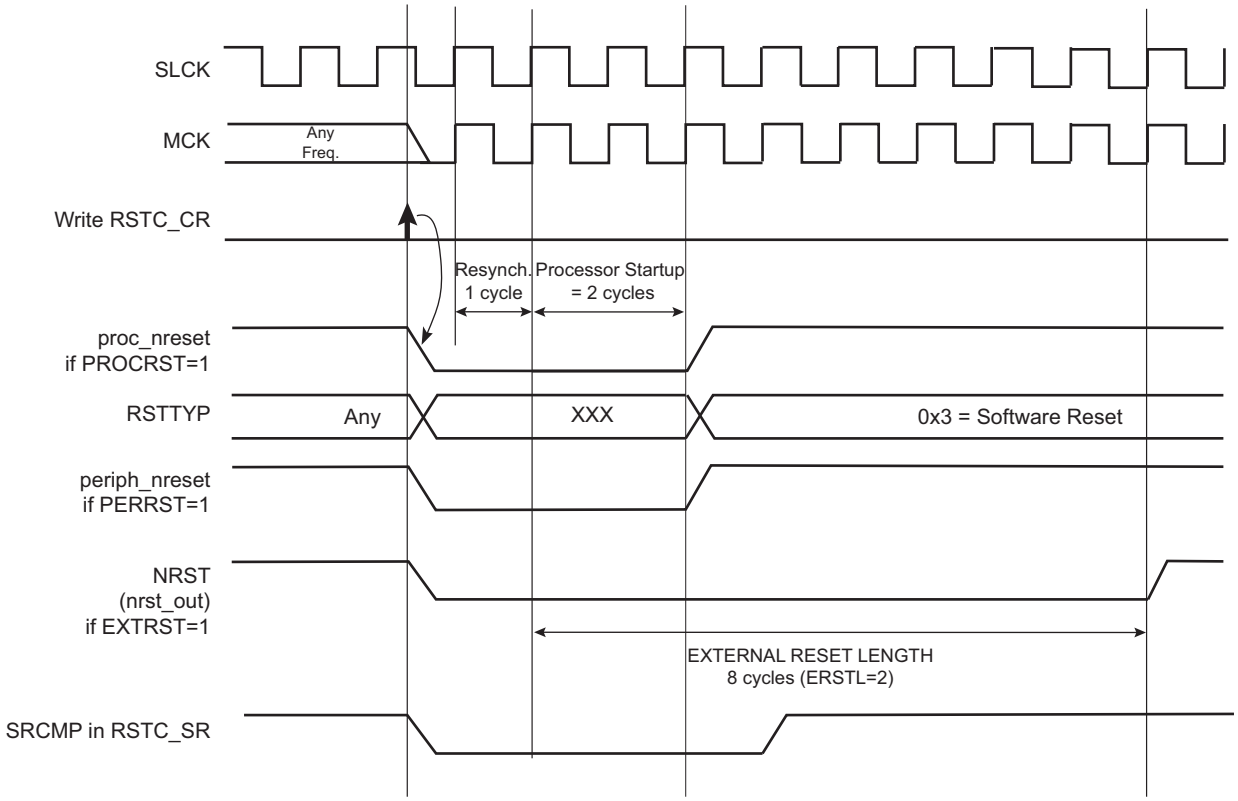
The internal reset signals are asserted as soon as the register write is performed. This is detected on the Master Clock (MCK). They are released when the software reset has ended, i.e., synchronously to SLCK.

If EXTRST is written to 1, the nrst_out signal is asserted depending on the configuration of field RSTC_MR.ERSTL. However, the resulting falling edge on NRST does not lead to a user reset.

If and only if the PROCRST bit is written to 1, the Reset Controller reports the software status in field RSTC_SR.RSTTYP. 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 written to 1 in the RSTC_SR. SRCMP is cleared at the end of the software reset. No other software reset can be performed while the SRCMP bit is written to 1, and writing any value in the RSTC_CR has no effect.

Figure 13-5. Software Reset



13.4.3.5 User Reset

The user reset is entered when a low level is detected on the NRST pin and bit URSTEN in the RSTC_MR is at 1. The NRST input signal is resynchronized with SLCK to ensure proper behavior of the system. Thus, the NRST pin must be asserted for at least 1 SLCK clock cycle to ensure execution of a user reset.

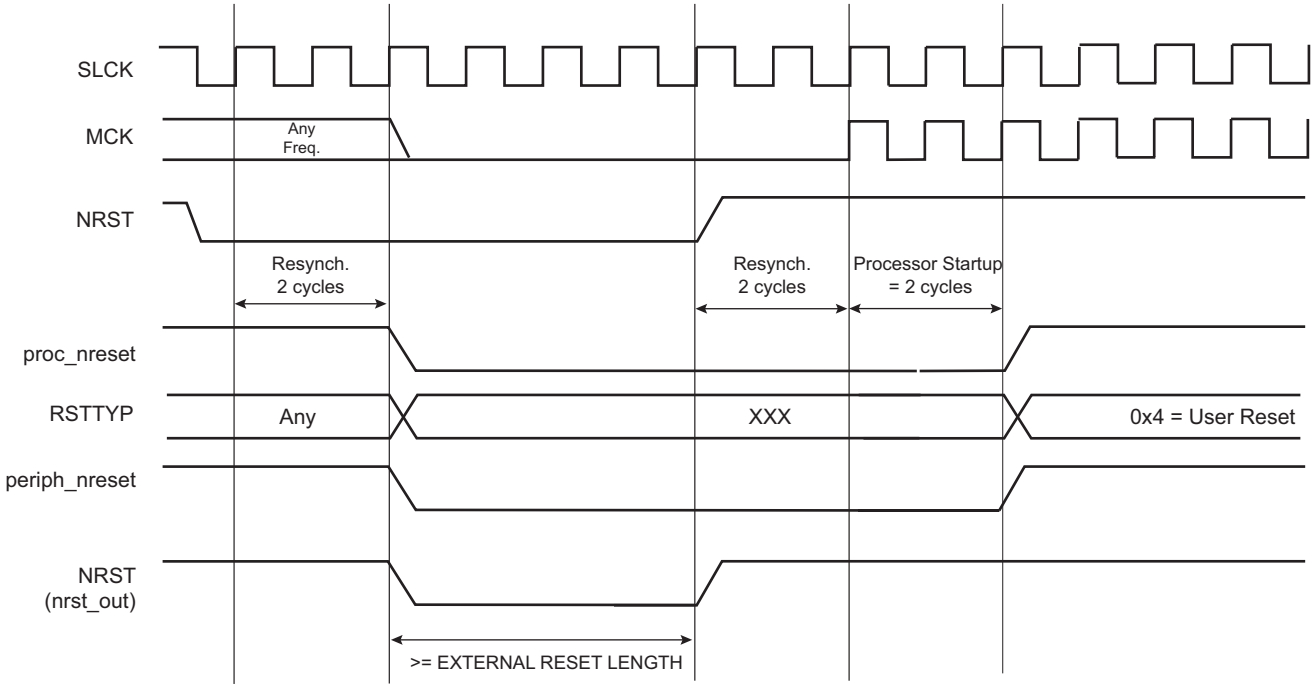
The user reset is entered 2 slow clock cycles (SLCK) after a low level is detected on NRST. The processor reset and the peripheral reset are asserted.

The user reset ends when NRST rises, after a two-cycle resynchronization time and a three-cycle processor startup. The processor clock is re-enabled as soon as NRST is confirmed high.

When the processor reset signal is released, field RSTC_SR.RSTTYP 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 field RSTC_MR.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 13-6. User Reset State



13.4.4 Reset State Priorities

The reset state manager manages the priorities among the different reset sources. The resets are listed in order of priority as follows:

1. General reset
2. Backup reset
3. Watchdog reset
4. Software reset
5. 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.

13.5 Reset Controller (RSTC) User Interface

Table 13-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	RSTC_CR	Write-only	–
0x04	Status Register	RSTC_SR	Read-only	0x0000_0000[1]
0x08	Mode Register	RSTC_MR	Read/Write	0x0000 0001

Note: 1. This value assumes that a general reset has been performed, subject to change if other types of reset are generated.

13.5.1 Reset Controller Control Register

Name: RSTC_CR

Address: 0x400E1800

Access: Write-only

31	30	29	28	27	26	25	24
KEY							

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	EXTRST	PERRST	–	PROCRST

- **PROCRST: Processor Reset**

0: No effect

1: If KEY is correct, resets the processor

- **PERRST: Peripheral Reset**

0: No effect

1: If KEY is correct, resets the peripherals

- **EXTRST: External Reset**

0: No effect

1: If KEY is correct, asserts the NRST pin

- **KEY: System Reset Key**

Value	Name	Description
0xA5	PASSWD	Writing any other value in this field aborts the write operation.

13.5.2 Reset Controller Status Register

Name: RSTC_SR

Address: 0x400E1804

Access: Read-only

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

- **URSTS: User Reset Status**

A high-to-low transition of the NRST pin sets the URSTS bit. This transition is also detected on the MCK rising edge. If the user reset is disabled (URSTEN = 0 in RSTC_MR) and if the interruption is enabled by the URSTIEN bit in the RSTC_MR, the URSTS bit triggers an interrupt. Reading the RSTC_SR resets the URSTS bit and clears the interrupt.

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**

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

Value	Name	Description
0	GENERAL_RST	First power-up reset
1	BACKUP_RST	Return from Backup Mode
2	WDT_RST	Watchdog fault occurred
3	SOFT_RST	Processor reset required by the software
4	USER_RST	NRST pin detected low
5	–	Reserved
6	–	Reserved
7	–	Reserved

- **NRSTL: NRST Pin Level**

This bit registers the NRST pin level sampled on each Master Clock (MCK) rising edge.

- **SRCMP: Software Reset Command in Progress**

When set, this bit 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.

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.

13.5.3 Reset Controller Mode Register

Name: RSTC_MR

Address: 0x400E1808

Access: Read/Write

31	30	29	28	27	26	25	24
KEY							

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	ERSTL			

7	6	5	4	3	2	1	0
–	–	–	URSTIEN	–	–	–	URSTEN

This register can only be written if the WPEN bit is cleared in the System Controller Write Protection Mode Register (SYSC_WPMR).

- **URSTEN: User Reset Enable**

0: The detection of a low level on the NRST pin does not generate a user reset.

1: The detection of a low level on the NRST pin 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. Note that synchronization cycles must also be considered when calculating the actual reset length as previously described.

- **KEY: Write Access Password**

Value	Name	Description
0xA5	PASSWD	Writing any other value in this field aborts the write operation. Always reads as 0.

14. Real-time Timer (RTT)

14.1 Description

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

The RTT can also be configured to be driven by the 1Hz RTC signal, thus taking advantage of a calibrated 1Hz clock.

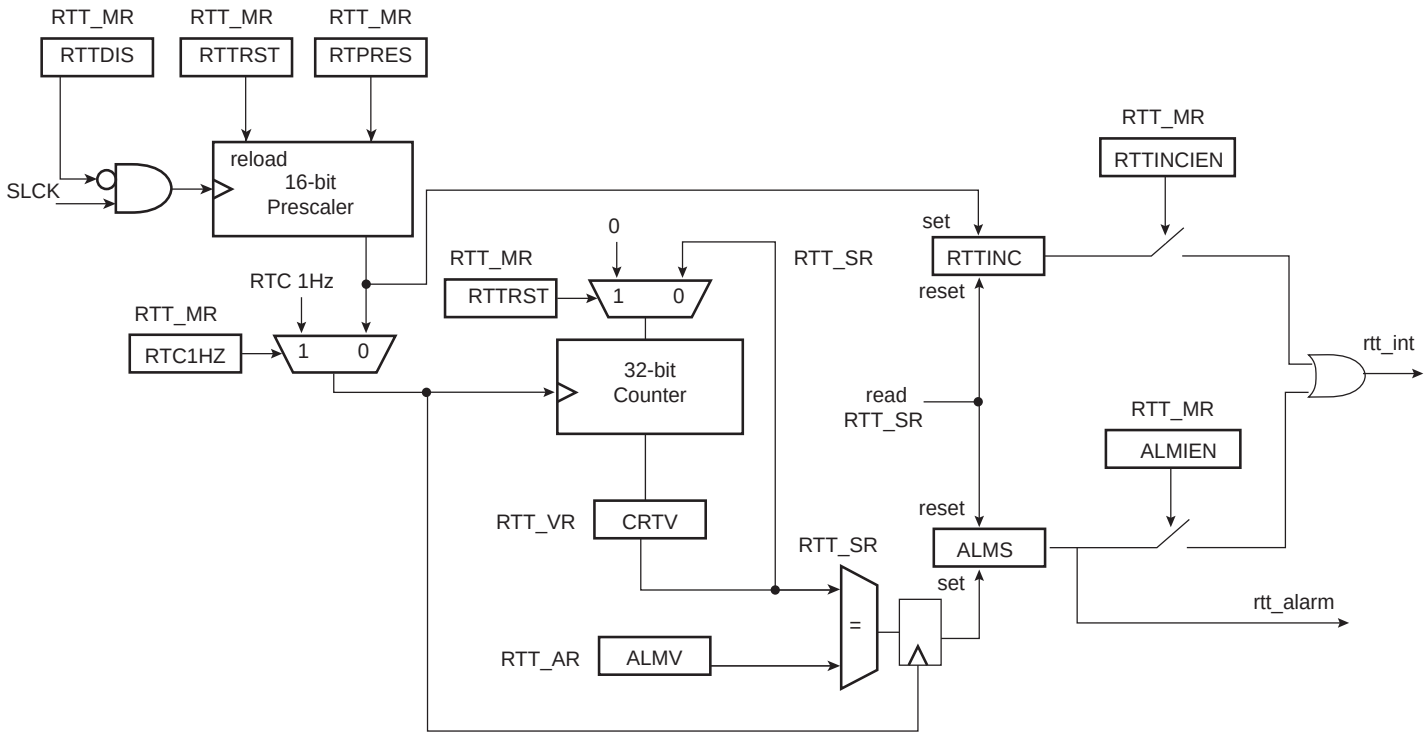
The slow clock source can be fully disabled to reduce power consumption when only an elapsed seconds count is required.

14.2 Embedded Characteristics

- 32-bit Free-running Counter on prescaled slow clock or RTC calibrated 1Hz clock
- 16-bit Configurable Prescaler
- Interrupt on Alarm or Counter Increment

14.3 Block Diagram

Figure 14-1. Real-time Timer



14.4 Functional Description

The programmable 16-bit prescaler value can be configured through the RTPRES field in the “Real-time Timer Mode Register” (RTT_MR).

Configuring the RTPRES field value to 0x8000 (default value) corresponds to feeding the real-time counter with a 1Hz 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. Bit RTTINC in the "Real-time Timer Status Register" (RTT_SR) is set each time there is a prescaler roll-over (see Figure 14-2)

The real-time 32-bit counter can also be supplied by the 1Hz RTC clock. This mode is interesting when the RTC 1Hz is calibrated (CORRECTION field ≠ 0 in RTC_MR) in order to guaranty the synchronism between RTC and RTT counters.

Setting the RTC1HZ bit in the RTT_MR drives the 32-bit RTT counter from the 1Hz RTC clock. In this mode, the RTPRES field has no effect on the 32-bit counter.

The prescaler roll-over generates an increment of the real-time timer counter if RTC1HZ = 0. Otherwise, if RTC1HZ = 1, the real-time timer counter is incremented every second. The RTTINC bit is set independently from the 32-bit counter increment.

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 in RTT_MR.

Programming RTPRES to 1 or 2 is forbidden.

If the RTT is configured to trigger an interrupt, the interrupt occurs two slow clock cycles after reading the RTT_SR. To prevent several executions of the interrupt handler, the interrupt must be disabled in the interrupt handler and re-enabled when the RTT_SR is cleared.

The CRTV field can be read at any time in the "Real-time Timer Value Register" (RTT_VR). As this value can be updated asynchronously with the Master Clock, the CRTV field must be read twice at the same value to read a correct value.

The current value of the counter is compared with the value written in the "Real-time Timer Alarm Register" (RTT_AR). If the counter value matches the alarm, the ALMS bit in the RTT_SR is set. The RTT_AR is set to its maximum value (0xFFFF_FFFF) after a reset.

The ALMS flag is always a source of the RTT alarm signal that may be used to exit the system from low power modes (see Figure 14-1).

The alarm interrupt must be disabled (ALMIEN must be cleared in RTT_MR) when writing a new ALMV value in the RTT_AR.

The RTTINC bit can be used to start a periodic interrupt, the period being one second when the RTPRES field value = 0x8000 and the slow clock = 32.768 kHz.

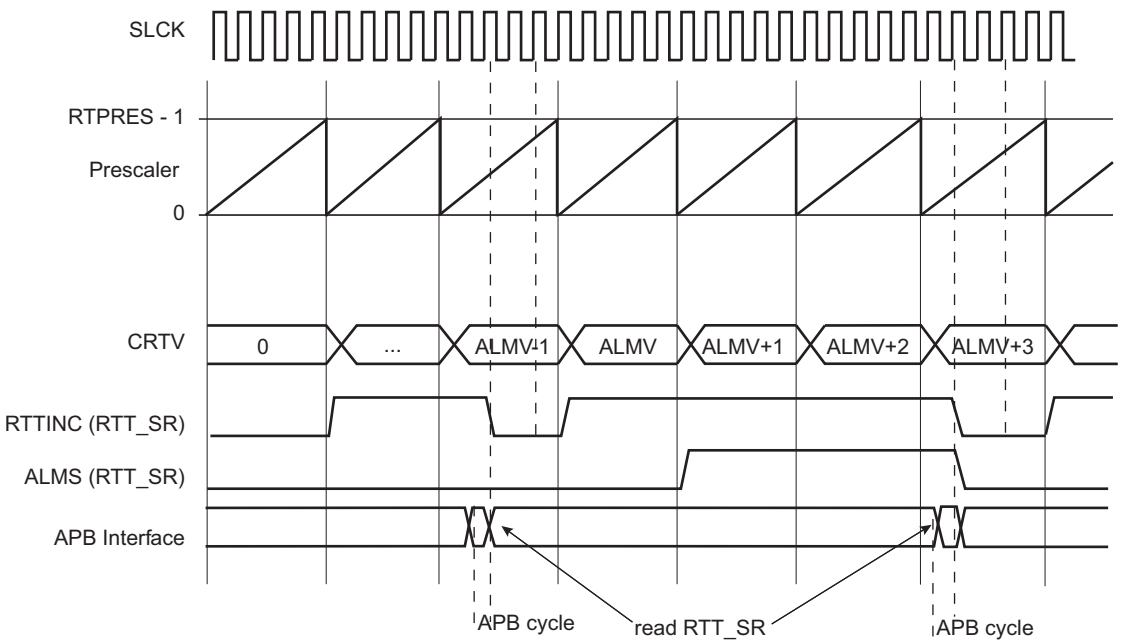
The RTTINCIEN bit must be cleared prior to writing a new RTPRES value in the RTT_MR.

Reading the RTT_SR automatically clears the RTTINC and ALMS bits.

Writing the RTTRST bit in the RTT_MR immediately reloads and restarts the clock divider with the new programmed value. This also resets the 32-bit counter.

When not used, the Real-time Timer can be disabled in order to suppress dynamic power consumption in this module. This can be achieved by setting the RTTDIS bit in the RTT_MR.

Figure 14-2. RTT Counting



14.5 Real-time Timer (RTT) User Interface

Table 14-1. 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

14.5.1 Real-time Timer Mode Register

Name: RTT_MR

Address: 0x400E1830

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	RTC1HZ

23	22	21	20	19	18	17	16
–	–	–	RTTDIS	–	RTTRST	RTTINCIEN	ALMIEN

15	14	13	12	11	10	9	8
RTPRES							

7	6	5	4	3	2	1	0
RTPRES							

- **RTPRES: Real-time Timer Prescaler Value**

Defines the number of SLCK periods required to increment the Real-time timer. RTPRES is defined as follows:

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

RTPRES = 1 or 2: forbidden.

RTPRES ≠ 0,1 or 2: The prescaler period is equal to RTPRES * SLCK periods.

Note: The RTTINCIEN bit must be cleared prior to writing a new RTPRES value.

- **ALMIEN: Alarm Interrupt Enable**

0: The bit ALMS in RTT_SR has no effect on interrupt.

1: The bit ALMS in RTT_SR asserts interrupt.

- **RTTINCIEN: Real-time Timer Increment Interrupt Enable**

0: The bit RTTINC in RTT_SR has no effect on interrupt.

1: The bit RTTINC in RTT_SR asserts interrupt.

- **RTTRST: Real-time Timer Restart**

0: No effect.

1: Reloads and restarts the clock divider with the new programmed value. This also resets the 32-bit counter.

- **RTTDIS: Real-time Timer Disable**

0: The real-time timer is enabled.

1: The real-time timer is disabled (no dynamic power consumption).

Note: RTTDIS is write only.

- **RTC1HZ: Real-Time Clock 1Hz Clock Selection**

0: The RTT 32-bit counter is driven by the 16-bit prescaler roll-over events.

1: The RTT 32-bit counter is driven by the 1Hz RTC clock.

Note: RTC1HZ is write only.

14.5.2 Real-time Timer Alarm Register

Name: RTT_AR

Address: 0x400E1834

Access: Read/Write

31	30	29	28	27	26	25	24
ALMV							

23	22	21	20	19	18	17	16
ALMV							

15	14	13	12	11	10	9	8
ALMV							

7	6	5	4	3	2	1	0
ALMV							

- **ALMV: Alarm Value**

When the CRTV value in RTT_VR equals the ALMV field, the ALMS flag is set in RTT_SR. As soon as the ALMS flag rises, the CRTV value equals ALMV+1 (refer to Figure 14-2).

Note: The alarm interrupt must be disabled (ALMIEN must be cleared in RTT_MR) when writing a new ALMV value.

14.5.3 Real-time Timer Value Register

Name: RTT_VR

Address: 0x400E1838

Access: Read-only

31	30	29	28	27	26	25	24
CRTV							

23	22	21	20	19	18	17	16
CRTV							

15	14	13	12	11	10	9	8
CRTV							

7	6	5	4	3	2	1	0
CRTV							

- **CRTV: Current Real-time Value**

Returns the current value of the Real-time Timer.

Note: As CRTV can be updated asynchronously, it must be read twice at the same value.

14.5.4 Real-time Timer Status Register

Name: RTT_SR

Address: 0x400E183C

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	RTTINC	ALMS

- **ALMS: Real-time Alarm Status (cleared on read)**

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: Prescaler Roll-over Status (cleared on read)**

0: No prescaler roll-over occurred since the last read of the RTT_SR.

1: Prescaler roll-over occurred since the last read of the RTT_SR.

15. Real-time Clock (RTC)

15.1 Description

The Real-time Clock (RTC) peripheral is designed for very low power consumption. For optimal functionality, the RTC requires an accurate external 32.768 kHz clock, which can be provided by a crystal oscillator.

It combines a complete time-of-day clock with alarm and a Gregorian or Persian calendar, complemented by a programmable periodic interrupt. The alarm and calendar registers are accessed by a 32-bit data bus.

The time and calendar values are coded in binary-coded decimal (BCD) format. The time format can be 24-hour mode or 12-hour mode with an AM/PM indicator.

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

A clock divider calibration circuitry can be used to compensate for crystal oscillator frequency variations.

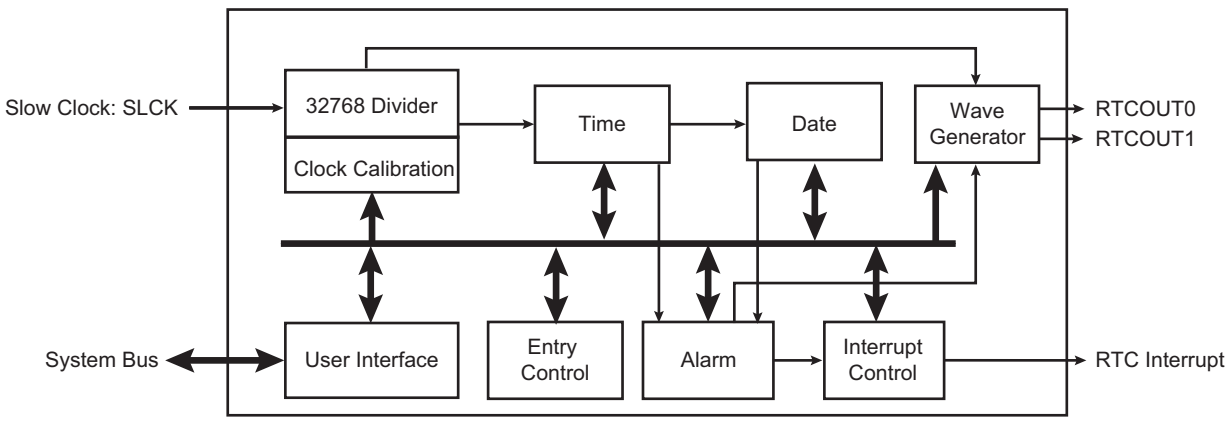
An RTC output can be programmed to generate several waveforms, including a prescaled clock derived from 32.768 kHz.

15.2 Embedded Characteristics

- Full Asynchronous Design for Ultra Low Power Consumption
- Gregorian and Persian Modes Supported
- Programmable Periodic Interrupt
- Safety/security Features:
  - Valid Time and Date Programming Check
  - On-The-Fly Time and Date Validity Check
- Counters Calibration Circuitry to Compensate for Crystal Oscillator Variations
- Waveform Generation
- Register Write Protection

15.3 Block Diagram

Figure 15-1. Real-time Clock Block Diagram



15.4 Product Dependencies

15.4.1 Power Management

The Real-time Clock is continuously clocked at 32.768 kHz. The Power Management Controller has no effect on RTC behavior.

15.4.2 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.

Table 15-1. Peripheral IDs

Instance	ID
RTC	2

15.5 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 reported in RTC Time Register (RTC_TIMR) and RTC Calendar Register (RTC_CALR).

The valid year range is up to 2099 in Gregorian mode (or 1300 to 1499 in Persian mode).

The RTC can operate in 24-hour mode or in 12-hour mode with an AM/PM indicator.

Corrections for leap years are included (all years divisible by 4 being leap years except 1900). This is correct up to the year 2099.

The RTC can generate configurable waveforms on RTCOUT0/1 outputs.

15.5.1 Reference Clock

The reference clock is the 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.

15.5.2 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.

15.5.3 Alarm

The RTC has five programmable fields: month, date, hours, minutes and seconds.

Each of these fields can be enabled or disabled to match the alarm condition:

- If all the fields are enabled, an alarm flag is generated (the corresponding flag is asserted and an interrupt generated if enabled) at a given month, date, hour/minute/second.
- If only the “seconds” field is enabled, then an alarm is generated every minute.

Depending on the combination of fields enabled, a large number of possibilities are available to the user ranging from minutes to 365/366 days.

Hour, minute and second matching alarm (SECEN, MINEN, HOUREN) can be enabled independently of SEC, MIN, HOUR fields.

Note: To change one of the SEC, MIN, HOUR, DATE, MONTH fields, it is recommended to disable the field before changing the value and then re-enable it after the change has been made. This requires up to three accesses to the RTC_TIMALR or RTC_CALALR. The first access clears the enable corresponding to the field to change (SECEN, MINEN, HOUREN, DATEEN, MTHEN). If the field is already cleared, this access is not required. The second access performs the change of the value (SEC, MIN, HOUR, DATE, MONTH). The third access is required to re-enable the field by writing 1 in SECEN, MINEN, HOUREn, DATEEN, MTHEN fields.

15.5.4 Error Checking when Programming

Verification on user interface data is performed when accessing the century, year, month, date, day, hours, minutes, seconds and alarms. A check is performed on illegal BCD entries such as illegal date of the month with regard to the year and century configured.

If one of the time fields is not correct, the data is not loaded into the register/counter and a flag is set in the validity register. The user can not reset this flag. It is reset as soon as an acceptable value is programmed. This avoids any further side effects in the hardware. The same procedure is followed for the alarm.

The following checks are performed:

1. Century (check if it is in range 19–20 or 13–14 in Persian mode)
2. Year (BCD entry check)
3. Date (check range 01–31)
4. Month (check if it is in BCD range 01–12, check validity regarding “date”)
5. Day (check range 1–7)
6. Hour (BCD checks: in 24-hour mode, check range 00–23 and check that AM/PM flag is not set if RTC is set in 24-hour mode; in 12-hour mode check range 01–12)
7. Minute (check BCD and range 00–59)
8. Second (check BCD and range 00–59)

Note: If the 12-hour mode is selected by means of the RTC Mode Register (RTC_MR), a 12-hour value can be programmed and the returned value on RTC_TIMR will be the corresponding 24-hour value. The entry control checks the value of the AM/PM indicator (bit 22 of RTC_TIMR) to determine the range to be checked.

15.5.5 RTC Internal Free Running Counter Error Checking

To improve the reliability and security of the RTC, a permanent check is performed on the internal free running counters to report non-BCD or invalid date/time values.

An error is reported by TDERR bit in the status register (RTC_SR) if an incorrect value has been detected. The flag can be cleared by setting the TDERRCLR bit in the Status Clear Command Register (RTC_SCCR).

Anyway the TDERR error flag will be set again if the source of the error has not been cleared before clearing the TDERR flag. The clearing of the source of such error can be done by reprogramming a correct value on RTC_CALR and/or RTC_TIMR.

The RTC internal free running counters may automatically clear the source of TDERR due to their roll-over (i.e., every 10 seconds for SECONDS[3:0] field in RTC_TIMR). In this case the TDERR is held high until a clear command is asserted by TDERRCLR bit in RTC_SCCR.

15.5.6 Updating Time/Calendar

The update of the time/calendar must be synchronized on a second periodic event by either polling the RTC_SR.SEC status bit or by enabling the SECEN interrupt in the RTC_IER register.

Once the second event occurs, the user must stop the RTC by setting the corresponding field in the Control Register (RTC_CR). 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).

The ACKUPD bit must then be read to 1 by either polling the RTC_SR or by enabling the ACKUPD interrupt in the RTC_IER. Once ACKUPD is read to 1, it is mandatory to clear this flag by writing the corresponding bit in the RTC_SCCR, after which the user can write to the Time Register, the Calendar Register, or both.

Once the update is finished, the user must write UPDTIM and/or UPDCAL to 0 in the RTC_CR.

The timing sequence of the time/calendar update is described in Figure 15-2.

When entering the Programming mode of the calendar fields, the time fields remain enabled. When entering the Programming mode of the time fields, both the time and the calendar fields are stopped. This is due to the location of the calendar logical 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 for at least one second after resetting the UPDTIM/UPDCAL bit in the RTC_CR before setting these bits again. This is done by waiting for the SEC flag in the RTC_SR before setting the UPDTIM/UPDCAL bit. After resetting UPDTIM/UPDCAL, the SEC flag must also be cleared.

Figure 15-2. Time/Calendar Update Timing Diagram

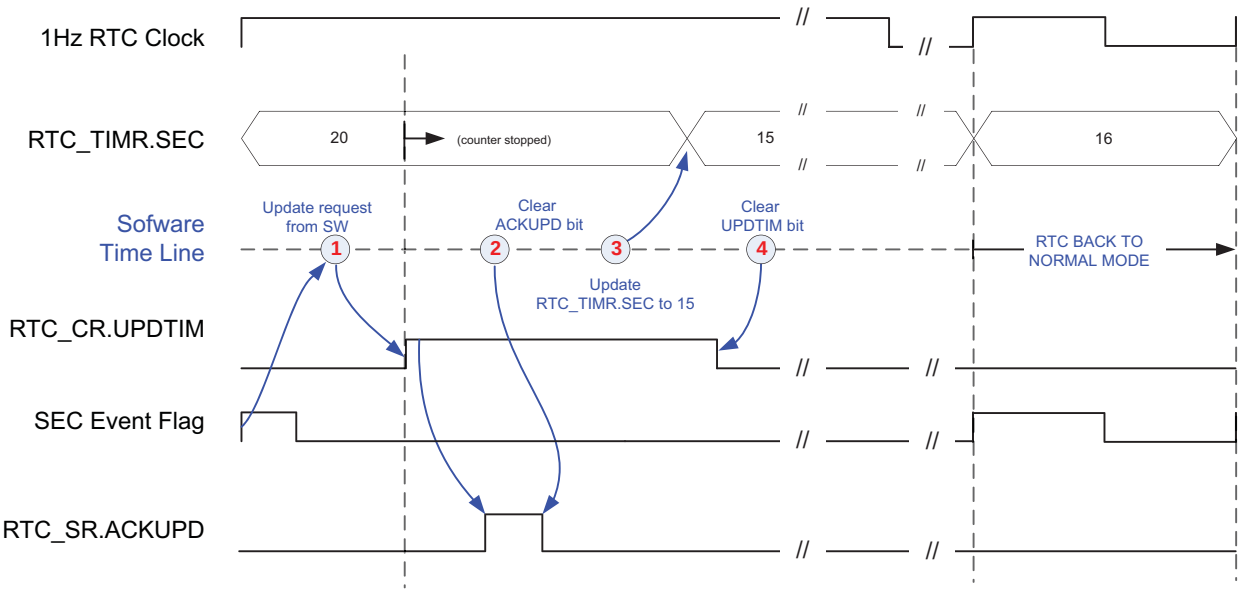
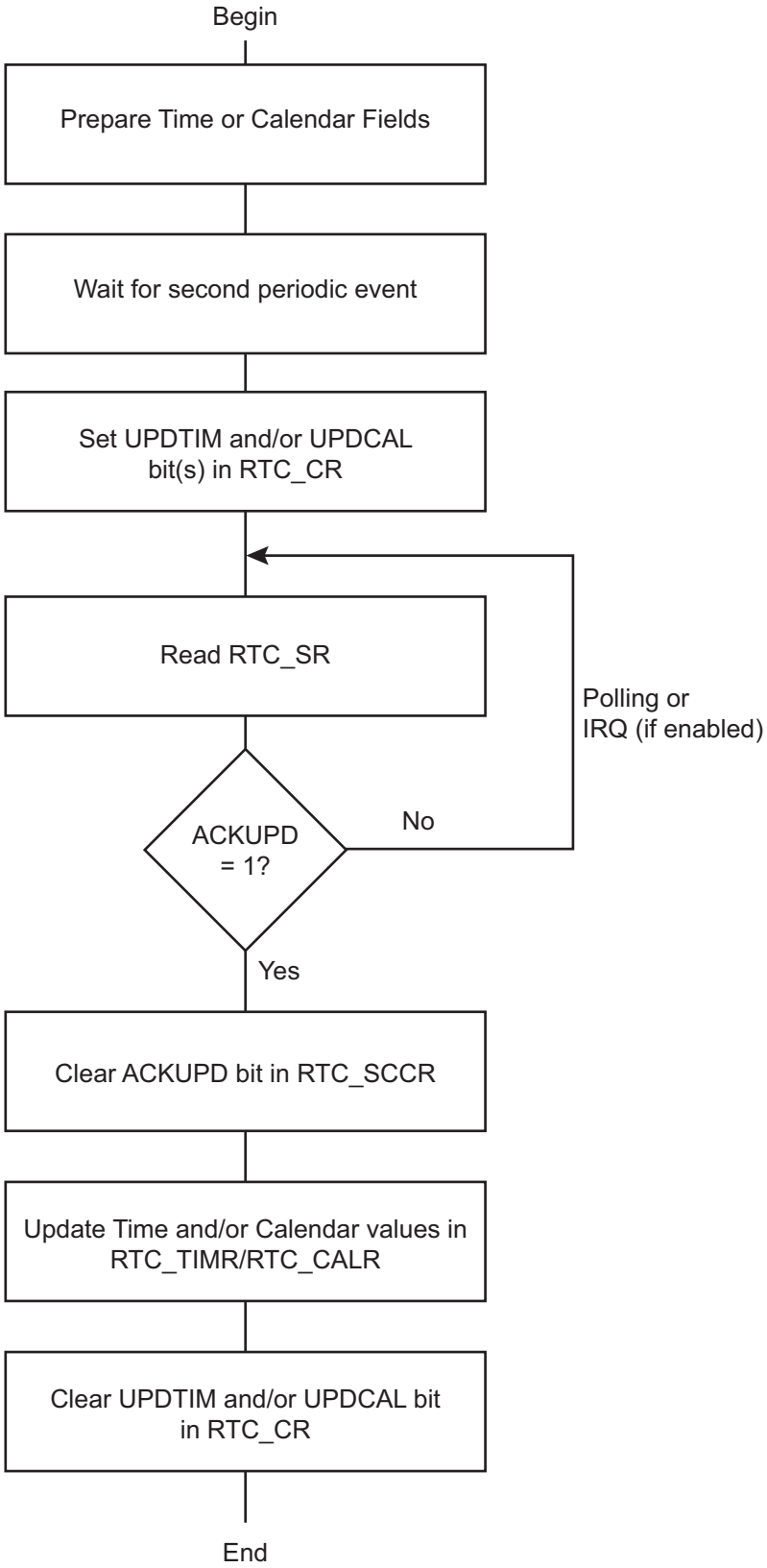


Figure 15-3. Gregorian and Persian Modes Update Sequence



15.5.7 RTC Accurate Clock Calibration

The crystal oscillator that drives the RTC may not be as accurate as expected mainly due to temperature variation. The RTC is equipped with circuitry able to correct slow clock crystal drift.

To compensate for possible temperature variations over time, this accurate clock calibration circuitry can be programmed on-the-fly and also programmed during application manufacturing, in order to correct the crystal frequency accuracy at room temperature (20–25°C). The typical clock drift range at room temperature is ±20 ppm.

In the device operating temperature range, the 32.768 kHz crystal oscillator clock inaccuracy can be up to -200 ppm.

The RTC clock calibration circuitry allows positive or negative correction in a range of 1.5 ppm to 1950 ppm.

The calibration circuitry is fully digital. Thus, the configured correction is independent of temperature, voltage, process, etc., and no additional measurement is required to check that the correction is effective.

If the correction value configured in the calibration circuitry results from an accurate crystal frequency measure, the remaining accuracy is bounded by the values listed below:

- Below 1 ppm, for an initial crystal drift between 1.5 ppm up to 20 ppm, and from 30 ppm to 90 ppm
- Below 2 ppm, for an initial crystal drift between 20 ppm up to 30 ppm, and from 90 ppm to 130 ppm
- Below 5 ppm, for an initial crystal drift between 130 ppm up to 200 ppm

The calibration circuitry does not modify the 32.768 kHz crystal oscillator clock frequency but it acts by slightly modifying the 1 Hz clock period from time to time. The correction event occurs every 1 + [(20 - (19 x HIGHPPM)) x CORRECTION] seconds. When the period is modified, depending on the sign of the correction, the 1 Hz clock period increases or reduces by around 4 ms. Depending on the CORRECTION, NEGPPM and HIGHPPM values configured in RTC_MR, the period interval between two correction events differs.

Figure 15-4. Calibration Circuitry

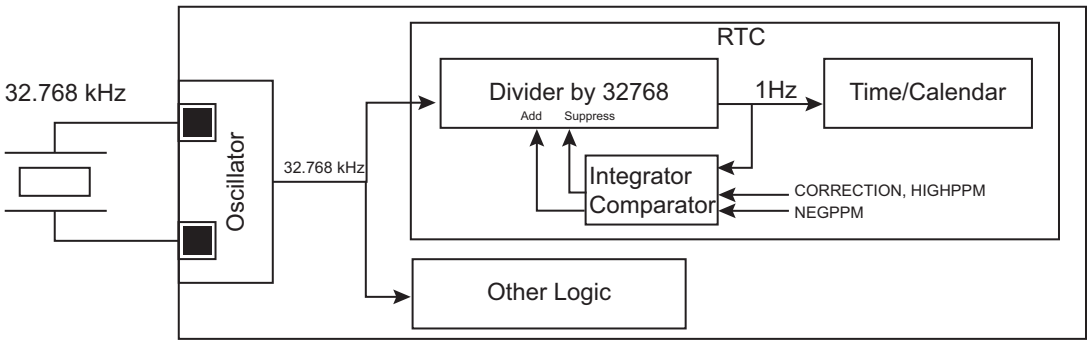
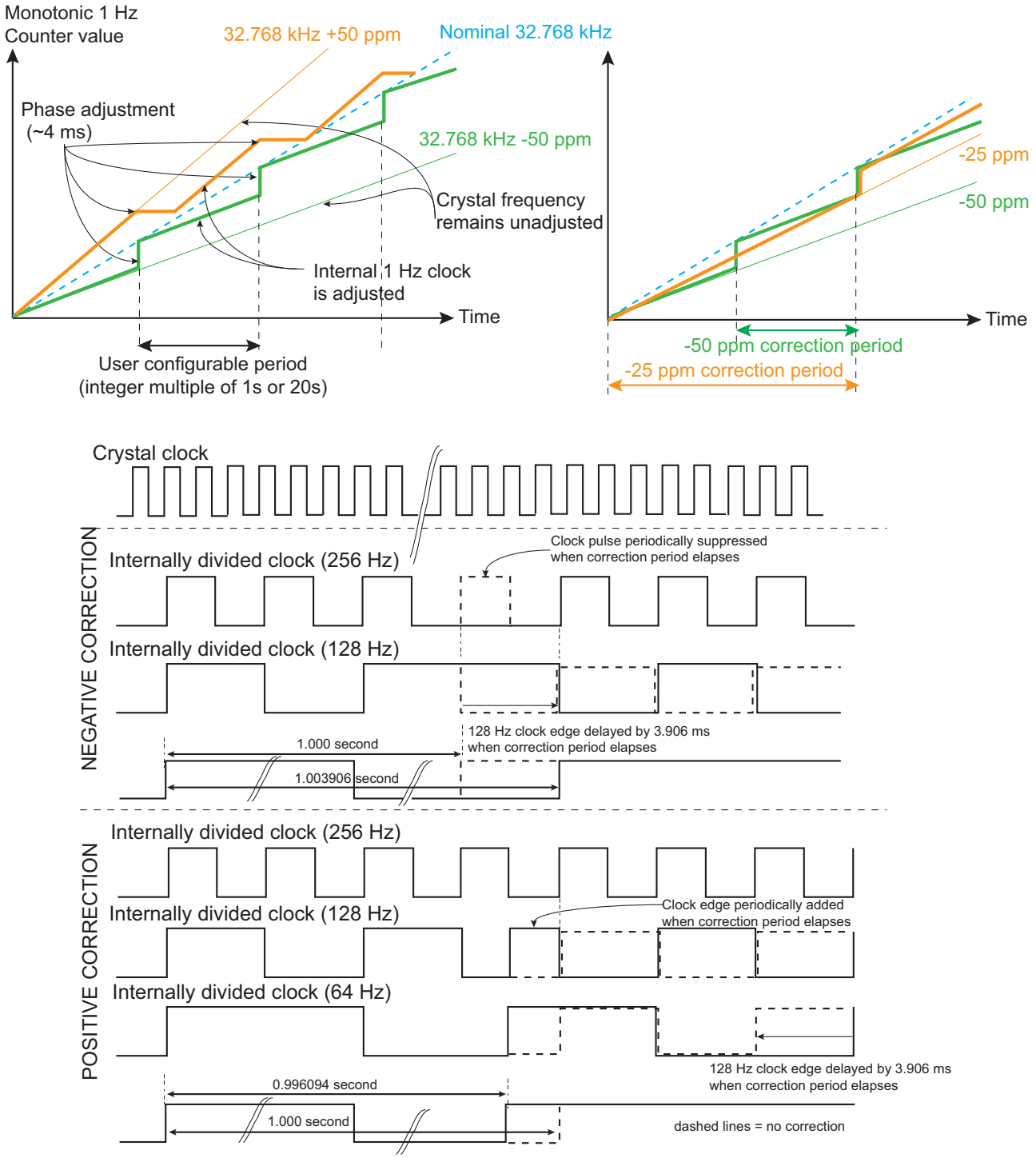


Figure 15-5. Calibration Circuitry Waveforms



The inaccuracy of a crystal oscillator at typical room temperature (±20 ppm at 20–25 °C) can be compensated if a reference clock/signal is used to measure such inaccuracy. This kind of calibration operation can be set up during the final product manufacturing by means of measurement equipment embedding such a reference clock. The correction of value must be programmed into the (RTC_MR), and this value is kept as long as the circuitry is powered (backup area). Removing the backup power supply cancels this calibration. This room temperature calibration can be further processed by means of the networking capability of the target application.

To ease the comparison of the inherent crystal accuracy with the reference clock/signal during manufacturing, an internal prescaled 32.768 kHz clock derivative signal can be assigned to drive RTC output. To accommodate the measure, several clock frequencies can be selected among 1 Hz, 32 Hz, 64 Hz, 512 Hz.

The clock calibration correction drives the internal RTC counters but can also be observed in the RTC output when one of the following three frequencies 1 Hz, 32 Hz or 64 Hz is configured. The correction is not visible in the RTC output if 512 Hz frequency is configured.

In any event, this adjustment does not take into account the temperature variation.

The frequency drift (up to -200 ppm) due to temperature variation can be compensated using a reference time if the application can access such a reference. If a reference time cannot be used, a temperature sensor can be placed close to the crystal oscillator in order to get the operating temperature of the crystal oscillator. Once obtained, the temperature may be converted using a lookup table (describing the accuracy/temperature curve of the crystal oscillator used) and RTC_MR configured accordingly. The calibration can be performed on-the-fly. This adjustment method is not based on a measurement of the crystal frequency/drift and therefore can be improved by means of the networking capability of the target application.

If no crystal frequency adjustment has been done during manufacturing, it is still possible to do it. In the case where a reference time of the day can be obtained through LAN/WAN network, it is possible to calculate the drift of the application crystal oscillator by comparing the values read on RTC Time Register (RTC_TIMR) and programming the HIGHPPM and CORRECTION fields on RTC_MR according to the difference measured between the reference time and those of RTC_TIMR.

15.5.8 Waveform Generation

Waveforms can be generated by the RTC in order to take advantage of the RTC inherent prescalers while the RTC is the only powered circuitry (Low-power mode of operation, Backup mode) or in any active mode. Going into Backup or Low-power operating modes does not affect the waveform generation outputs.

The RTC outputs (RTCOUT0 and RTCOUT1) have a source driver selected among seven possibilities.

The first selection choice sticks the associated output at 0 (This is the reset value and it can be used at any time to disable the waveform generation).

Selection choices 1 to 4 respectively select 1 Hz, 32 Hz, 64 Hz and 512 Hz.

32 Hz or 64 Hz can drive, for example, a TN LCD backplane signal while 1 Hz can be used to drive a blinking character like “:” for basic time display (hour, minute) on TN LCDs.

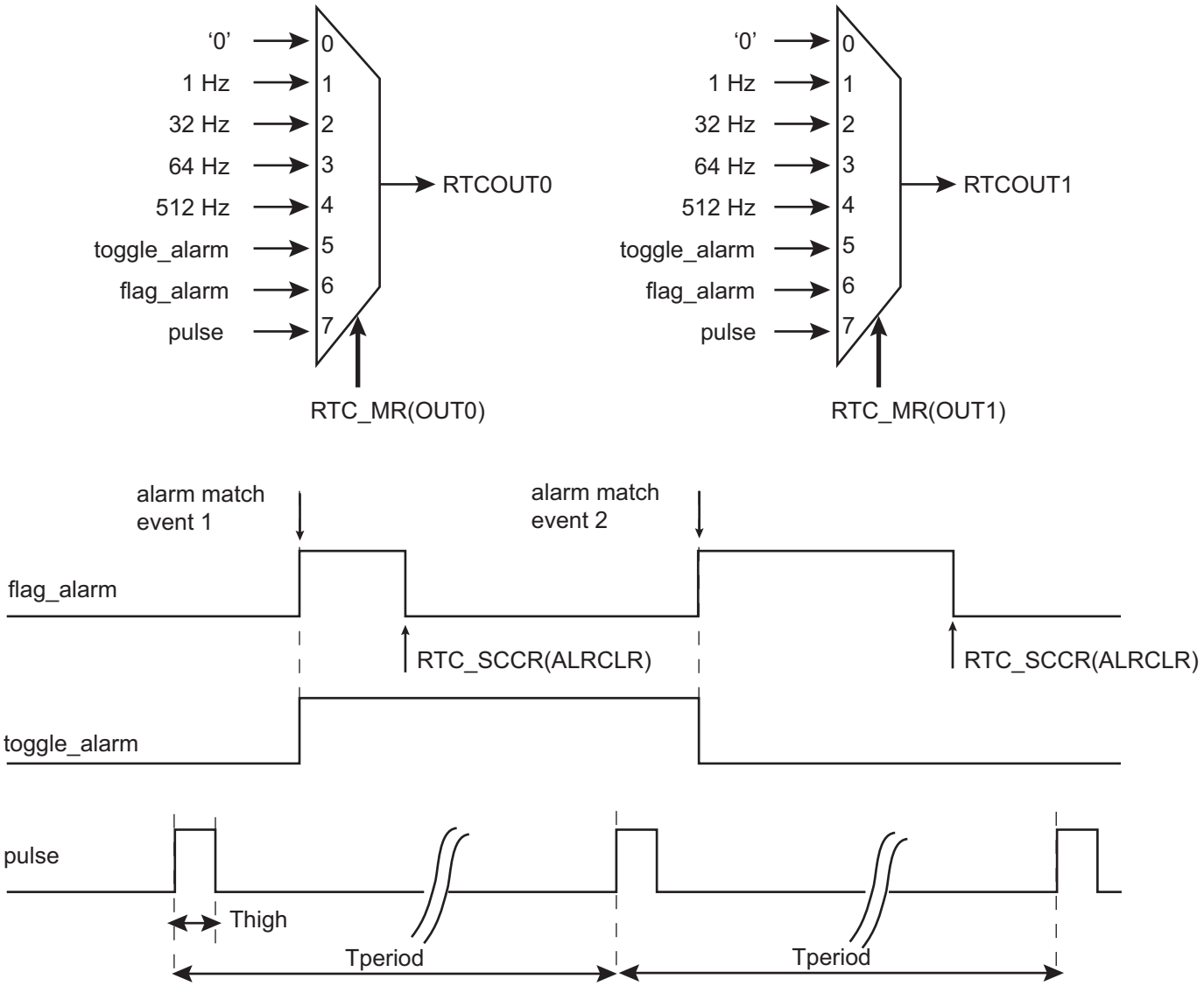
Selection choice 5 provides a toggling signal when the RTC alarm is reached.

Selection choice 6 provides a copy of the alarm flag, so the associated output is set high (logical 1) when an alarm occurs and immediately cleared when software clears the alarm interrupt source.

Selection choice 7 provides a 1 Hz periodic high pulse of 15 µs duration that can be used to drive external devices for power consumption reduction or any other purpose.

PIO lines associated to RTC outputs are automatically selecting these waveforms as soon as RTC_MR corresponding fields OUT0 and OUT1 differ from 0.

Figure 15-6. Waveform Generation



15.6 Real-time Clock (RTC) User Interface

Table 15-2. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	RTC_CR	Read/Write	0x00000000
0x04	Mode Register	RTC_MR	Read/Write	0x00000000
0x08	Time Register	RTC_TIMR	Read/Write	0x00000000
0x0C	Calendar Register	RTC_CALR	Read/Write	0x01a11020
0x10	Time Alarm Register	RTC_TIMALR	Read/Write	0x00000000
0x14	Calendar Alarm Register	RTC_CALALR	Read/Write	0x01010000
0x18	Status Register	RTC_SR	Read-only	0x00000000
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	0x00000000
0x2C	Valid Entry Register	RTC_VER	Read-only	0x00000000
0x30–0xC8	Reserved	–	–	–
0xCC	Reserved	–	–	–
0xD0	Reserved	–	–	–
0xD4–0xF8	Reserved	–	–	–
0xFC	Reserved	–	–	–

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

15.6.1 RTC Control Register

Name: RTC_CR

Address: 0x400E1860

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the System Controller Write Protection Mode Register (SYSC_WPMR).

- **UPDTIM: Update Request Time Register**

0: No effect or, if UPDTIM has been previously written to 1, stops the update procedure.

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 RTC_SR.

- **UPDCAL: Update Request Calendar Register**

0: No effect or, if UPDCAL has been previously written to 1, stops the update procedure.

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 and acknowledged by the bit ACKUPD of the RTC_SR.

- **TIMEVSEL: Time Event Selection**

The event that generates the flag TIMEV in RTC_SR 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	–	Reserved

15.6.2 RTC Mode Register

Name: RTC_MR

Address: 0x400E1864

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	TPERIOD		–	THIGH		
23	22	21	20	19	18	17	16
–	OUT1			–	OUT0		
15	14	13	12	11	10	9	8
HIGHPPM	CORRECTION						
7	6	5	4	3	2	1	0
–	–	–	NEGPPM	–	–	PERSIAN	HRMOD

This register can only be written if the WPEN bit is cleared in the System Controller Write Protection Mode Register (SYSC_WPMR).

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

0: 24-hour mode is selected.

1: 12-hour mode is selected.

- **PERSIAN: PERSIAN Calendar**

0: Gregorian calendar.

1: Persian calendar.

- **NEGPPM: NEGative PPM Correction**

0: Positive correction (the divider will be slightly higher than 32768).

1: Negative correction (the divider will be slightly lower than 32768).

Refer to CORRECTION and HIGHPPM field descriptions.

Note: NEGPPM must be cleared to correct a crystal slower than 32.768 kHz.

- **CORRECTION: Slow Clock Correction**

0: No correction

1–127: The slow clock will be corrected according to the formula given in HIGHPPM description.

- **HIGHPPM: HIGH PPM Correction**

0: Lower range ppm correction with accurate correction.

1: Higher range ppm correction with accurate correction.

If the absolute value of the correction to be applied is lower than 30 ppm, it is recommended to clear HIGHPPM. HIGHPPM set to 1 is recommended for 30 ppm correction and above.

Formula:

If HIGHPPM = 0, then the clock frequency correction range is from 1.5 ppm up to 98 ppm. The RTC accuracy is less than 1 ppm for a range correction from 1.5 ppm up to 30 ppm.

The correction field must be programmed according to the required correction in ppm; the formula is as follows:

$$CORRECTION = \frac{3906}{20 \times ppm} - 1$$

The value obtained must be rounded to the nearest integer prior to being programmed into CORRECTION field.

If HIGHPPM = 1, then the clock frequency correction range is from 30.5 ppm up to 1950 ppm. The RTC accuracy is less than 1 ppm for a range correction from 30.5 ppm up to 90 ppm.

The correction field must be programmed according to the required correction in ppm; the formula is as follows:

$$CORRECTION = \frac{3906}{ppm} - 1$$

The value obtained must be rounded to the nearest integer prior to be programmed into CORRECTION field.

If NEGPPM is set to 1, the ppm correction is negative (used to correct crystals that are faster than the nominal 32.768 kHz).

- **OUT0: RTCOUT0 OutputSource Selection**

Value	Name	Description
0	NO_WAVE	No waveform, stuck at ‘0’
1	FREQ1HZ	1 Hz square wave
2	FREQ32HZ	32 Hz square wave
3	FREQ64HZ	64 Hz square wave
4	FREQ512HZ	512 Hz square wave
5	ALARM_TOGGLE	Output toggles when alarm flag rises
6	ALARM_FLAG	Output is a copy of the alarm flag
7	PROG_PULSE	Duty cycle programmable pulse

- **OUT1: RTCOUT1 Output Source Selection**

Value	Name	Description
0	NO_WAVE	No waveform, stuck at ‘0’
1	FREQ1HZ	1 Hz square wave
2	FREQ32HZ	32 Hz square wave
3	FREQ64HZ	64 Hz square wave
4	FREQ512HZ	512 Hz square wave
5	ALARM_TOGGLE	Output toggles when alarm flag rises
6	ALARM_FLAG	Output is a copy of the alarm flag
7	PROG_PULSE	Duty cycle programmable pulse

- **THIGH: High Duration of the Output Pulse**

Value	Name	Description
0	H_31MS	31.2 ms
1	H_16MS	15.6 ms
2	H_4MS	3.91 ms
3	H_976US	976 µs

Value	Name	Description
4	H_488US	488 µs
5	H_122US	122 µs
6	H_30US	30.5 µs
7	H_15US	15.2 µs

- **TPERIOD: Period of the Output Pulse**

Value	Name	Description
0	P_1S	1 second
1	P_500MS	500 ms
2	P_250MS	250 ms
3	P_125MS	125 ms

15.6.3 RTC Time Register

Name: RTC_TIMR

Address: 0x400E1868

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	AMPM	HOUR					

15	14	13	12	11	10	9	8
–	MIN						

7	6	5	4	3	2	1	0
–	SEC						

- **SEC: Current Second**

The range that can be set is 0–59 (BCD).

The lowest four bits encode the units. The higher bits encode the tens.

- **MIN: Current Minute**

The range that can be set is 0–59 (BCD).

The lowest four bits encode the units. The higher bits encode the tens.

- **HOUR: Current Hour**

The range that can be set is 1–12 (BCD) in 12-hour mode or 0–23 (BCD) in 24-hour mode.

- **AMPM: Ante Meridiem Post Meridiem Indicator**

This bit is the AM/PM indicator in 12-hour mode.

0: AM.

1: PM.

15.6.4 RTC Calendar Register

Name: RTC_CALR

Address: 0x400E186C

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 (Gregorian) or 13–14 (Persian) (BCD).

The lowest four bits encode the units. The higher bits encode the tens.

- **YEAR: Current Year**

The range that can be set is 00–99 (BCD).

The lowest four bits encode the units. The higher bits encode the tens.

- **MONTH: Current Month**

The range that can be set is 01–12 (BCD).

The lowest four bits encode the units. The higher bits encode the tens.

- **DAY: Current Day in Current Week**

The range that can be set is 1–7 (BCD).

The coding of the number (which number represents which day) is user-defined as it has no effect on the date counter.

- **DATE: Current Day in Current Month**

The range that can be set is 01–31 (BCD).

The lowest four bits encode the units. The higher bits encode the tens.

15.6.5 RTC Time Alarm Register

Name: RTC_TIMALR

Address: 0x400E1870

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
HOUREN	AMPM	HOUR					
15	14	13	12	11	10	9	8
MINEN	MIN						
7	6	5	4	3	2	1	0
SECEN	SEC						

This register can only be written if the WPEN bit is cleared in the System Controller Write Protection Mode Register (SYSC_WPMR).

Note: To change one of the SEC, MIN, HOUR fields, it is recommended to disable the field before changing the value and then re-enable it after the change has been made. This requires up to three accesses to the RTC_TIMALR. The first access clears the enable corresponding to the field to change (SECEN, MINEN, HOUREN). If the field is already cleared, this access is not required. The second access performs the change of the value (SEC, MIN, HOUR). The third access is required to re-enable the field by writing 1 in SECEN, MINEN, HOUREN fields.

- **SEC: Second Alarm**

This field is the alarm field corresponding to the BCD-coded second counter.

- **SECEN: Second Alarm Enable**

0: The second-matching alarm is disabled.

1: The second-matching alarm is enabled.

- **MIN: Minute Alarm**

This field is the alarm field corresponding to the BCD-coded minute counter.

- **MINEN: Minute Alarm Enable**

0: The minute-matching alarm is disabled.

1: The minute-matching alarm is enabled.

- **HOUR: Hour Alarm**

This field is the alarm field corresponding to the BCD-coded hour counter.

- **AMPM: AM/PM Indicator**

This field is the alarm field corresponding to the BCD-coded hour counter.

- **HOUREN: Hour Alarm Enable**

0: The hour-matching alarm is disabled.

1: The hour-matching alarm is enabled.

15.6.6 RTC Calendar Alarm Register

Name: RTC_CALALR

Address: 0x400E1874

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the System Controller Write Protection Mode Register (SYSC_WPMR).

Note: To change one of the DATE, MONTH fields, it is recommended to disable the field before changing the value and then re-enable it after the change has been made. This requires up to three accesses to the RTC_CALALR. The first access clears the enable corresponding to the field to change (DATEEN, MTHEN). If the field is already cleared, this access is not required. The second access performs the change of the value (DATE, MONTH). The third access is required to re-enable the field by writing 1 in DATEEN, MTHEN fields.

- **MONTH: Month Alarm**

This field is the alarm field corresponding to the BCD-coded month counter.

- **MTHEN: Month Alarm Enable**

0: The month-matching alarm is disabled.

1: The month-matching alarm is enabled.

- **DATE: Date Alarm**

This field is the alarm field corresponding to the BCD-coded date counter.

- **DATEEN: Date Alarm Enable**

0: The date-matching alarm is disabled.

1: The date-matching alarm is enabled.

15.6.7 RTC Status Register

Name: RTC_SR

Address: 0x400E1878

Access: Read-only

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

- **ACKUPD: Acknowledge for Update**

Value	Name	Description
0	FREERUN	Time and calendar registers cannot be updated.
1	UPDATE	Time and calendar registers can be updated.

- **ALARM: Alarm Flag**

Value	Name	Description
0	NO_ALARMEVENT	No alarm matching condition occurred.
1	ALARMEVENT	An alarm matching condition has occurred.

- **SEC: Second Event**

Value	Name	Description
0	NO_SECEVENT	No second event has occurred since the last clear.
1	SECEVENT	At least one second event has occurred since the last clear.

- **TIMEV: Time Event**

Value	Name	Description
0	NO_TIMEVENT	No time event has occurred since the last clear.
1	TIMEVENT	At least one time event has occurred since the last clear.

Note: The time event is selected in the TIMEVSEL field in the Control Register (RTC_CR) and can be any one of the following events: minute change, hour change, noon, midnight (day change).

- **CALEV: Calendar Event**

Value	Name	Description
0	NO_CALEVENT	No calendar event has occurred since the last clear.
1	CALEVENT	At least one calendar event has occurred since the last clear.

Note: The calendar event is selected in the CALEVSEL field in the Control Register (RTC_CR) and can be any one of the following events: week change, month change and year change.

- **TDERR: Time and/or Date Free Running Error**

Value	Name	Description
0	CORRECT	The internal free running counters are carrying valid values since the last read of the Status Register (RTC_SR).
1	ERR_TIMEDATE	The internal free running counters have been corrupted (invalid date or time, non-BCD values) since the last read and/or they are still invalid.

15.6.8 RTC Status Clear Command Register

Name: RTC_SCCR

Address: 0x400E187C

Access: Write-only

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

- **ACKCLR: Acknowledge Clear**

0: No effect.

1: Clears corresponding status flag in the Status Register (RTC_SR).

- **ALRCLR: Alarm Clear**

0: No effect.

1: Clears corresponding status flag in the Status Register (RTC_SR).

- **SECCLR: Second Clear**

0: No effect.

1: Clears corresponding status flag in the Status Register (RTC_SR).

- **TIMCLR: Time Clear**

0: No effect.

1: Clears corresponding status flag in the Status Register (RTC_SR).

- **CALCLR: Calendar Clear**

0: No effect.

1: Clears corresponding status flag in the Status Register (RTC_SR).

- **TDERRCLR: Time and/or Date Free Running Error Clear**

0: No effect.

1: Clears corresponding status flag in the Status Register (RTC_SR).

15.6.9 RTC Interrupt Enable Register

Name: RTC_IER

Address: 0x400E1880

Access: Write-only

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

- **ACKEN: Acknowledge Update Interrupt Enable**

0: No effect.

1: The acknowledge for update interrupt is enabled.

- **ALREN: Alarm Interrupt Enable**

0: No effect.

1: The alarm interrupt is enabled.

- **SECEN: Second Event Interrupt Enable**

0: No effect.

1: The second periodic interrupt is enabled.

- **TIMEN: Time Event Interrupt Enable**

0: No effect.

1: The selected time event interrupt is enabled.

- **CALEN: Calendar Event Interrupt Enable**

0: No effect.

1: The selected calendar event interrupt is enabled.

- **TDERREN: Time and/or Date Error Interrupt Enable**

0: No effect.

1: The time and date error interrupt is enabled.

15.6.10 RTC Interrupt Disable Register

Name: RTC_IDR

Address: 0x400E1884

Access: Write-only

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

- **ACKDIS: Acknowledge Update Interrupt Disable**

0: No effect.

1: The acknowledge for update interrupt is disabled.

- **ALRDIS: Alarm Interrupt Disable**

0: No effect.

1: The alarm interrupt is disabled.

- **SECDIS: Second Event Interrupt Disable**

0: No effect.

1: The second periodic interrupt is disabled.

- **TIMDIS: Time Event Interrupt Disable**

0: No effect.

1: The selected time event interrupt is disabled.

- **CALDIS: Calendar Event Interrupt Disable**

0: No effect.

1: The selected calendar event interrupt is disabled.

- **TDERRDIS: Time and/or Date Error Interrupt Disable**

0: No effect.

1: The time and date error interrupt is disabled.

15.6.11 RTC Interrupt Mask Register

Name: RTC_IMR

Address: 0x400E1888

Access: Read-only

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

- **TDERR: Time and/or Date Error Mask**

0: The time and/or date error event is disabled.

1: The time and/or date error event is enabled.

15.6.12 RTC Valid Entry Register

Name: RTC_VER

Address: 0x400E188C

Access: Read-only

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

- **NVTIM: Non-valid Time**

0: No invalid data has been detected in RTC_TIMR (Time Register).

1: RTC_TIMR has contained invalid data since it was last programmed.

- **NVCAL: Non-valid Calendar**

0: No invalid data has been detected in RTC_CALR (Calendar Register).

1: RTC_CALR has contained invalid data since it was last programmed.

- **NVTIMALR: Non-valid Time Alarm**

0: No invalid data has been detected in RTC_TIMALR (Time Alarm Register).

1: RTC_TIMALR has contained invalid data since it was last programmed.

- **NVCALALR: Non-valid Calendar Alarm**

0: No invalid data has been detected in RTC_CALALR (Calendar Alarm Register).

1: RTC_CALALR has contained invalid data since it was last programmed.

15.6.13 RTC TimeStamp Time Register 0 (UTC_MODE)

Name: RTC_TSTR0 (UTC_MODE)

Access: Read-only

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

RTC_TSTR0 reports the timestamp of the first tamper event.

- **TEVCNT: Tamper Events Counter (cleared by reading RTC_TSSR0)**

Each time a tamper event occurs, this counter is incremented. This counter saturates at 15. Once this value is reached, it is no more possible to know the exact number of tamper events.

If this field is not null, this implies that at least one tamper event occurs since last register reset and that the values stored in timestamping registers are valid.

- **BACKUP: System Mode of the Tamper (cleared by reading RTC_TSSR0)**

0: The state of the system is different from Backup mode when the tamper event occurs.

1: The system is in Backup mode when the tamper event occurs.

15.6.14 RTC TimeStamp Time Register 1 (UTC_MODE)

Name: RTC_TSTR1 (UTC_MODE)

Access: Read-only

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

RTC_TSTR1 reports the timestamp of the last tamper event.

- **BACKUP: System Mode of the Tamper**

0: The state of the system is different from Backup mode when the tamper event occurs.

1: The system is in Backup mode when the tamper event occurs.

15.6.15 RTC TimeStamp Date Register (UTC_MODE)

Name: RTC_TSDRx (UTC_MODE)

Access: Read-only

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

- **UTC_TIME: Time of the Tamper (UTC format)**

This configuration is relevant only if UTC = 1 in RTC_MR.

16. Watchdog Timer (WDT)

16.1 Description

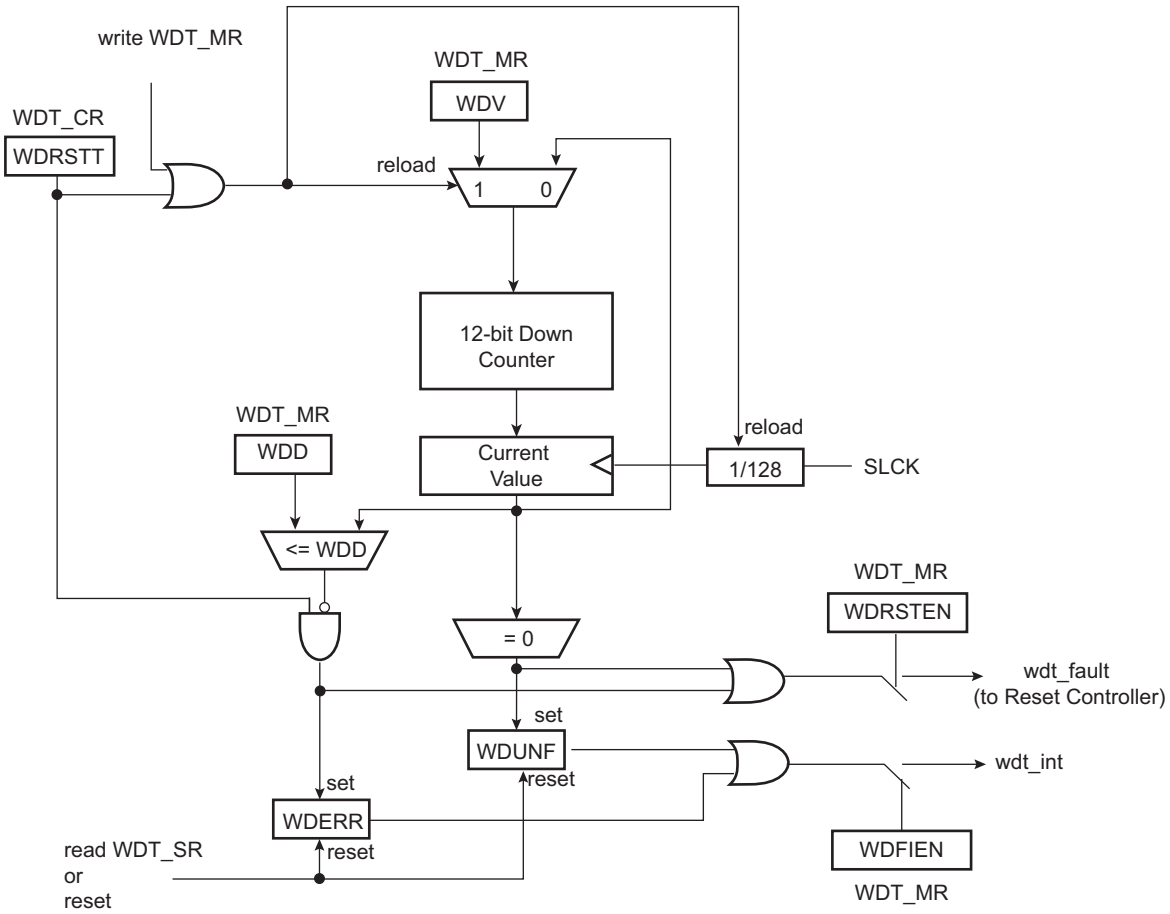
The Watchdog Timer (WDT) is used to prevent system lock-up if the software becomes trapped in a deadlock. It features a 12-bit down counter that allows a watchdog period of up to 16 seconds (slow clock around 32 kHz). It can generate a general reset or a processor reset only. In addition, it can be stopped while the processor is in Debug mode or Sleep mode (Idle mode).

16.2 Embedded Characteristics

- 12-bit Key-protected Programmable Counter
- Watchdog Clock is Independent from Processor Clock
- Provides Reset or Interrupt Signals to the System
- Counter May Be Stopped while the Processor is in Debug State or in Idle Mode

16.3 Block Diagram

Figure 16-1. Watchdog Timer Block Diagram



16.4 Functional Description

The Watchdog Timer is 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 can either disable the WDT by setting bit WDT_MR.WDDIS or reprogram the WDT to meet the maximum watchdog period the application requires.

When setting the WDDIS bit, and while it is set, the fields WDV and WDD must not be modified.

If the watchdog is restarted by writing into the Control Register (WDT_CR), WDT_MR must not be programmed during a period of time of three slow clock periods following the WDT_CR write access. In any case, programming a new value in WDT_MR automatically initiates a restart instruction.

WDT_MR can be written only once. Only a processor reset resets it. Writing WDT_MR 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 setting bit WDT_CR.WDRSTT. The watchdog counter is then immediately reloaded from WDT_MR and restarted, and the slow clock 128 divider is reset and restarted. WDT_CR 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 bit WDT_MR.WDRSTEN is set. Moreover, the bit WDUNF is set in the Status Register (WDT_SR).

The reload of the watchdog must occur while the watchdog counter is within a window between 0 and WDD. WDD is defined in 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 WDT_SR.WDERR is updated 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).

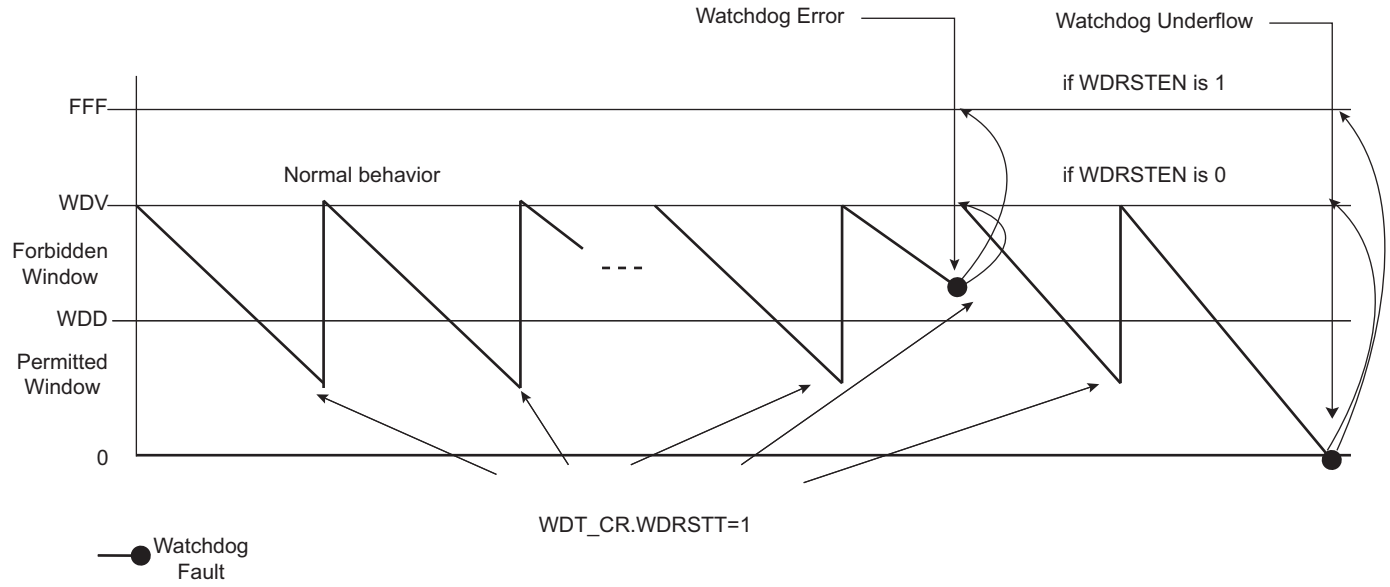
The status bits WDUNF (Watchdog Underflow) and WDERR (Watchdog Error) trigger an interrupt, provided the bit WDT_MR.WDFIEN is set. The signal “wdt_fault” to the Reset Controller causes a watchdog reset if the WDRSTEN bit is set as already explained in the Reset Controller documentation. In this case, 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 WDT_MR reloads and restarts the down counter.

While the processor is in debug state or in Sleep mode, the counter may be stopped depending on the value programmed for the bits WDIDLEHLT and WDDBGHLT in WDT_MR.

Figure 16-2. Watchdog Behavior



16.5 Watchdog Timer (WDT) User Interface

Table 16-1. 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

16.5.1 Watchdog Timer Control Register

Name: WDT_CR

Address: 0x400E1850

Access: Write-only

31	30	29	28	27	26	25	24
KEY							

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WDRSTT

Note: The WDT_CR register values must not be modified within three slow clock periods following a restart of the watchdog performed by a write access in WDT_CR. Any modification will cause the watchdog to trigger an end of period earlier than expected.

- **WDRSTT: Watchdog Restart**

0: No effect.

1: Restarts the watchdog if KEY is written to 0xA5.

- **KEY: Password**

Value	Name	Description
0xA5	PASSWD	Writing any other value in this field aborts the write operation.

16.5.2 Watchdog Timer Mode Register

Name: WDT_MR

Address: 0x400E1854

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							

Notes:
1. The first write access prevents any further modification of the value of this register. Read accesses remain possible.
2. The WDT_MR register values must not be modified within three slow clock periods following a restart of the watchdog performed by a write access in WDT_CR. Any modification will cause the watchdog to trigger an end of period earlier than expected.

- **WDV: Watchdog Counter Value**

Defines the value loaded in the 12-bit watchdog counter.

- **WDFIEN: Watchdog Fault Interrupt Enable**

0: A watchdog fault (underflow or error) has no effect on interrupt.

1: A watchdog fault (underflow or error) asserts interrupt.

- **WDRSTEN: Watchdog Reset Enable**

0: A watchdog fault (underflow or error) has no effect on the resets.

1: A watchdog fault (underflow or error) triggers a watchdog reset.

- **WDRPROC: Watchdog Reset Processor**

0: If WDRSTEN is 1, a watchdog fault (underflow or error) activates all resets.

1: If WDRSTEN is 1, a watchdog fault (underflow or error) activates the processor reset.

- **WDDIS: Watchdog Disable**

0: Enables the Watchdog Timer.

1: Disables the Watchdog Timer.

Note: When setting the WDDIS bit, and while it is set, the fields WDV and WDD must not be modified.

- **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, setting bit WDT_CR.WDRSTT restarts the timer.

If the Watchdog Timer value is greater than WDD, setting bit WDT_CR.WDRSTT 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 state.

1: The watchdog stops when the system is in idle state.

16.5.3 Watchdog Timer Status Register

Name: WDT_SR

Address: 0x400E1858

Access Read-only

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

- **WDUNF: Watchdog Underflow (cleared on read)**

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 (cleared on read)**

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.

17. Reinforced Safety Watchdog Timer (RSWDT)

17.1 Description

The Reinforced Safety Watchdog Timer (RSWDT) works in parallel with the Watchdog Timer (WDT) to reinforce safe watchdog operations.

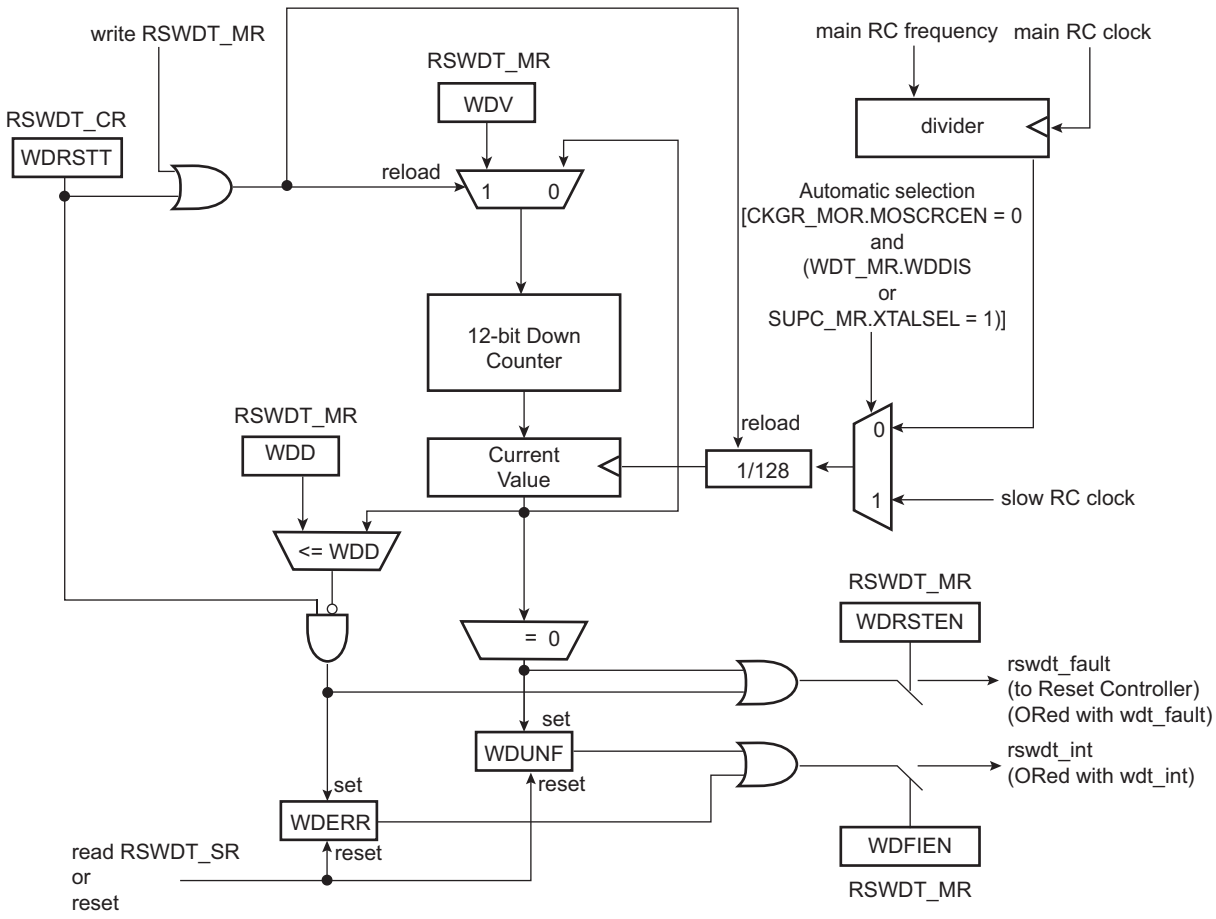
The RSWDT can be used to reinforce the safety level provided by the WDT in order to prevent system lock-up if the software becomes trapped in a deadlock. The RSWDT works in a fully operable mode, independent of the WDT. Its clock source is automatically selected from either the slow RC oscillator clock or main RC oscillator divided clock to get an equivalent slow RC oscillator clock. If the WDT clock source (for example, the 32 kHz crystal oscillator) fails, the system lock-up is no longer monitored by the WDT because the RSWDT performs the monitoring. Thus, there is no lack of safety irrespective of the external operating conditions. The RSWDT shares the same features as the WDT (i.e., a 12-bit down counter that allows a watchdog period of up to 16 seconds with slow clock at 32.768 kHz). It can generate a general reset or a processor reset only. In addition, it can be stopped while the processor is in debug mode or idle mode.

17.2 Embedded Characteristics

- Automatically Selected Reliable RSWDT Clock Source (independent of WDT clock source)
- Windowed Watchdog
- 12-bit Key-protected Programmable Counter
- Provides Reset or Interrupt Signals to the System
- Counter may be Stopped While Processor is in Debug State or Idle Mode

17.3 Block Diagram

Figure 17-1. Reinforced Safety Watchdog Timer Block Diagram



17.4 Functional Description

The RSWDT is supplied by VDDCORE. The RSWDT is initialized with default values on processor reset or on a power-on sequence and is disabled (its default mode) under such conditions.

The RSWDT must not be enabled if the WDT is disabled.

The main RC oscillator divided clock is selected if the main RC oscillator is already enabled by the application (CKGR_MOR.MOSCRCEN = 1) or if the WDT is driven by the slow RC oscillator.

The RSWDT is built around a 12-bit down counter, which is loaded with a slow clock value other than that of the slow clock in the WDT, defined in the WDV (Watchdog Counter Value) field of the Mode Register (RSWDT_MR). The RSWDT 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 (RSWDT_MR.WDRSTEN = 1 after a backup reset). This means that a default watchdog is running at reset, i.e., at power-up.

If the watchdog is restarted by writing into the Control Register (RSWDT_CR), the RSWDT_MR must not be programmed during a period of time of three slow clock periods following the RSWDT_CR write access. Programming a new value in the RSWDT_MR automatically initiates a restart instruction.

RSWDT_MR can be written only once. Only a processor reset resets it. Writing RSWDT_MR 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 setting bit RSWDT_CR.WDRSTT. The watchdog counter is then immediately reloaded from the RSWDT_MR and restarted, and the slow clock 128 divider is reset and restarted. The RSWDT_CR is write-protected. As a result, writing RSWDT_CR without the correct hard-coded key has no effect. If an underflow does occur, the “wdt_fault” signal to the reset controller is asserted if the bit RSWDT_MR.WDRSTEN is set. Moreover, the bit WDUNF (Watchdog Underflow) is set in the Status Register (RSWDT_SR).

To prevent a software deadlock that continuously triggers the RSWDT, the reload of the RSWDT must occur while the watchdog counter is within a window between 0 and the Watchdog Delta Value (WDD). WDD is defined in the RSWDT_MR.

Any attempt to restart the watchdog while the watchdog counter is between the two values WDV and WDD results in a watchdog error, even if the RSWDT is disabled. The WDERR (Watchdog Error) bit is updated in the RSWDT_SR and the “wdt_fault” signal to the reset controller is asserted.

Note that the Windowed Watchdog feature can be disabled by programming a WDD value greater than or equal to the WDV value. In such a configuration, restarting the RSWDT is permitted in the whole range 0 to 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 and WDERR trigger an interrupt, provided the WDFIEN bit is set in the RSWDT_MR. The signal “wdt_fault” to the reset controller causes a Watchdog reset if the WDRSTEN bit is set as explained in the “Reset Controller (RSTC)” section of the product datasheet. In this case, the processor and the RSWDT are reset, and the WDUNF and WDERR flags are reset.

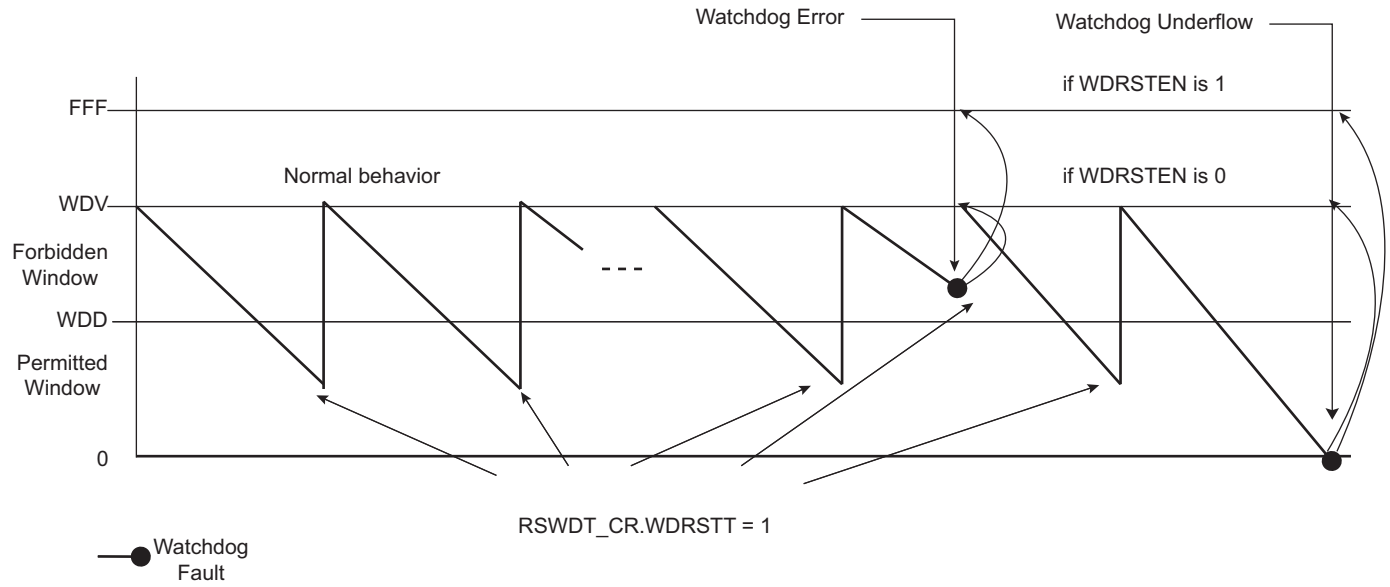
If a reset is generated, or if RSWDT_SR is read, the status bits are reset, the interrupt is cleared, and the “wdt_fault” signal to the reset controller is deasserted

Writing RSWDT_MR reloads and restarts the down counter.

The RSWDT is disabled after any power-on sequence.

While the processor is in debug state or in idle mode, the counter may be stopped depending on the value programmed for the WDIDLEHLT and WDDBGHLT bits in the RSWDT_MR.

Figure 17-2. Watchdog Behavior



17.5 Reinforced Safety Watchdog Timer (RSWDT) User Interface

Table 17-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	RSWDT_CR	Write-only	–
0x04	Mode Register	RSWDT_MR	Read-write Once	0x3FFF_AFFF
0x08	Status Register	RSWDT_SR	Read-only	0x0000_0000

17.5.1 Reinforced Safety Watchdog Timer Control Register

Name: RSWDT_CR

Address: 0x400E1900

Access: Write-only

31	30	29	28	27	26	25	24
KEY							

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WDRSTT

- **WDRSTT: Watchdog Restart**

0: No effect.

1: Restarts the watchdog.

- **KEY: Password**

Value	Name	Description
0xC4	PASSWD	Writing any other value in this field aborts the write operation.

17.5.2 Reinforced Safety Watchdog Timer Mode Register

Name: RSWDT_MR

Address: 0x400E1904

Access: Read-write Once

31	30	29	28	27	26	25	24
–	–	WDIDLEHLT	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							

Note: The first write access prevents any further modification of the value of this register; read accesses remain possible.

Note: The WDD and WDV values must not be modified within three slow clock periods following a restart of the watchdog performed by means of a write access in the RSWDT_CR, else the watchdog may trigger an end of period earlier than expected.

- **WDV: Watchdog Counter Value**

Defines the value loaded in the 12-bit watchdog counter.

- **WDFIEN: Watchdog Fault Interrupt Enable**

0: A Watchdog fault (underflow or error) has no effect on interrupt.

1: A Watchdog fault (underflow or error) asserts interrupt.

- **WDRSTEN: Watchdog Reset Enable**

0: A Watchdog fault (underflow or error) has no effect on the resets.

1: A Watchdog fault (underflow or error) triggers a watchdog reset.

- **WDRPROC: Watchdog Reset Processor**

0: If WDRSTEN is 1, a watchdog fault (underflow or error) activates all resets.

1: If WDRSTEN is 1, a watchdog fault (underflow or error) activates the processor reset.

- **WDD: Watchdog Delta Value**

Defines the permitted range for reloading the RSWDT.

If the RSWDT value is less than or equal to WDD, writing RSWDT_CR with WDRSTT = 1 restarts the timer.

If the RSWDT value is greater than WDD, writing RSWDT_CR with WDRSTT = 1 causes a Watchdog error.

- **WDDBGHLT: Watchdog Debug Halt**

0: The RSWDT runs when the processor is in debug state.

1: The RSWDT stops when the processor is in debug state.

- **WDIDLEHLT: Watchdog Idle Halt**

0: The RSWDT runs when the system is in idle mode.

1: The RSWDT stops when the system is in idle state.

- **WDDIS: Watchdog Disable**

0: Enables the RSWDT.

1: Disables the RSWDT.

17.5.3 Reinforced Safety Watchdog Timer Status Register

Name: RSWDT_SR

Address: 0x400E1908

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	WDERR	WDUNF

- **WDUNF: Watchdog Underflow**

0: No watchdog underflow occurred since the last read of RSWDT_SR.

1: At least one watchdog underflow occurred since the last read of RSWDT_SR.

- **WDERR: Watchdog Error**

0: No watchdog error occurred since the last read of RSWDT_SR.

1: At least one watchdog error occurred since the last read of RSWDT_SR.

18. Supply Controller (SUPC)

18.1 Description

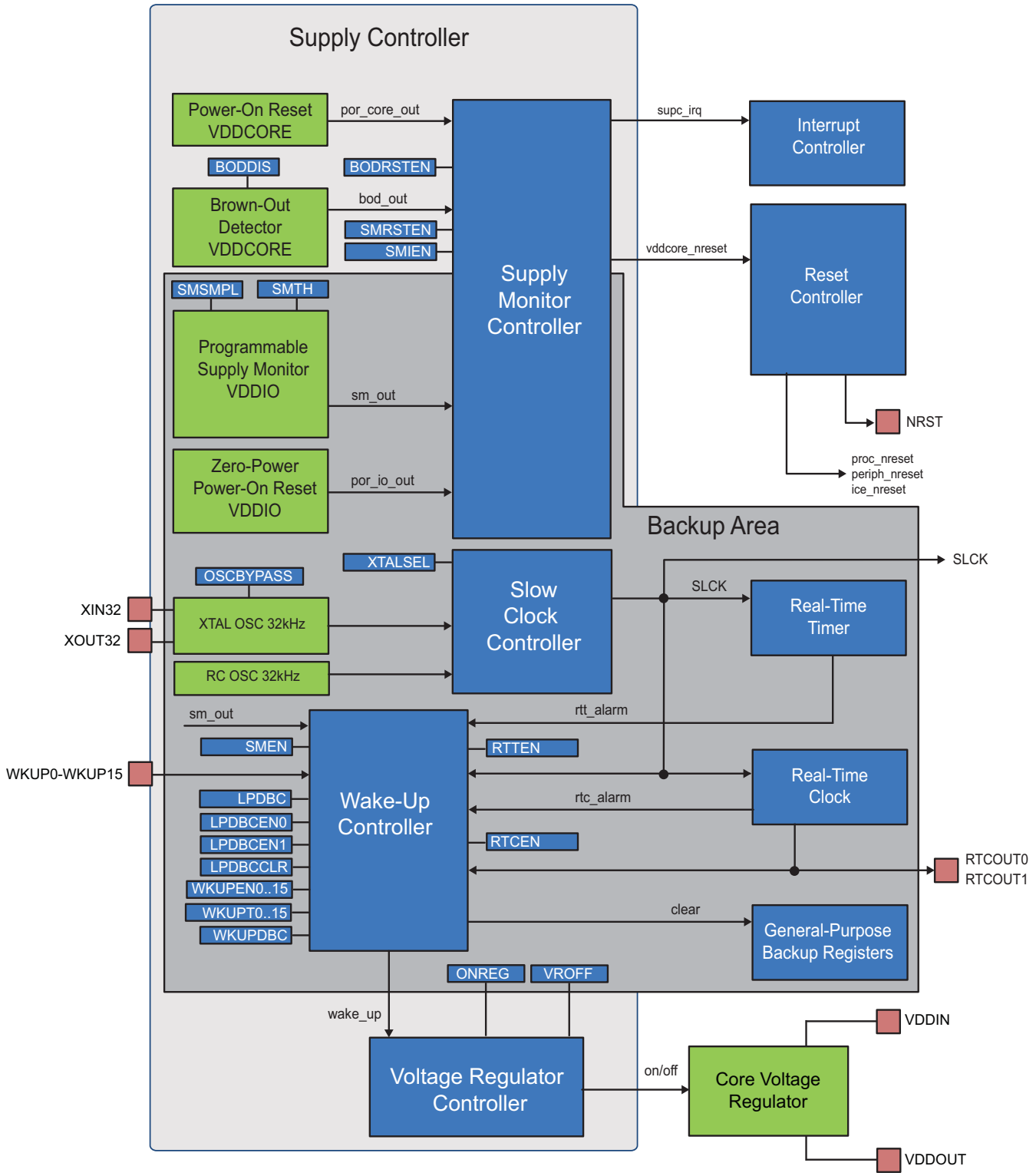
The Supply Controller (SUPC) controls the supply voltages of the system and manages the Backup mode. In this mode, current consumption is reduced to a few microamps for backup power retention. Exit from this mode is possible on multiple wake-up sources. The SUPC also generates the slow clock by selecting either the low-power RC oscillator or the low-power crystal oscillator.

18.2 Embedded Characteristics

- Manages the core power supply VDDCORE and backup mode by controlling the embedded voltage regulator
- A supply monitor detection on VDDIO or a brownout detection on VDDCORE triggers a core reset
- Generates the slow clock SLCK by selecting either the 22-42 kHz low-power RC oscillator or the 32 kHz low-power crystal oscillator
- Low-power tamper detection on two inputs
- Anti-tampering by immediate clear of the general-purpose backup registers
- Supports multiple wake-up sources for exit from backup 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 VDDIO, with programmable scan period and voltage threshold

18.3 Block Diagram

Figure 18-1. Supply Controller Block Diagram



18.4 Functional Description

18.4.1 Overview

The device is divided into two power supply areas:

- VDDIO power supply: includes the Supply Controller, 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.
- Core power supply: includes part of the Reset Controller, the Brownout Detector, the processor, the SRAM memory, the Flash memory and the peripherals.

The Supply Controller (SUPC) controls the supply voltage of the core power supply. The SUPC intervenes when the VDDIO power supply rises (when the system is starting) or when Backup 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 SUPC and the VDDIO power supply have a reset circuitry based on a zero-power power-on reset cell. The zero-power power-on reset allows the SUPC to start correctly as soon as the VDDIO voltage becomes valid.

At start-up of the system, once the backup voltage VDDIO is valid and the embedded 32 kHz RC oscillator is stabilized, the SUPC starts up the core by sequentially enabling the internal voltage regulator. The SUPC waits until the core voltage VDDCORE is valid, then releases 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 level on VDDIO that is too low, the SUPC asserts the reset signal of the core vddcore_nreset signal until VDDIO is valid. Likewise, if the brownout detector detects a core voltage level VDDCORE that is too low, the SUPC asserts the reset signal vddcore_nreset until VDDCORE is valid.

When Backup mode is entered, the SUPC sequentially asserts the reset signal of the core power supply vddcore_nreset and disables the voltage regulator, in order to supply only the VDDIO power supply. Current consumption is reduced to a few microamps for the backup part retention. Exit from this mode is possible on multiple wake-up sources including an event on the FWUP pin or WKUP pins, or a clock alarm. To exit this mode, the SUPC operates in the same way as system start-up.

18.4.2 Slow Clock Generator

The SUPC embeds a slow clock generator that is supplied with the VDDIO power supply. As soon as the VDDIO is supplied, both the crystal oscillator and the embedded RC oscillator are powered up, but only the embedded RC oscillator is enabled. When the RC oscillator is selected as the slow clock source, the slow clock stabilizes more quickly than when the crystal oscillator is selected.

The user can select the crystal oscillator to be the source of the slow clock, as it provides a more accurate frequency than the RC oscillator. The crystal oscillator is selected by setting the XTALSEL bit in the SUPC Control register (SUPC_CR). The following sequence must be used to switch from the RC oscillator to the crystal oscillator:

9. The PIO lines multiplexed with XIN32 and XOUT32 are configured to be driven by the oscillator.
10. The crystal oscillator is enabled.
11. A number of RC oscillator clock periods is counted to cover the start-up time of the crystal oscillator. Refer to the section “Electrical Characteristics” for information on 32 kHz crystal oscillator start-up time.
12. The slow clock is switched to the output of the crystal oscillator.
13. The RC oscillator is disabled to save power.

The switching time may vary depending on the RC oscillator clock frequency range. The switch of the slow clock source is glitch-free. The OSCSEL bit of the SUPC Status register (SUPC_SR) indicates when the switch sequence is finished.

Reverting to the RC oscillator as a slow clock source is only possible by shutting down the VDDIO power supply.

If the user does not need the crystal oscillator, the XIN32 and XOUT32 pins should be left unconnected.

The user can also set the crystal oscillator in Bypass mode instead of connecting a crystal. In this case, the user has to provide the external clock signal on XIN32. The input characteristics of the XIN32 pin are given in the section ‘Electrical Characteristics. To enter Bypass mode, the OSCBYPASS bit in the Mode register (SUPC_MR) must be set before setting XTALSEL.

18.4.3 Core Voltage Regulator Control/Backup Low-power Mode

The SUPC can be used to control the embedded voltage regulator.

The voltage regulator automatically adapts its quiescent current depending on the required load current. Refer to the section “Electrical Characteristics”.

The user can switch off the voltage regulator, and thus put the device in Backup mode, by writing a 1 to the VROFF bit in SUPC_CR.

This asserts the vddcore_nreset signal after the write resynchronization time, which lasts two slow clock cycles (worst case). 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 internal voltage regulator is not used and VDDCORE is supplied by an external supply, the voltage regulator can be disabled by writing a 1 to the ONREG bit in SUPC_MR.

18.4.4 Supply Monitor

The SUPC embeds a supply monitor located in the VDDIO power supply and which monitors VDDIO power supply.

The supply monitor can be used to prevent the processor from falling into an unpredictable state if the main power supply drops below a certain level.

The threshold of the supply monitor is programmable in the SMTH field of the Supply Monitor Mode register (SUPC_SMMR). Refer to Supply Monitor characteristics in the section “Electrical Characteristics”.

The supply monitor can also be enabled during one slow clock period on every one of either 32, 256 or 2048 slow clock periods, depending on the user selection. This is configured in the SMSMPL field in SUPC_SMMR.

Enabling the supply monitor for such reduced times divides the typical supply monitor power consumption by factors of 2, 16 and 128, respectively, if continuous monitoring of the VDDIO power supply is not required.

A supply monitor detection generates either 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 setting the SMRSTEN bit in SUPC_SMMR.

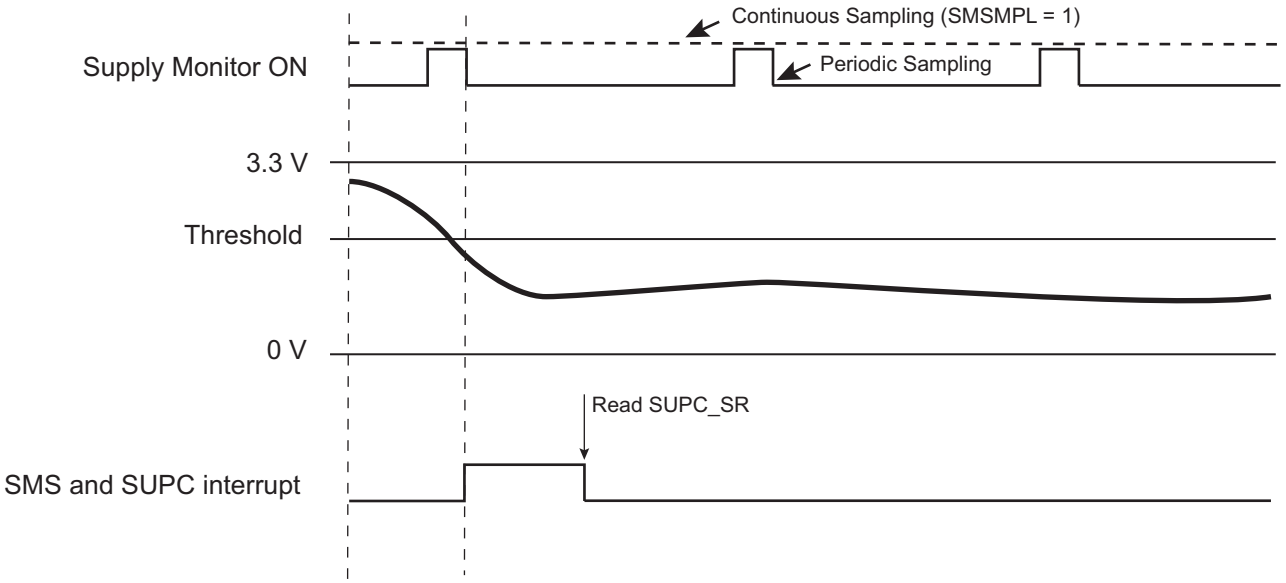
Waking up the core power supply when a supply monitor detection occurs can be enabled by setting the SMEN bit in the Wake-up Mode register (SUPC_WUMR).

The SUPC provides two status bits in the SUPC_SR for the supply monitor that determine whether the last wake-up was due to the supply monitor:

- The SMOS bit provides real-time information, 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 flag generates an interrupt if the SMIEN bit is set in SUPC_SMMR.

Figure 18-2. Supply Monitor Status Bit and Associated Interrupt



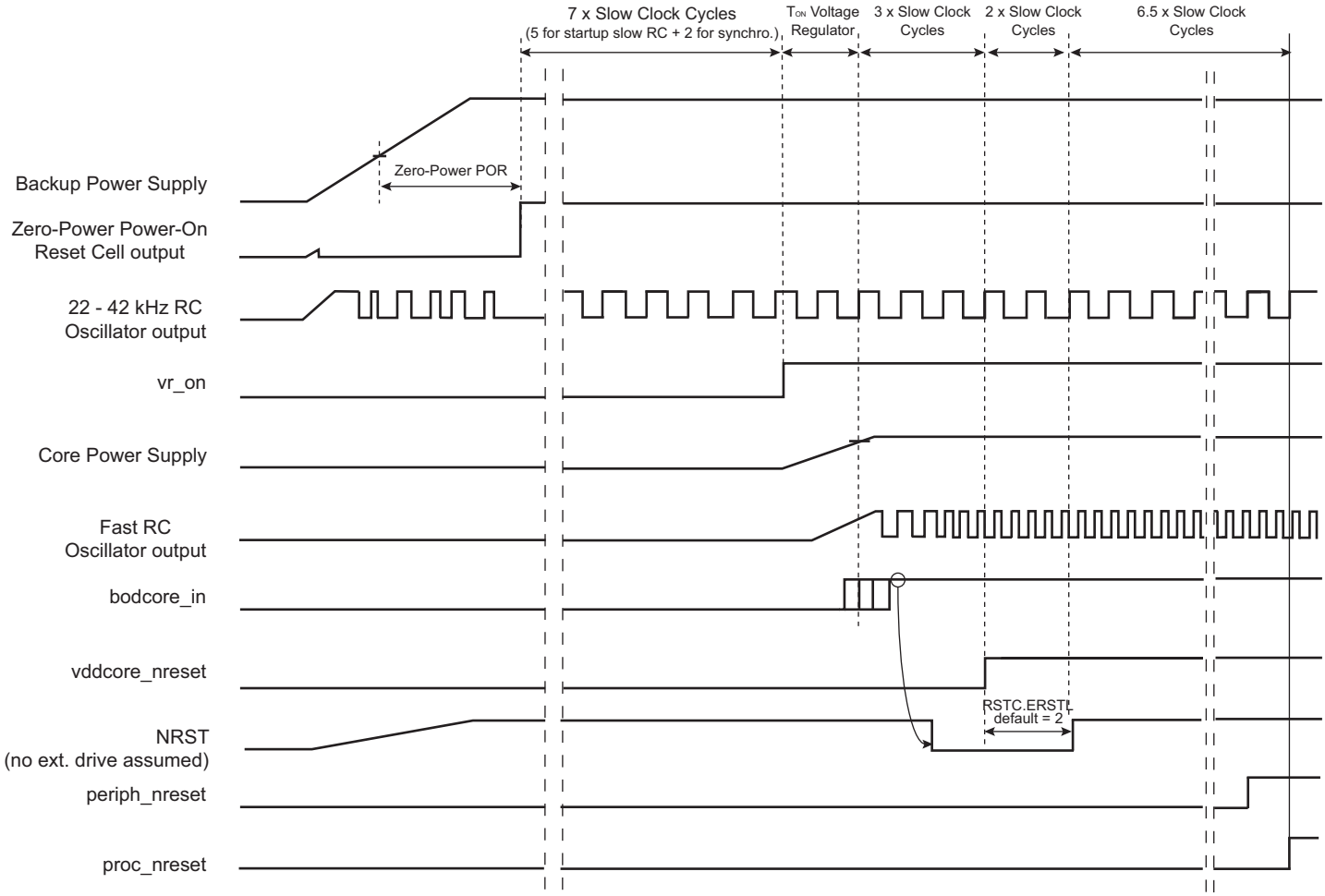
18.4.5 Backup Power Supply Reset

18.4.5.1 Raising the Backup Power Supply

When the backup voltage VDDIO rises, the RC oscillator is powered up and the zero-power power-on reset cell maintains its output low as long as VDDIO has not reached its target voltage. During this period, the SUPC is reset. When the VDDIO voltage becomes valid and the zero-power power-on reset signal is released, a counter is started for five slow clock cycles. This is the time required for the 32 kHz RC oscillator to stabilize.

After this time, the voltage regulator is enabled. The core power supply rises and the brownout detector provides the bodcore_in signal as soon as the core voltage VDDCORE is valid. This results in releasing the vddcore_nreset signal to the Reset Controller after the bodcore_in signal has been confirmed as being valid for at least one slow clock cycle.

Figure 18-3. Raising the VDDIO Power Supply



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

18.4.6 Core Reset

The Supply Controller manages the vddcore_nreset signal to the Reset Controller, as described in Section 18.4.5 ”Backup Power Supply Reset”. The vddcore_nreset signal is normally asserted before shutting down the core power supply and released as soon as the core power supply is correctly regulated.

There are two additional sources which can be programmed to activate vddcore_nreset:

- a supply monitor detection
- a brownout detection

18.4.6.1 Supply Monitor Reset

The supply monitor is capable of generating a reset of the system. This is enabled by setting the SMRSTEN bit in SUPC_SMMR.

If SMRSTEN is set and if a supply monitor detection occurs, the vddcore_nreset signal is immediately activated for a minimum of one slow clock cycle.

18.4.6.2 Brownout Detector Reset

The brownout detector provides the bodcore_in signal to the SUPC. This signal 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 SUPC asserts vddcore_nreset if BODRSTEN is written to 1 in SUPC_MR.

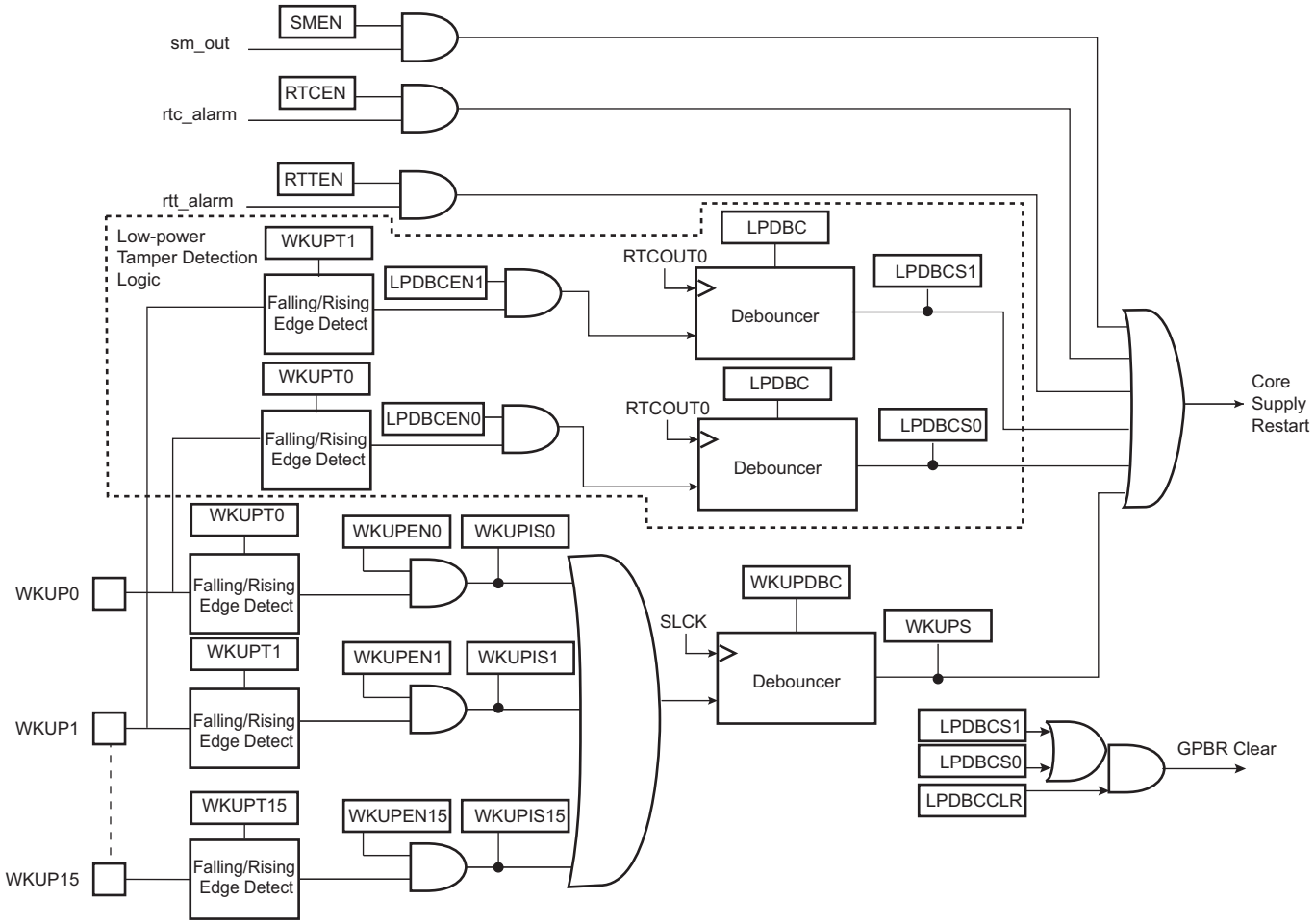
If BODRSTEN is set and the voltage regulation is lost (output voltage of the regulator too low), the vddcore_nreset signal is asserted for a minimum of one slow clock cycle and then released if bodcore_in has been reactivated. The BODRSTS bit in SUPC_SR indicates the source of the last reset.

Until bodcore_in is deactivated, the vddcore_nreset signal remains active.

18.4.7 Wake-up Sources

The wake-up events allow the device to exit Backup mode. When a wake-up event is detected, the SUPC performs a sequence that automatically reenables the core power supply.

Figure 18-4. Wake-up Sources



18.4.7.1 Force Wake-up

The FWUP pin is enabled as a wake-up source by writing a 1 to the FWUPEN bit in SUPC_WUMR. The FWUPDBC field then selects a debouncing period of 3, 32, 512, 4,096 or 32,768 slow clock cycles. The duration of these periods 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 at least 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 FWUPS bit in SUPC_SR is set and remains high until the register is read.

18.4.7.2 Wake-up Inputs

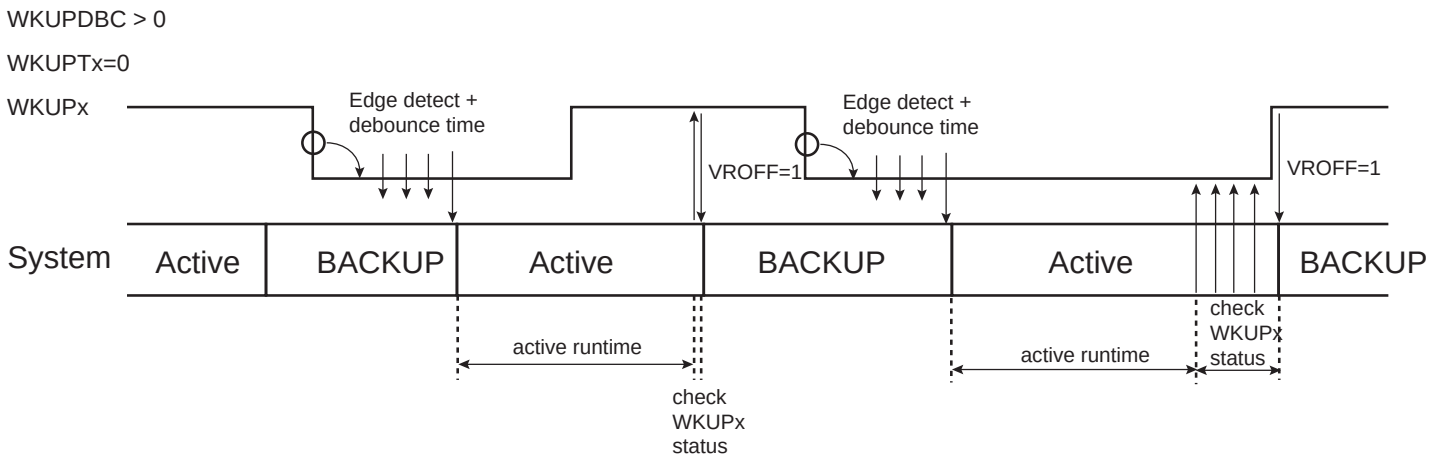
The wake-up inputs, WKUPx, can be programmed to perform a wake-up of the core power supply. Each input can be enabled by writing a 1 to the corresponding bit, WKUPENx, in the Wake-up Inputs register (SUPC_WUIR). The wake-up level can be selected with the corresponding polarity bit, WKUPTx, also located in SUPC_WUIR.

The resulting signals are wired-ORed to trigger a debounce counter, which is programmed with the WKUPDBC field in SUPC_WUMR. The WKUPDBC field selects a debouncing period of 3, 32, 512, 4,096 or 32,768 slow clock cycles. The duration of these periods 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 duration longer than the debouncing period, a wake-up of the core power supply is started and the signals, WKUP0 to WKUPx as shown in Figure 18-4 "Wake-up Sources", are latched in 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.

Before instructing the system to enter Backup mode, if the field WKUPDBC > 0, it must be checked that none of the WKUPx pins that are enabled for a wake-up (exit from Backup mode) holds an active polarity. This is checked by reading the pin status in the PIO Controller. If WKUPENx=1 and the pin WKUPx holds an active polarity, the system must not be instructed to enter Backup mode.

Figure 18-5. Entering and Exiting Backup Mode with a WKUP Pin



18.4.7.3 Low-power Tamper Detection and Anti-Tampering

Low-power debouncer inputs (WKUP0, WKUP1) can be used for tamper detection. If the tamper sensor is biased through a resistor and constantly driven by the power supply, this leads to power consumption as long as the tamper detection switch is in its active state. To prevent power consumption when the switch is in active state, the tamper sensor circuitry must be intermittently powered, and thus a specific waveform must be applied to the sensor circuitry.

The waveform is generated using RTCOUTx in all modes including Backup mode. Refer to the section “Real-Time Clock (RTC)” for waveform generation.

Separate debouncers are embedded, one for WKUP0 input, one for WKUP1 input.

The WKUP0 and/or WKUP1 inputs perform a system wake-up upon tamper detection. This is enabled by setting the LPDBCEN0/1 bit in the SUPC_WUMR.

WKUP0 and/or WKUP1 inputs can also be used when VDDCORE is powered to detect a tamper.

When the bit LPDBCENx is written to 1, WKUPx pins must not be configured to act as a debouncing source for the WKUPDBC counter (WKUPENx must be cleared in SUPC_WUIR).

Low-power tamper detection or debounce requires RTC output (RTCOUTx) to be configured to generate a duty cycle programmable pulse (i.e., OUT0 = 0x7 in RTC_MR) in order to create the sampling points of both debouncers. The sampling point is the falling edge of the RTCOUTx waveform.

Figure 18-6 shows an example of an application where two tamper switches are used. RTCOUTx powers the external pull-up used by the tamper sensor circuitry.

Figure 18-6. Low-power Debouncer (Push-to-Make Switch, Pull-up Resistors)

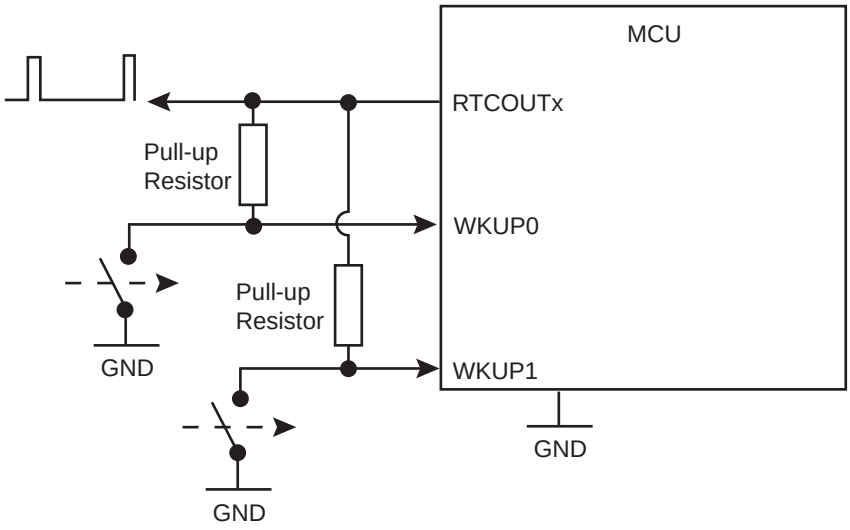
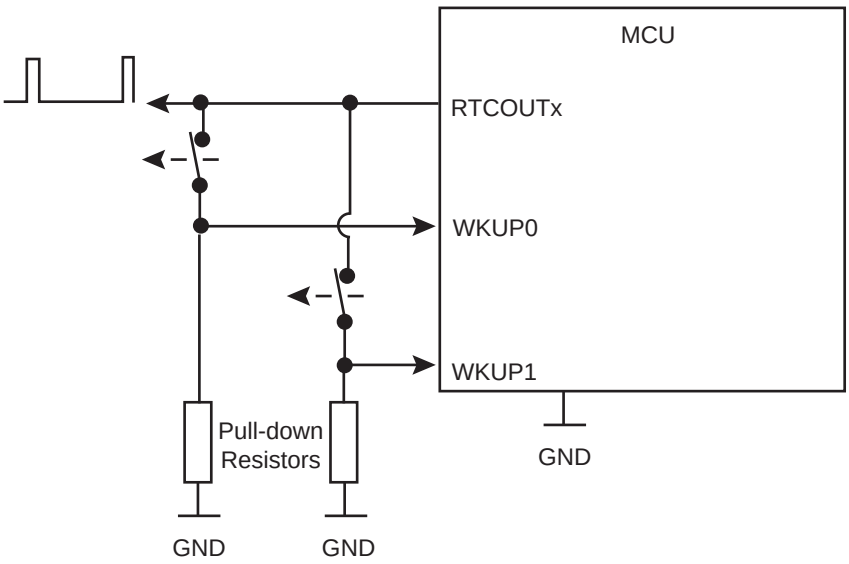


Figure 18-7. Low-power Debouncer (Push-to-Break Switch, Pull-down Resistors)



The debouncing period duration is configurable. The period is set for all debouncers (i.e., the duration cannot be adjusted for each debouncer). The number of successive identical samples to wake up the system can be configured from 2 up to 8 in the LPDBC field of SUPC_WUMR. The period of time between two samples can be configured by programming the TPERIOD field in the RTC_MR. Power parameters can be adjusted by modifying the period of time in the THIGH field in RTC_MR.

The wake-up polarity of the inputs can be independently configured by writing WKUPT0 and/ or WKUPT1 fields in SUPC_WUMR.

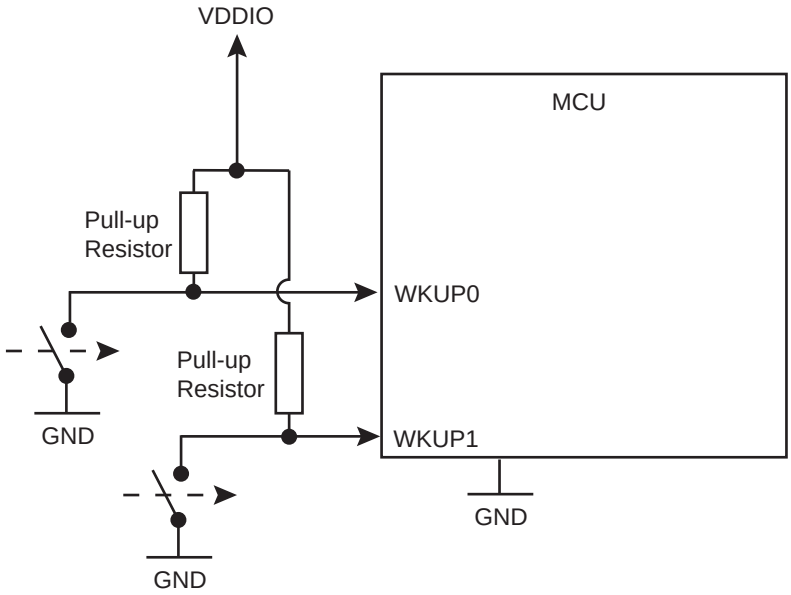
In order to determine which wake-up/tamper pin triggers the system wake-up, a status flag is associated for each low-power debouncer. These flags are read in SUPC_SR.

A debounce event (tamper detection) can perform an immediate clear (0 delay) on the first half the general-purpose backup registers (GPBR). The LPDBCCLR bit must be set in SUPC_WUMR.

Note that it is not mandatory to use the RTCOUTx pin when using the WKUP0/WKUP1 pins as tampering inputs in any mode. Using the RTCOUTx pin provides a “sampling mode” to further reduce the power consumption of the tamper detection circuitry. If RTCOUTx is not used, the RTC must be configured to create an internal sampling point for the debouncer logic. The period of time between two samples can be configured by programming the TPERIOD field in RTC_MR.

Figure 18-8 illustrates the use of WKUPx without the RTCOUTx pin.

Figure 18-8. Using WKUP Pins Without RTCOUTx Pins



18.4.7.4 Clock Alarms

The RTC and the RTT alarms can generate a wake-up of the core power supply. This can be enabled by setting, respectively, the bits RTCEN and RTTEN in 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.

18.4.7.5 Supply Monitor Detection

The supply monitor can generate a wake-up of the core power supply. See Section 18.4.4 ”Supply Monitor”.

18.4.8 Register Write Protection

To prevent any single software error from corrupting SYSC behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the "System Controller Write Protection Mode Register" (SYSC_WPMR).

The following registers can be write-protected:

- RSTC Mode Register
- RTT Mode Register
- RTT Alarm Register
- RTC Control Register
- RTC Mode Register
- RTC Time Alarm Register
- RTC Calendar Alarm Register
- General Purpose Backup Registers
- Supply Controller Control Register
- Supply Controller Supply Monitor Mode Register
- Supply Controller Mode Register
- Supply Controller Wake-up Mode Register

18.4.9 Register Bits in Backup Domain (VDDIO)

The following configuration registers, or certain bits of the registers, are physically located in the product backup domain:

- RSTC Mode Register (all bits)
- RTT Mode Register (all bits)
- RTT Alarm Register (all bits)
- RTC Control Register (all bits)
- RTC Mode Register (all bits)
- RTC Time Alarm Register (all bits)
- RTC Calendar Alarm Register (all bits)
- General Purpose Backup Registers (all bits)
- Supply Controller Control Register (see register description for details)
- Supply Controller Supply Monitor Mode Register (all bits)
- Supply Controller Mode Register (see register description for details)
- Supply Controller Wake-up Mode Register (all bits)
- Supply Controller Wake-up Inputs Register (all bits)
- Supply Controller Status Register (all bits)

18.5 Supply Controller (SUPC) User Interface

The user interface of the Supply Controller is part of the System Controller user interface.

18.5.1 System Controller (SYSC) User Interface

Table 18-1. System Controller Registers

Offset	System Controller Peripheral	Name
0x00-0x0c	Reset Controller	RSTC
0x10-0x2C	Supply Controller	SUPC
0x30-0x3C	Real Time Timer	RTT
0x50-0x5C	Watchdog Timer	WDT
0x60-0x8C	Real Time Clock	RTC
0x90-0xDC	General Purpose Backup Register	GPBR
0xE0	Reserved	–
0xE4	Write Protection Mode Register	SYSC_WPMR
0xE8-0xF8	Reserved	–
0x100-0x10C	Reinforced Safety Watchdog Timer	RSWDT

18.5.2 Supply Controller (SUPC) User Interface

Table 18-2. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Supply Controller Control Register	SUPC_CR	Write-only	–
0x04	Supply Controller Supply Monitor Mode Register	SUPC_SMMR	Read/Write	0x0000_0000
0x08	Supply Controller Mode Register	SUPC_MR	Read/Write	0x0000_5A00
0x0C	Supply Controller Wake-up Mode Register	SUPC_WUMR	Read/Write	0x0000_0000
0x10	Supply Controller Wake-up Inputs Register	SUPC_WUIR	Read/Write	0x0000_0000
0x14	Supply Controller Status Register	SUPC_SR	Read-only	0x0000_0000
0x18	Reserved	–	–	–
0xD4	Write Protection Mode Register	SYSC_WPMR	Read/Write	0x0000_0000

18.5.3 Supply Controller Control Register

Name: SUPC_CR

Address: 0x400E1810

Access: Write-only

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

This register can only be written if the WPEN bit is cleared in the System Controller Write Protection Mode Register (SYSC_MR).

- **VROFF: Voltage Regulator Off**

0 (NO_EFFECT): No effect.

1 (STOP_VREG): If KEY is correct, VROFF asserts the vddcore_nreset and stops the voltage regulator.

Note: This bit is located in the VDDIO domain.

- **XTALSEL: Crystal Oscillator Select**

0 (NO_EFFECT): No effect.

1 (CRYSTAL_SEL): If KEY is correct, XTALSEL switches the slow clock on the crystal oscillator output.

Note: This bit is located in the VDDIO domain.

- **KEY: Password**

Value	Name	Description
0xA5	PASSWD	Writing any other value in this field aborts the write operation.

18.5.4 Supply Controller Supply Monitor Mode Register

Name: SUPC_SMMR

Address: 0x400E1814

Access: Read/Write

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

This register is located in the VDDIO domain.

This register can only be written if the WPEN bit is cleared in the System Controller Write Protection Mode Register (SYSC_MR).

- **SMTH: Supply Monitor Threshold**

Selects the threshold voltage of the supply monitor. Refer to the section “Electrical Characteristics” for voltage values.

- **SMSMPL: Supply Monitor Sampling Period**

Value	Name	Description
0x0	SMD	Supply Monitor disabled
0x1	CSM	Continuous Supply Monitor
0x2	32SLCK	Supply Monitor enabled one SLCK period every 32 SLCK periods
0x3	256SLCK	Supply Monitor enabled one SLCK period every 256 SLCK periods
0x4	2048SLCK	Supply Monitor enabled one SLCK period every 2,048 SLCK periods

- **SMRSTEN: Supply Monitor Reset Enable**

0 (NOT_ENABLE): The core reset signal vddcore_nreset is not affected when a supply monitor detection occurs.

1 (ENABLE): The core reset signal, vddcore_nreset is asserted when a supply monitor detection occurs.

- **SMIEN: Supply Monitor Interrupt Enable**

0 (NOT_ENABLE): The SUPC interrupt signal is not affected when a supply monitor detection occurs.

1 (ENABLE): The SUPC interrupt signal is asserted when a supply monitor detection occurs.

18.5.5 Supply Controller Mode Register

Name: SUPC_MR

Address: 0x400E1818

Access: Read/Write

31	30	29	28	27	26	25	24
KEY							

23	22	21	20	19	18	17	16
–	–	–	OSCBYPASS	–	–	–	–

15	14	13	12	11	10	9	8
–	ONREG	BODDIS	BODRSTEN	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

This register can only be written if the WPEN bit is cleared in the System Controller Write Protection Mode Register (SYSC_MR).

- **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.

Note: This bit is located in the VDDIO domain.

- **BODDIS: Brownout Detector Disable**

0 (ENABLE): The core brownout detector is enabled.

1 (DISABLE): The core brownout detector is disabled.

Note: This bit is located in the VDDIO domain.

- **ONREG: Voltage Regulator Enable**

0 (ONREG_UNUSED): Internal voltage regulator is not used (external power supply is used).

1 (ONREG_USED): Internal voltage regulator is used.

Note: This bit is located in the VDDIO domain.

- **OSCBYPASS: Oscillator Bypass**

0 (NO_EFFECT): No effect. Clock selection depends on the value of XTALSEL (SUPC_CR).

1 (BYPASS): The 32 kHz crystal oscillator is bypassed if XTALSEL (SUPC_CR) is set. OSCBYPASS must be set prior to setting XTALSEL.

Note: This bit is located in the VDDIO domain.

- **KEY: Password Key**

Value	Name	Description
0xA5	PASSWD	Writing any other value in this field aborts the write operation.

18.5.6 Supply Controller Wake-up Mode Register

Name: SUPC_WUMR

Address: 0x400E181C

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	LPDBC		

15	14	13	12	11	10	9	8
–	WKUPDBC			–	FWUPDBC		

7	6	5	4	3	2	1	0
LPDBCCLR	LPDBCEN1	LPDBCEN0	–	RTCEN	RTTEN	SMEN	FWUPEN

This register is located in the VDDIO domain.

This register can only be written if the WPEN bit is cleared in the System Controller Write Protection Mode Register (SYSC_MR).

- **FWUPEN: Force Wake-up Enable**

0 (NOT_ENABLE): The force wake-up pin has no wake-up effect.

1 (ENABLE): The force wake-up pin low forces the wake-up of the core power supply.

- **SMEN: Supply Monitor Wake-up Enable**

0 (NOT_ENABLE): The supply monitor detection has no wake-up effect.

1 (ENABLE): The supply monitor detection forces the wake-up of the core power supply.

- **RTTEN: Real-time Timer Wake-up Enable**

0 (NOT_ENABLE): The RTT alarm signal has no wake-up effect.

1 (ENABLE): The RTT alarm signal forces the wake-up of the core power supply.

- **RTCEN: Real-time Clock Wake-up Enable**

0 (NOT_ENABLE): The RTC alarm signal has no wake-up effect.

1 (ENABLE): The RTC alarm signal forces the wake-up of the core power supply.

- **LPDBCEN0: Low-power Debouncer Enable WKUP0**

0 (NOT_ENABLE): The WKUP0 input pin is not connected to the low-power debouncer.

1 (ENABLE): The WKUP0 input pin is connected to the low-power debouncer and forces a system wake-up.

- **LPDBCEN1: Low-power Debouncer Enable WKUP1**

0 (NOT_ENABLE): The WKUP1 input pin is not connected to the low-power debouncer.

1 (ENABLE): The WKUP1 input pin is connected to the low-power debouncer and forces a system wake-up.

- **LPDBCCLR: Low-power Debouncer Clear**

0 (NOT_ENABLE): A low-power debounce event does not create an immediate clear on the first half of GPBR registers.

1 (ENABLE): A low-power debounce event on WKUP0 or WKUP1 generates an immediate clear on the first half of GPBR registers.

- **FWUPDBC: Force Wake-up Debouncer Period**

Value	Name	Description
0	IMMEDIATE	Immediate, no debouncing, detected active at least on one Slow Clock edge.
1	3_SCLK	FWUP shall be low for at least 3 SLCK periods
2	32_SCLK	FWUP shall be low for at least 32 SLCK periods
3	512_SCLK	FWUP shall be low for at least 512 SLCK periods
4	4096_SCLK	FWUP shall be low for at least 4,096 SLCK periods
5	32768_SCLK	FWUP shall be low for at least 32,768 SLCK periods

- **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

- **LPDBC: Low-power Debouncer Period**

Value	Name	Description
0	DISABLE	Disables the low-power debouncers.
1	2_RTCOUT0	WKUP0/1 in active state for at least 2 RTCOUTx clock periods
2	3_RTCOUT0	WKUP0/1 in active state for at least 3 RTCOUTx clock periods
3	4_RTCOUT0	WKUP0/1 in active state for at least 4 RTCOUTx clock periods
4	5_RTCOUT0	WKUP0/1 in active state for at least 5 RTCOUTx clock periods
5	6_RTCOUT0	WKUP0/1 in active state for at least 6 RTCOUTx clock periods
6	7_RTCOUT0	WKUP0/1 in active state for at least 7 RTCOUTx clock periods
7	8_RTCOUT0	WKUP0/1 in active state for at least 8 RTCOUTx clock periods

18.5.7 Supply Controller Wake-up Inputs Register

Name: SUPC_WUIR

Address: 0x400E1820

Access: Read/Write

31	30	29	28	27	26	25	24
WKUPT15	WKUPT14	WKUPT13	WKUPT12	WKUPT11	WKUPT10	WKUPT9	WKUPT8
23	22	21	20	19	18	17	16
WKUPT7	WKUPT6	WKUPT5	WKUPT4	WKUPT3	WKUPT2	WKUPT1	WKUPT0
15	14	13	12	11	10	9	8
WKUPEN15	WKUPEN14	WKUPEN13	WKUPEN12	WKUPEN11	WKUPEN10	WKUPEN9	WKUPEN8
7	6	5	4	3	2	1	0
WKUPEN7	WKUPEN6	WKUPEN5	WKUPEN4	WKUPEN3	WKUPEN2	WKUPEN1	WKUPEN0

This register is located in the VDDIO domain.

- **WKUPEN0 - WKUPENx: Wake-up Input Enable 0 to x**

0 (DISABLE): The corresponding wake-up input has no wake-up effect.

1 (ENABLE): The corresponding wake-up input is enabled for a wake-up of the core power supply.

- **WKUPT0 - WKUPTx: Wake-up Input Type 0 to x**

0 (LOW): A falling edge followed by a low level for a period defined by WKUPDBC on the corresponding wake-up input forces the wake-up of the core power supply.

1 (HIGH): A rising edge followed by a high level for a period defined by WKUPDBC on the corresponding wake-up input forces the wake-up of the core power supply.

18.5.8 Supply Controller Status Register

Name: SUPC_SR

Address: 0x400E1824

Access: Read-only

31	30	29	28	27	26	25	24
WKUPIS15	WKUPIS14	WKUPIS13	WKUPIS12	WKUPIS11	WKUPIS10	WKUPIS9	WKUPIS8
23	22	21	20	19	18	17	16
WKUPIS7	WKUPIS6	WKUPIS5	WKUPIS4	WKUPIS3	WKUPIS2	WKUPIS1	WKUPIS0
15	14	13	12	11	10	9	8
–	LPDBCS1	LPDBCS0	FWUPIS	–	–	–	–
7	6	5	4	3	2	1	0
OSCSEL	SMOS	SMS	SMRSTS	BODRSTS	SMWS	WKUPS	FWUPS

Note: Because of the asynchronism between the Slow Clock (SLCK) 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.

This register is located in the VDDIO domain.

- **FWUPS: FWUP Wake-up Status (cleared on read)**

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 (cleared on read)**

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 (cleared on read)**

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 (cleared on read)**

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 (cleared on read)**

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 (cleared on read)**

0 (NO): No supply monitor detection since the last read of SUPC_SR.

1 (PRESENT): At least one supply monitor detection since the last read of SUPC_SR.

- **SMOS: Supply Monitor Output Status**

0 (HIGH): The supply monitor detected VDDIO higher than its threshold at its last measurement.

1 (LOW): The supply monitor detected VDDIO lower than its threshold at its last measurement.

- **OSCSEL: 32-kHz Oscillator Selection Status**

0 (RC): The slow clock, SLCK, is generated by the embedded 32 kHz RC oscillator.

1 (CRYST): The slow clock, SLCK, is generated by the 32 kHz crystal oscillator.

- **FWUPIS: FWUP Input Status**

0 (LOW): FWUP input is tied low.

1 (HIGH): FWUP input is tied high.

- **LPDBCS0: Low-power Debouncer Wake-up Status on WKUP0 (cleared on read)**

0 (NO): No wake-up due to the assertion of the WKUP0 pin has occurred since the last read of SUPC_SR.

1 (PRESENT): At least one wake-up due to the assertion of the WKUP0 pin has occurred since the last read of SUPC_SR.

- **LPDBCS1: Low-power Debouncer Wake-up Status on WKUP1 (cleared on read)**

0 (NO): No wake-up due to the assertion of the WKUP1 pin has occurred since the last read of SUPC_SR.

1 (PRESENT): At least one wake-up due to the assertion of the WKUP1 pin has occurred since the last read of SUPC_SR.

- **WKUPISx: WKUPx Input Status (cleared on read)**

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 since the last read of SUPC_SR.

18.5.9 System Controller Write Protection Mode Register

Name: SYSC_WPMR

Address: 0x400E18E4

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 0x525443 (“RTC” in ASCII).

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

See Section 18.4.8 ”Register Write Protection” for the list of registers that can be write-protected.

- **WPKEY: Write Protection Key**.

Value	Name	Description
0x525443	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

19. General Purpose Backup Registers (GPBR)

19.1 Description

The System Controller embeds 640 bits of General Purpose Backup registers organized as 20 32-bit registers.

It is possible to generate an immediate clear of the content of General Purpose Backup registers 0 to 9 (first half) if a Low-power Debounce event is detected on one of the wakeup pins, WKUP0 or WKUP1. The content of the other General Purpose Backup registers (second half) remains unchanged.

The Supply Controller module must be programmed accordingly. In the register SUPC_WUMR in the Supply Controller module, LPDBCCLR, LPDBCEN0 and/or LPDBCEN1 bit must be configured to 1 and LPDBC must be other than 0.

If a Tamper event has been detected, it is not possible to write to the General Purpose Backup registers while the LPDBCS0 or LPDBCS1 flags are not cleared in the Supply Controller Status Register (SUPC_SR).

19.2 Embedded Characteristics

- 640 bits of General Purpose Backup Registers

19.3 General Purpose Backup Registers (GPBR) User Interface

Table 19-1. Register Mapping

Offset	Register	Name	Access	Reset
0x0	General Purpose Backup Register 0	SYS_GPBR0	Read/Write	0x00000000
...	...	...	...	...
0x64	General Purpose Backup Register 19	SYS_GPBR19	Read/Write	0x00000000

19.3.1 General Purpose Backup Register x

Name: SYS_GPBRx

Address: 0x400E1890

Access: Read/Write

31	30	29	28	27	26	25	24
GPBR_VALUE							

23	22	21	20	19	18	17	16
GPBR_VALUE							

15	14	13	12	11	10	9	8
GPBR_VALUE							

7	6	5	4	3	2	1	0
GPBR_VALUE							

These registers are reset at first power-up and on each loss of VDDIO.

- **GPBR_VALUE: Value of GPBR x**

If a Tamper event has been detected, it is not possible to write GPBR_VALUE as long as the LPDBCS0 or LPDBCS1 flag has not been cleared in the Supply Controller Status Register (SUPC_SR).

20. Enhanced Embedded Flash Controller (EEFC)

20.1 Description

The Enhanced Embedded Flash Controller (EEFC) provides 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.

20.2 Embedded Characteristics

- Increases Performance in Thumb-2 Mode with 128-bit or 64-bit-wide Memory Interface up to 120 MHz
- Code Loop Optimization
- 128 Lock Bits, Each Protecting a Lock Region
- GPNVMx General-purpose GPNVM Bits
- One-by-one Lock Bit Programming
- Commands Protected by a Keyword
- Erase the Entire Flash
- Erase by Plane
- Erase by Sector
- Erase by Page
- Provides Unique Identifier
- Provides 512-byte User Signature Area
- Supports Erasing before Programming
- Locking and Unlocking Operations
- Supports Read of the Calibration Bits

20.3 Product Dependencies

20.3.1 Power Management

The Enhanced Embedded Flash Controller (EEFC) is continuously clocked. The Power Management Controller has no effect on its behavior.

20.3.2 Interrupt Sources

The EEFC interrupt line is connected to the interrupt controller. Using the EEFC interrupt requires the interrupt controller to be programmed first. The EEFC interrupt is generated only if the value of EEFC_FMR.FRDY is ‘1’.

Table 20-1. Peripheral IDs

Instance	ID
EFC	6

20.4 Functional Description

20.4.1 Embedded Flash Organization

The embedded Flash interfaces directly with the internal bus. The embedded Flash is composed of:

- One memory plane organized in several pages of the same size for the code
- A separate 2 x 512-byte memory area which includes the unique chip identifier
- A separate 512-byte memory area for the user signature
- 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 EEFC interface, called general-purpose non-volatile memory bits (GPNVM bits)

The embedded Flash size, the page size, the organization of lock regions and the definition of GPNVM bits are specific to the device. The EEFC returns a descriptor of the Flash controller after a ‘Get Flash Descriptor’ command has been issued by the application (see Section 20.4.3.1 ”Get Flash Descriptor Command”).

Figure 20-1. Flash Memory Areas

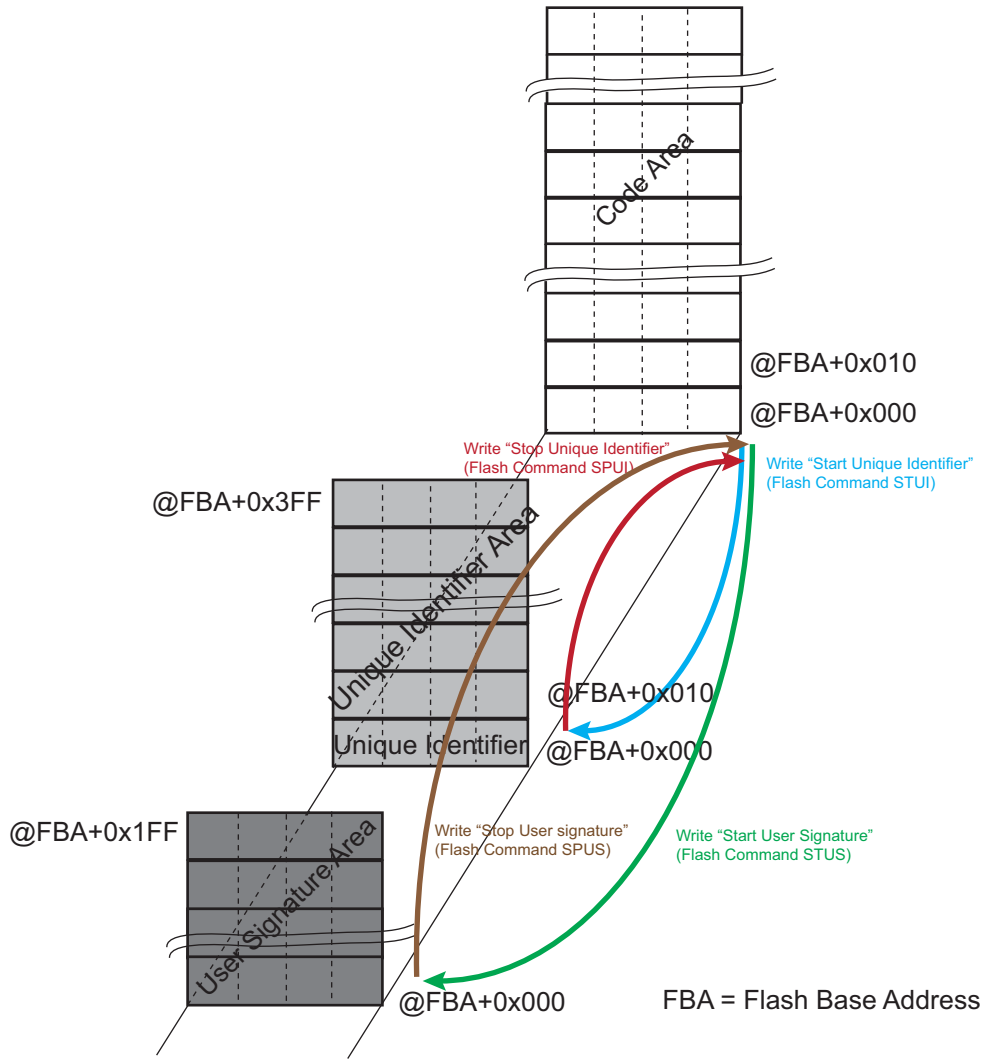
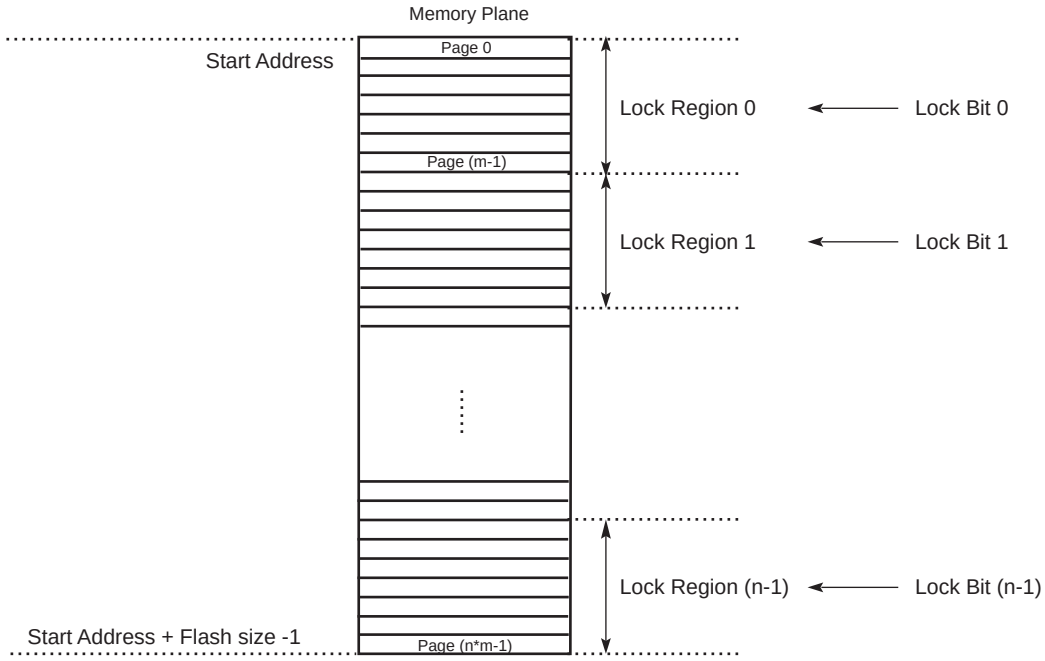


Figure 20-2. Organization of Embedded Flash for Code



20.4.2 Read Operations

An optimized controller manages embedded Flash reads, thus increasing performance when the processor is running in Thumb-2 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 in the Flash Mode register (EEFC_FMR). Defining FWS as 0 enables the single-cycle access of the embedded Flash. For more details, refer to the section “Electrical Characteristics” of this datasheet.

20.4.2.1 128- or 64-bit Access Mode

By default, the read accesses of the Flash are performed through a 128-bit wide memory interface. It improves system performance especially when two or three wait states are needed.

For systems requiring only 1 wait state, or to focus on current consumption rather than performance, the user can select a 64-bit wide memory access via the bit EEFC_FMR.FAM.

For more details, refer to the section “Electrical Characteristics” of this datasheet.

20.4.2.2 Code Read Optimization

Code read optimization is enabled if the bit EEFC_FMR.SCOD is cleared.

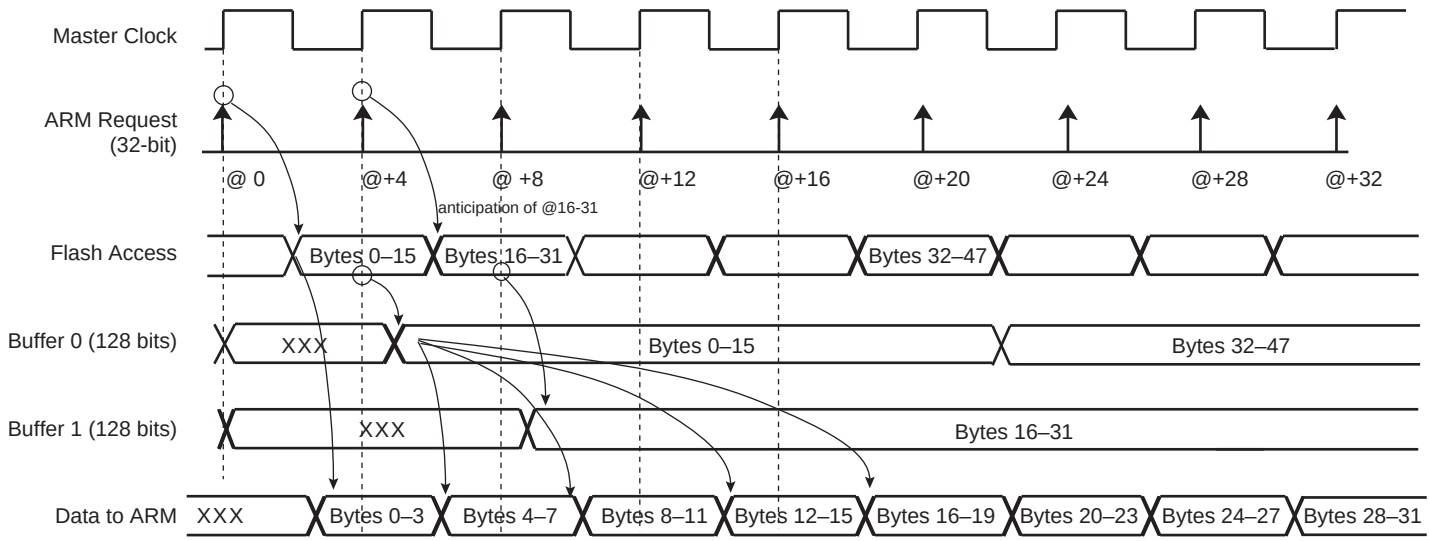
A system of 2 x 128-bit or 2 x 64-bit buffers is added in order to optimize sequential code fetch.

Note: Immediate consecutive code read accesses are not mandatory to benefit from this optimization.

The sequential code read optimization is enabled by default. If the bit EEFC_FMR.SCOD is set, these buffers are disabled and the sequential code read is no longer optimized.

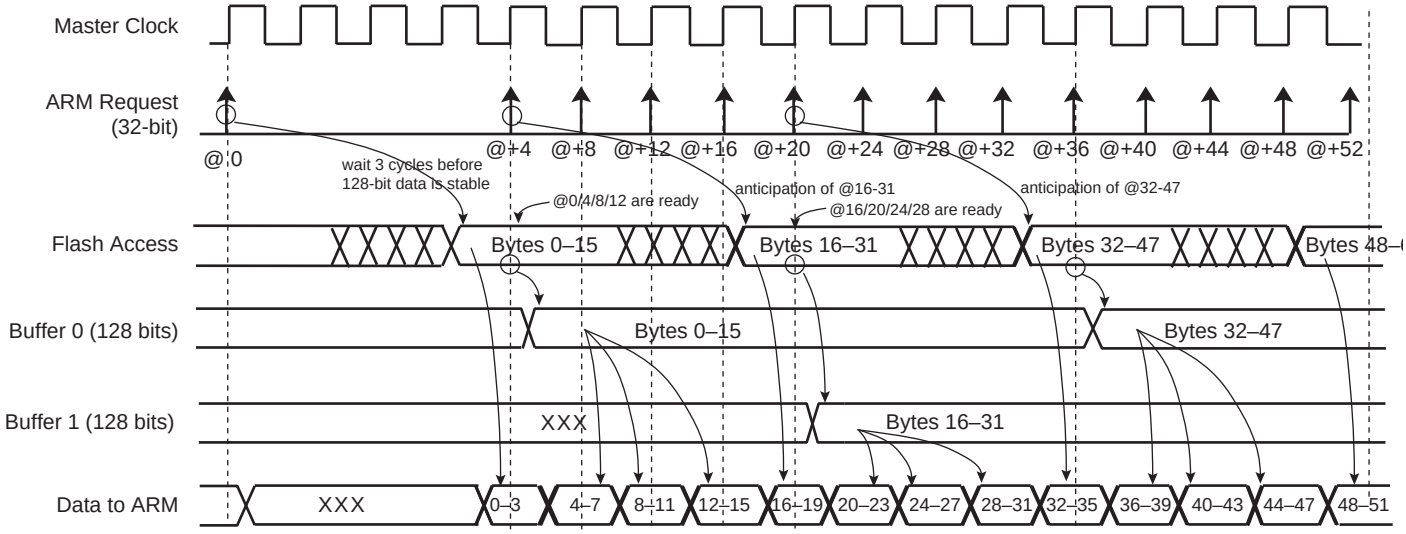
Another system of 2 x 128-bit or 2 x 64-bit buffers is added in order to optimize loop code fetch. Refer to Section 20.4.2.3 ”Code Loop Optimization” for more details.

Figure 20-3. Code Read Optimization for FWS = 0



Note: When FWS is equal to 0, all the accesses are performed in a single-cycle access.

Figure 20-4. Code Read Optimization for FWS = 3



Note: When FWS is between 1 and 3, in case of sequential reads, the first access takes (FWS + 1) cycles. The following accesses take only one cycle.

20.4.2.3 Code Loop Optimization

Code loop optimization is enabled when the bit EEFC_FMR.CLOE is set.

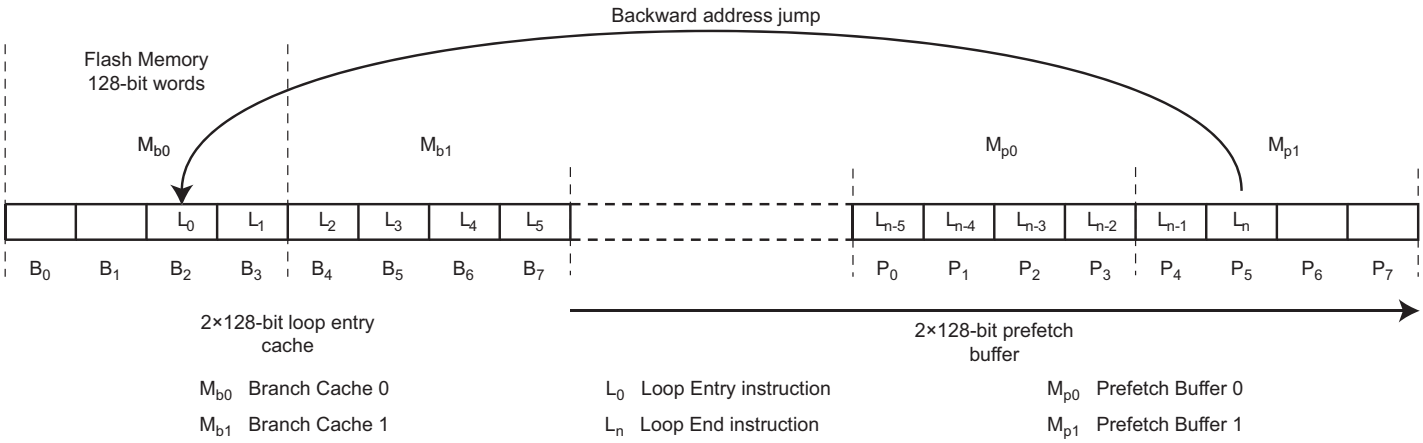
When a backward jump is inserted in the code, the pipeline of the sequential optimization is broken and becomes inefficient. In this case, the loop code read optimization takes over from the sequential code read optimization to prevent the insertion of wait states. The loop code read optimization is enabled by default. In EEFC_FMR, if the bit CLOE is reset to 0 or the bit SCOD is set, these buffers are disabled and the loop code read is not optimized.

When code loop optimization is enabled, if inner loop body instructions L_0 to L_n are positioned from the 128-bit Flash memory cell M_{b0} to the memory cell M_{p1}, after recognition of a first backward branch, the first two Flash memory cells M_{b0} and M_{b1} targeted by this branch are cached for fast access from the processor at the next loop iteration.

Then by combining the sequential prefetch (described in Section 20.4.2.2 ”Code Read Optimization”) through the loop body with the fast read access to the loop entry cache, the entire loop can be iterated with no wait state.

Figure 20-5 illustrates code loop optimization.

Figure 20-5. Code Loop Optimization

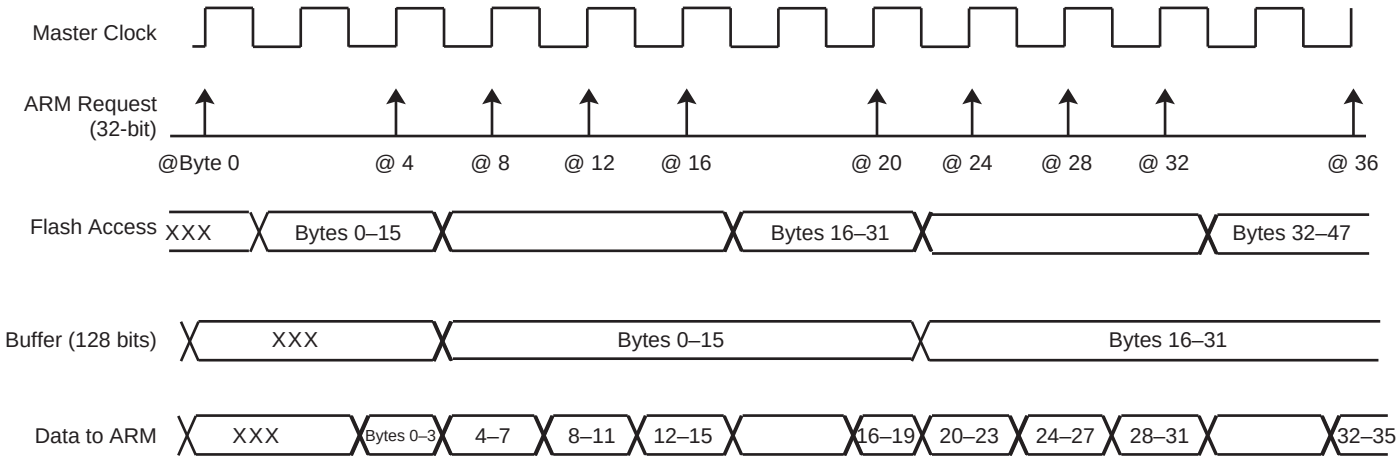


20.4.2.4 Data Read Optimization

The organization of the Flash in 128 bits or 64 bits is associated with two 128-bit or 64-bit prefetch buffers and one 128-bit or 64-bit data read buffer, thus providing maximum system performance. This buffer is added in order to store the requested data plus all the data contained in the 128-bit or 64-bit aligned data. This speeds up sequential data reads if, for example, FWS is equal to 1 (see Figure 20-6). The data read optimization is enabled by default. If the bit EEFC_FMR.SCOD is set, this buffer is disabled and the data read is no longer optimized.

Note: No consecutive data read accesses are mandatory to benefit from this optimization.

Figure 20-6. Data Read Optimization for FWS = 1



20.4.3 Flash Commands

The EEFC offers a set of commands to manage programming the Flash memory, locking and unlocking lock regions, consecutive programming, locking and full Flash erasing, etc.

The commands are listed in the following table.

Table 20-2. Set of Commands

Command	Value	Mnemonic
Get Flash descriptor	0x00	GETD
Write page	0x01	WP
Write page and lock	0x02	WPL
Erase page and write page	0x03	EWP
Erase page and write page then lock	0x04	EWPL
Erase all	0x05	EA
Erase pages	0x07	EPA
Set lock bit	0x08	SLB
Clear lock bit	0x09	CLB
Get lock bit	0x0A	GLB
Set GPNVM bit	0x0B	SGPB
Clear GPNVM bit	0x0C	CGPB

Table 20-2. Set of Commands (Continued)

Command	Value	Mnemonic
Get GPNVM bit	0x0D	GGPB
Start read unique identifier	0x0E	STUI
Stop read unique identifier	0x0F	SPUI
Get CALIB bit	0x10	GCALB
Erase sector	0x11	ES
Write user signature	0x12	WUS
Erase user signature	0x13	EUS
Start read user signature	0x14	STUS
Stop read user signature	0x15	SPUS

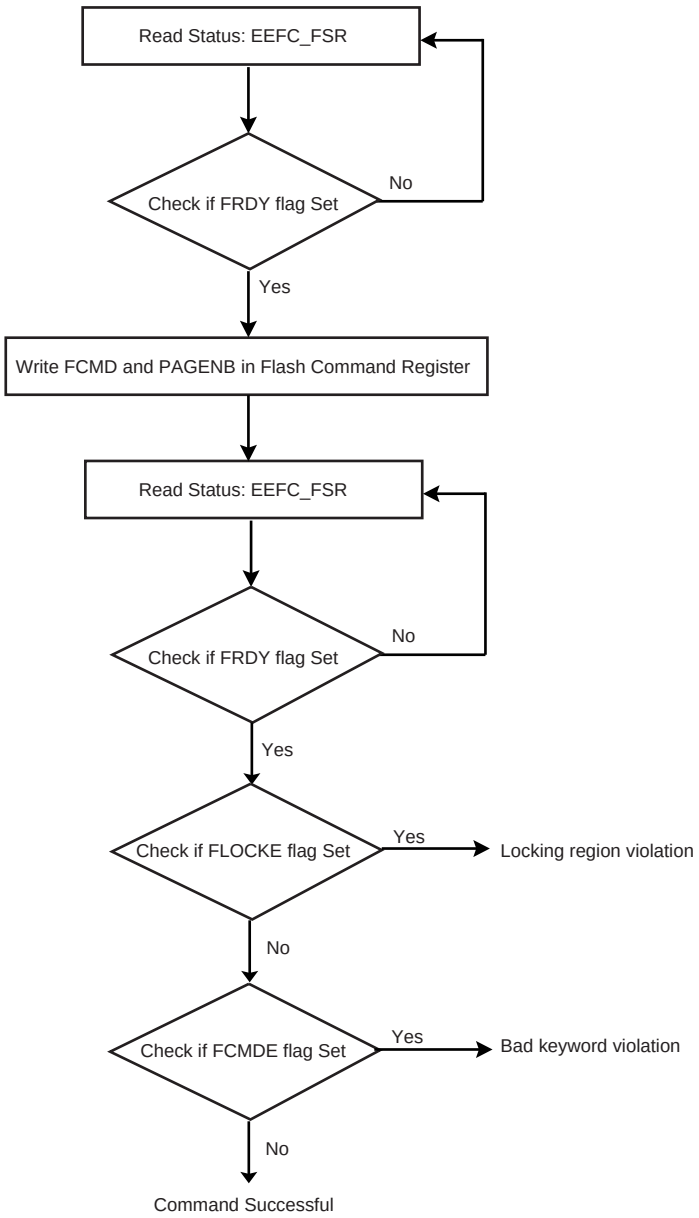
In order to execute one of these commands, select the required command using the FCMD field in the Flash Command register (EEFC_FCR). As soon as EEFC_FCR is written, the FRDY flag and the FVALUE field in the Flash Result register (EEFC_FRR) are automatically cleared. Once the current command has completed, the FRDY flag is automatically set. If an interrupt has been enabled by setting the bit EEFC_FMR.FRDY, the corresponding interrupt line of the interrupt controller 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 has completed.)

All the commands are protected by the same keyword, which must be written in the eight highest bits of EEFC_FCR.

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 Flash Status register (EEFC_FSR). This flag is automatically cleared by a read access to EEFC_FSR.

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 EEFC_FSR. This flag is automatically cleared by a read access to EEFC_FSR.

Figure 20-7. Command State Chart



20.4.3.1 Get Flash Descriptor Command

This command provides the system with information on 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 EEFC_FCR. The first word of the descriptor can be read by the software application in EEFC_FRR as soon as the FRDY flag in EEFC_FSR rises. The next reads of EEFC_FRR provide the following word of the descriptor. If extra read operations to EEFC_FRR are done after the last word of the descriptor has been returned, the EEFC_FRR value is 0 until the next valid command.

Table 20-3. 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 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

20.4.3.2 Write Commands

Several commands are used to program the Flash.

Only 0 values can be programmed using Flash technology; 1 is the erased value. In order to program words in a page, the page must first be erased. Commands are available to erase the full memory plane or a given number of pages. With the EWP and EWPL commands, a page erase is done automatically before a page programming.

After programming, the page (the entire 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 programmed in the Flash must be written in an internal latch buffer before writing the programming command in EEFC_FCR. Data can be written at their final destination address, as the latch buffer is mapped into the Flash memory address space and wraps around within this Flash address space.

Byte and half-word AHB accesses to the latch buffer are not allowed. Only 32-bit word accesses are supported.

32-bit words must be written continuously, in either ascending or descending order. Writing the latch buffer in a random order is not permitted. This prevents mapping a C-code structure to the latch buffer and accessing the data of the structure in any order. It is instead recommended to fill in a C-code structure in SRAM and copy it in the latch buffer in a continuous order.

Write operations in the latch buffer are performed with the number of wait states programmed for reading the Flash.

The latch buffer is automatically re-initialized, i.e., written with logical ‘1’, after execution of each programming command. However, after power-up, the latch buffer is not initialized. If only part of the page is to be written with user data, the remaining part must be erased (written with ‘1’).

The programming sequence is the following:

1. Write the data to be programmed in the latch buffer.
2. Write the programming command in EEFC_FCR. This automatically clears the bit EEFC_FSR.FRDY.
3. When Flash programming is completed, the bit EEFC_FSR.FRDY rises. If an interrupt has been enabled by setting the bit EEFC_FMR.FRDY, the interrupt line of the EEFC is activated.

Three errors can be detected in EEFC_FSR after a programming sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Lock Error: The page to be programmed belongs to a locked region. A command must be run previously to unlock the corresponding region.
- Flash Error: When programming is completed, the WriteVerify test of the Flash memory has failed.

Only one page can be programmed at a time. It is possible to program all the bits of a page (full page programming) or only some of the bits of the page (partial page programming).

Depending on the number of bits to be programmed within the page, the EEFC adapts the write operations required to program the Flash.

When a ‘Write Page’ (WP) command is issued, the EEFC starts the programming sequence and all the bits written at 0 in the latch buffer are cleared in the Flash memory array.

During programming, i.e., until EEFC_FSR.FDRY rises, access to the Flash is not allowed.

Full Page Programming

To program a full page, all the bits of the page must be erased before writing the latch buffer and issuing the WP command. The latch buffer must be written in ascending order, starting from the first address of the page. See Figure 20-8 "Full Page Programming".

Partial Page Programming

To program only part of a page using the WP command, the following constraints must be respected:

- Data to be programmed must be contained in integer multiples of 64-bit address-aligned words.
- 64-bit words can be programmed only if all the corresponding bits in the Flash array are erased (at logical value ‘1’).

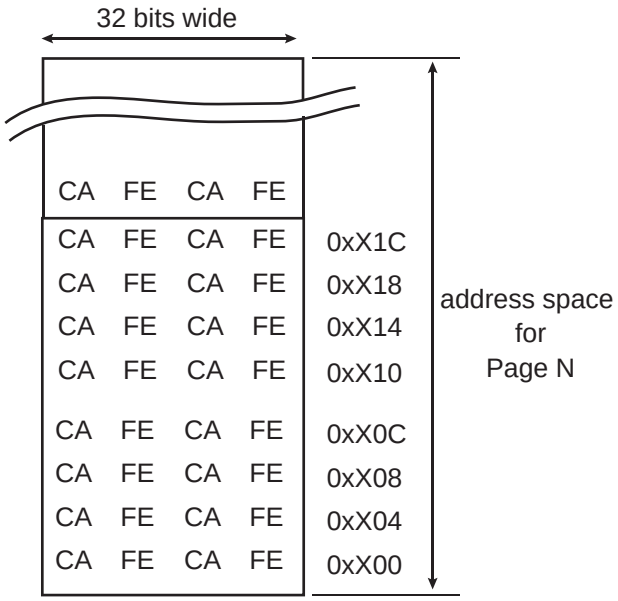
See Figure 20-9 "Partial Page Programming".

Programming Bytes

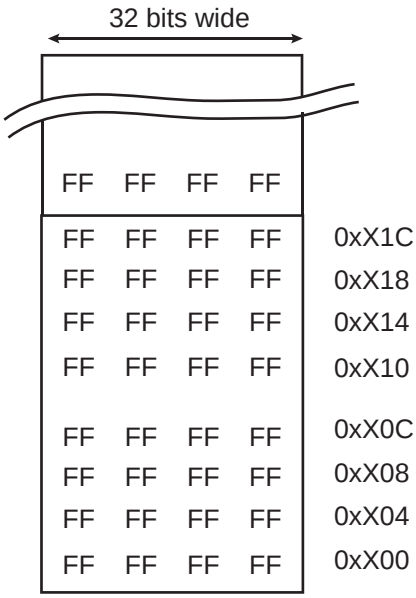
Individual bytes can be programmed using the Partial page programming mode.
In this case, an area of 64 bits must be reserved for each byte.

Refer to Figure 20-10 "Programming Bytes in the Flash".

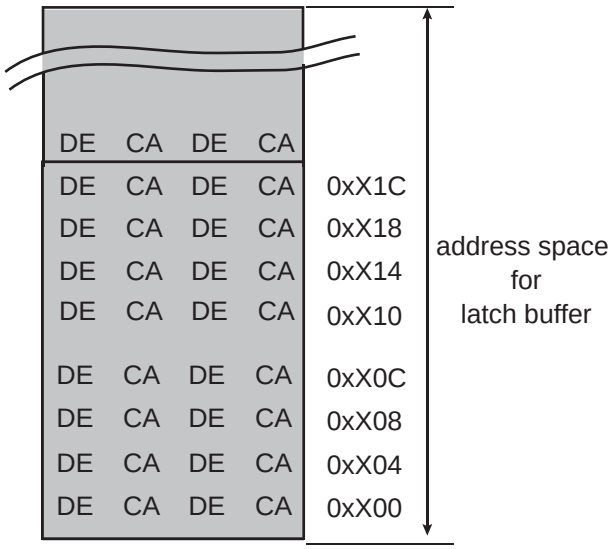
Figure 20-8. Full Page Programming



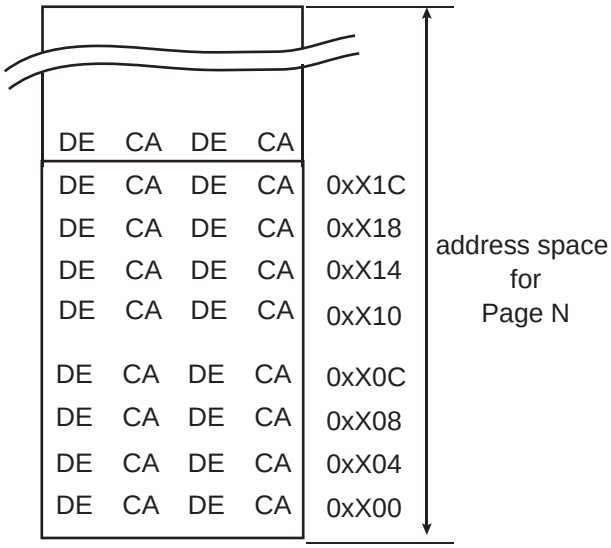
Before programming: Unerased page in Flash array



Step 1: Flash array after page erase

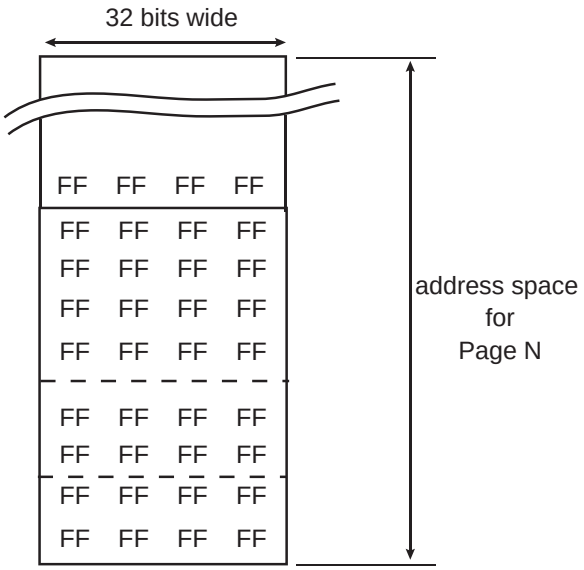


Step 2: Writing a page in the latch buffer

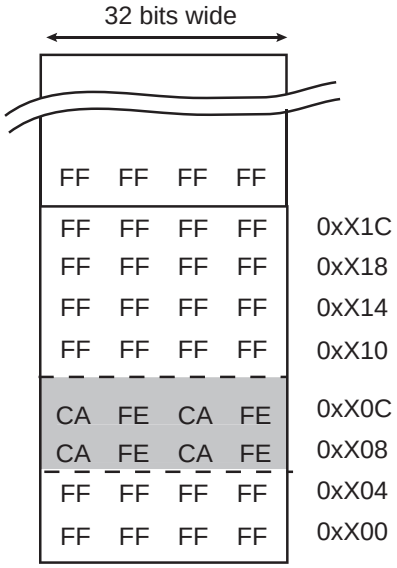


Step 3: Page in Flash array after issuing WP command and FRDY=1

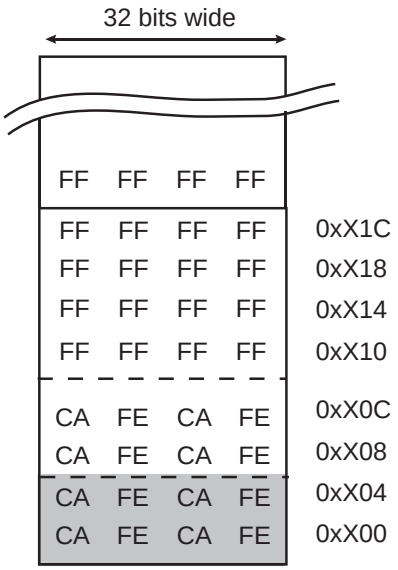
Figure 20-9. Partial Page Programming



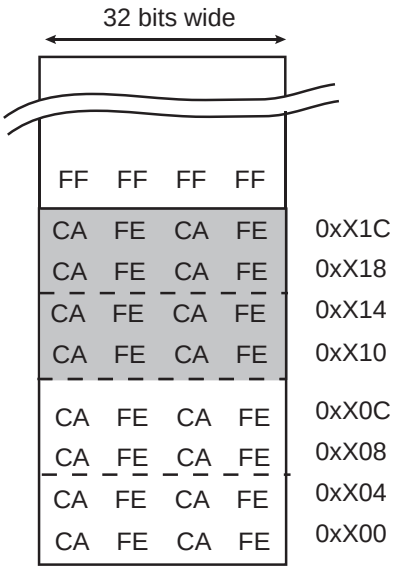
Step 1: Flash array after page erase



Step 2: Flash array after programming
64-bit at address 0xX08 (write latch buffer + WP)

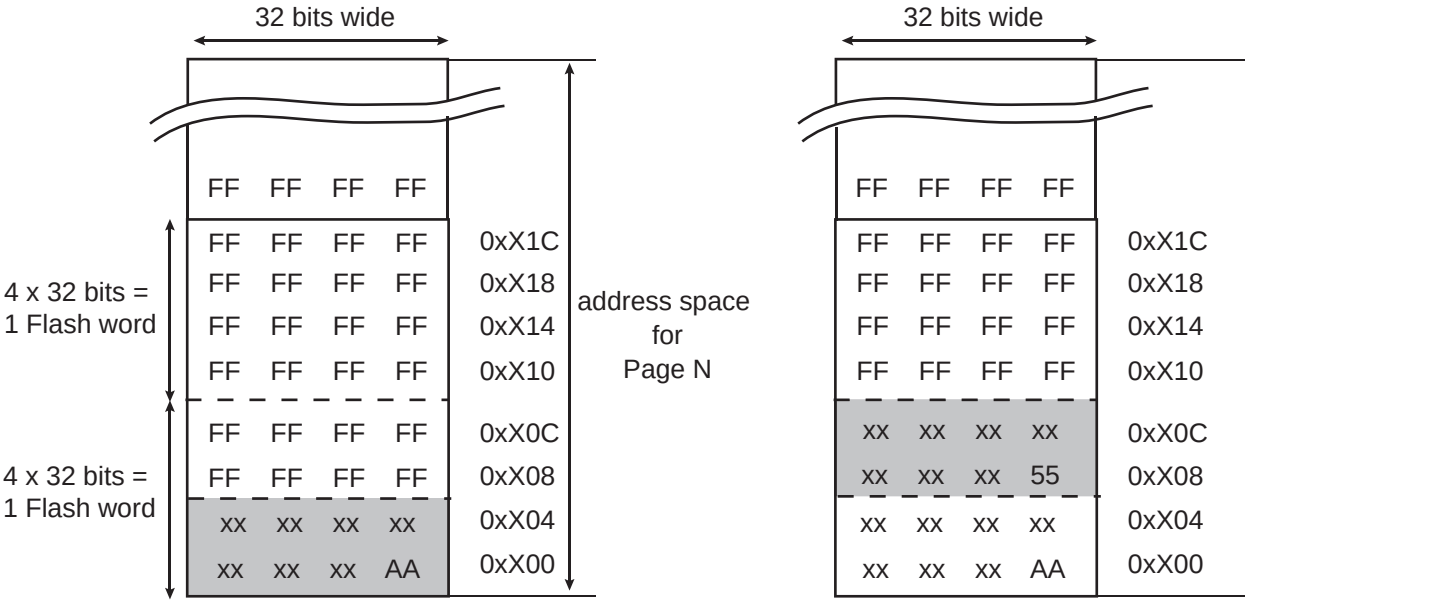


Step 3: Flash array after programming
a second 64-bit data at address 0xX00
(write latch buffer + WP)



Step 4: Flash array after programming
a 128-bit data word at address 0xX10
(write latch buffer + WP)

Figure 20-10. Programming Bytes in the Flash



Step 1: Flash array after programming first byte (0xAA)
64-bit used at address 0xX00 (write latch buffer + WP)

Step 2: Flash array after programming second byte (0x55)
64-bit used at address 0xX08 (write latch buffer + WP)

Note: The byte location shown here is for example only, it can be any byte location within a 64-bit word.

20.4.3.3 Erase Commands

Erase commands are allowed only on unlocked regions. Depending on the Flash memory, several commands can be used to erase the Flash:

- Erase All Memory (EA): All memory is erased. The processor must not fetch code from the Flash memory.
- Erase Pages (EPA): 8 or 16 pages are erased in the Flash sector selected. The first page to be erased is specified in the FARG[15:2] field of the EEFC_FCR. The first page number must be a multiple of 8, 16 or 32 depending on the number of pages to erase at the same time.
- Erase Sector (ES): A full memory sector is erased. Sector size depends on the Flash memory. EEFC_FCR.FARG must be set with a page number that is in the sector to be erased.

If the processor is fetching code from the Flash memory while the EPA or ES command is being executed, the processor accesses are stalled until the EPA command is completed. To avoid stalling the processor, the code can be run out of internal SRAM.

The erase sequence is the following:

1. Erase starts as soon as one of the erase commands and the FARG field are written in EEFC_FCR.
   – For the EPA command, the two lowest bits of the FARG field define the number of pages to be erased (FARG[1:0]):

Table 20-4. EEFC_FCR.FARG Field for EPA Command

FARG[1:0]	Number of pages to be erased with EPA command
0	4 pages (only valid for small 8 KB sectors)
1	8 pages
2	16 pages
3	32 pages (not valid for small 8 KB sectors)

2. When erasing is completed, the bit EEFC_FSR.FRDY rises. If an interrupt has been enabled by setting the bit EEFC_FMR.FRDY, the interrupt line of the interrupt controller is activated.

Three errors can be detected in EEFC_FSR after an erasing sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Lock Error: At least one page to be erased belongs to a locked region. The erase command has been refused, no page has been erased. A command must be run previously to unlock the corresponding region.
- Flash Error: At the end of the erase period, the EraseVerify test of the Flash memory has failed.

20.4.3.4 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 following:

1. Execute the ‘Set Lock Bit’ command by writing EEFC_FCR.FCMD with the SLB command and EEFC_FCR.FARG with a page number to be protected.
2. When the locking completes, the bit EEFC_FSR.FRDY rises. If an interrupt has been enabled by setting the bit EEFC_FMR.FRDY, the interrupt line of the interrupt controller is activated.
3. The result of the SLB command can be checked running a ‘Get Lock Bit’ (GLB) command.

Note: The value of the FARG argument passed together with SLB command must not exceed the higher lock bit index available in the product.

Two errors can be detected in EEFC_FSR after a programming sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Flash Error: At the end of the programming, the EraseVerify or WriteVerify test of the Flash memory has failed.

It is possible to clear lock bits previously set. After the lock bits are cleared, the locked region can be erased or programmed. The unlock sequence is the following:

1. Execute the ‘Clear Lock Bit’ command by writing EEFC_FCR.FCMD with the CLB command and EEFC_FCR.FARG with a page number to be unprotected.
2. When the unlock completes, the bit EEFC_FSR.FRDY rises. If an interrupt has been enabled by setting the bit EEFC_FMR.FRDY, the interrupt line of the interrupt controller is activated.

Note: The value of the FARG argument passed together with CLB command must not exceed the higher lock bit index available in the product.

Two errors can be detected in EEFC_FSR after a programming sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Flash Error: At the end of the programming, the EraseVerify or WriteVerify test of the Flash memory has failed.

The status of lock bits can be returned by the EEFC. The ‘Get Lock Bit’ sequence is the following:

1. Execute the ‘Get Lock Bit’ command by writing EEFC_FCR.FCMD with the GLB command. Field EEFC_FCR.FARG is meaningless.
2. Lock bits can be read by the software application in EEFC_FRR. 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 EEFC_FRR return 0.

For example, if the third bit of the first word read in EEFC_FRR is set, the third lock region is locked.

Two errors can be detected in EEFC_FSR after a programming sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Flash Error: At the end of the programming, the EraseVerify or WriteVerify test of the Flash memory has failed.

Note: Access to the Flash in read is permitted when a ‘Set Lock Bit’, ‘Clear Lock Bit’ or ‘Get Lock Bit’ command is executed.

20.4.3.5 GPNVM Bit

GPNVM bits do not interfere with the embedded Flash memory plane. For more details, refer to the section “Memories” of this datasheet.

The ‘Set GPNVM Bit’ sequence is the following:

1. Execute the ‘Set GPNVM Bit’ command by writing EEFC_FCR.FCMD with the SGPB command and EEFC_FCR.FARG with the number of GPNVM bits to be set.
2. When the GPNVM bit is set, the bit EEFC_FSR.FRDY rises. If an interrupt was enabled by setting the bit EEFC_FMR.FRDY, the interrupt line of the interrupt controller is activated.
3. The result of the SGPB command can be checked by running a ‘Get GPNVM Bit’ (GGPB) command.

Note: The value of the FARG argument passed together with SGPB command must not exceed the higher GPNVM index available in the product. Flash data content is not altered if FARG exceeds the limit. Command Error is detected only if FARG is greater than 8.

Two errors can be detected in EEFC_FSR after a programming sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Flash Error: At the end of the programming, the EraseVerify or WriteVerify test of the Flash memory has failed.

It is possible to clear GPNVM bits previously set. The ‘Clear GPNVM Bit’ sequence is the following:

1. Execute the ‘Clear GPNVM Bit’ command by writing EEFC_FCR.FCMD with the CGPB command and EEFC_FCR.FARG with the number of GPNVM bits to be cleared.
2. When the clear completes, the bit EEFC_FSR.FRDY rises. If an interrupt has been enabled by setting the bit EEFC_FMR.FRDY, the interrupt line of the interrupt controller is activated.

Note: The value of the FARG argument passed together with CGPB command must not exceed the higher GPNVM index available in the product. Flash data content is not altered if FARG exceeds the limit. Command Error is detected only if FARG is greater than 8.

Two errors can be detected in EEFC_FSR after a programming sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Flash Error: At the end of the programming, the EraseVerify or WriteVerify test of the Flash memory has failed.

The status of GPNVM bits can be returned by the EEFC. The sequence is the following:

1. Execute the ‘Get GPNVM Bit’ command by writing EEFC_FCR.FCMD with the GGPB command. Field EEFC_FCR.FARG is meaningless.
2. GPNVM bits can be read by the software application in EEFC_FRR. 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 EEFC_FRR return 0.

For example, if the third bit of the first word read in EEFC_FRR is set, the third GPNVM bit is active.

One error can be detected in EEFC_FSR after a programming sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.

Note: Access to the Flash in read is permitted when a ‘Set GPNVM Bit’, ‘Clear GPNVM Bit’ or ‘Get GPNVM Bit’ command is executed.

20.4.3.6 Calibration Bit

Calibration bits do not interfere with the embedded Flash memory plane.

The calibration bits cannot be modified.

The status of calibration bits are returned by the EEFC. The sequence is the following:

1. Execute the ‘Get CALIB Bit’ command by writing EEFC_FCR.FCMD with the GCALB command. Field EEFC_FCR.FARG is meaningless.
2. Calibration bits can be read by the software application in EEFC_FRR. The first word read corresponds to the first 32 calibration bits. The following reads provide the next 32 calibration bits as long as it is meaningful. Extra reads to EEFC_FRR return 0.

The 8/12 MHz internal RC oscillator is calibrated in production. This calibration can be read through the GCALB command. Table 20-5 shows the bit implementation.

The RC calibration for the 4 MHz is set to ‘1000000’.

Table 20-5. Calibration Bit Indexes

Description	EEFC_FRR Bits
8 MHz RC calibration output	[28–22]
12 MHz RC calibration output	[38–32]

20.4.3.7 Security Bit Protection

When the security bit is enabled, access to the Flash through the 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.

20.4.3.8 Unique Identifier Area

Each device is programmed with a 128-bit unique identifier area . See Figure 20-1 "Flash Memory Areas".

The sequence to read the unique identifier area is the following:

1. Execute the ‘Start Read Unique Identifier’ command by writing EEFC_FCR.FCMD with the STUI command. Field EEFC_FCR.FARG is meaningless.
2. Wait until the bit EEFC_FSR.FRDY falls to read the unique identifier area. The unique identifier field is located in the first 128 bits of the Flash memory mapping. The ‘Start Read Unique Identifier’ command reuses some addresses of the memory plane for code, but the unique identifier area is physically different from the memory plane for code.
3. To stop reading the unique identifier area, execute the ‘Stop Read Unique Identifier’ command by writing EEFC_FCR.FCMD with the SPUI command. Field EEFC_FCR.FARG is meaningless.
4. When the SPUI command has been executed, the bit EEFC_FSR.FRDY rises. If an interrupt was enabled by setting the bit EEFC_FMR.FRDY, the interrupt line of the interrupt controller is activated.

Note that during the sequence, the software cannot be fetched from the Flash.

20.4.3.9 User Signature Area

Each product contains a user signature area of 512-bytes. It can be used for storage. Read, write and erase of this area is allowed.

See Figure 20-1 "Flash Memory Areas".

The sequence to read the user signature area is the following:

1. Execute the ‘Start Read User Signature’ command by writing EEFC_FCR.FCMD with the STUS command. Field EEFC_FCR.FARG is meaningless.
2. Wait until the bit EEFC_FSR.FRDY falls to read the user signature area. The user signature area is located in the first 512 bytes of the Flash memory mapping. The ‘Start Read User Signature’ command reuses some addresses of the memory plane but the user signature area is physically different from the memory plane
3. To stop reading the user signature area, execute the ‘Stop Read User Signature’ command by writing EEFC_FCR.FCMD with the SPUS command. Field EEFC_FCR.FARG is meaningless.
4. When the SPUI command has been executed, the bit EEFC_FSR.FRDY rises. If an interrupt was enabled by setting the bit EEFC_FMR.FRDY, the interrupt line of the interrupt controller is activated.

Note that during the sequence, the software cannot be fetched from the Flash or from the second plane in case of dual plane.

One error can be detected in EEFC_FSR after this sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.

The sequence to write the user signature area is the following:

1. Write the full page, at any page address, within the internal memory area address space.
2. Execute the ‘Write User Signature’ command by writing EEFC_FCR.FCMD with the WUS command. Field EEFC_FCR.FARG is meaningless.
3. When programming is completed, the bit EEFC_FSR.FRDY rises. If an interrupt has been enabled by setting the bit EEFC_FMR.FRDY, the corresponding interrupt line of the interrupt controller is activated.

Two errors can be detected in EEFC_FSR after this sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Flash Error: At the end of the programming, the WriteVerify test of the Flash memory has failed.

The sequence to erase the user signature area is the following:

1. Execute the ‘Erase User Signature’ command by writing EEFC_FCR.FCMD with the EUS command. Field EEFC_FCR.FARG is meaningless.
2. When programming is completed, the bit EEFC_FSR.FRDY rises. If an interrupt has been enabled by setting the bit EEFC_FMR.FRDY, the corresponding interrupt line of the interrupt controller is activated.

Two errors can be detected in EEFC_FSR after this sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Flash Error: At the end of the programming, the EraseVerify test of the Flash memory has failed.

20.5 Enhanced Embedded Flash Controller (EEFC) User Interface

The User Interface of the Embedded Flash Controller (EEFC) is integrated within the System Controller with base address 0x400E0A00.

Table 20-6. Register Mapping

Offset	Register	Name	Access	Reset State
0x00	EEFC Flash Mode Register	EEFC_FMR	Read/Write	0x0400_0000
0x04	EEFC Flash Command Register	EEFC_FCR	Write-only	–
0x08	EEFC Flash Status Register	EEFC_FSR	Read-only	0x0000_0001
0x0C	EEFC Flash Result Register	EEFC_FRR	Read-only	0x0
0x10–0x14	Reserved	–	–	–
0x18–0xE4	Reserved	–	–	–

20.5.1 EEFC Flash Mode Register

Name: EEFC_FMR

Address: 0x400E0A00

Access: Read/Write

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

- **FRDY: Flash 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:

FWS = Number of cycles for Read/Write operations - 1

- **SCOD: Sequential Code Optimization Disable**

0: The sequential code optimization is enabled.

1: The sequential code optimization is disabled.

No Flash read should be done during change of this field.

- **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 field.

- **CLOE: Code Loop Optimization Enable**

0: The opcode loop optimization is disabled.

1: The opcode loop optimization is enabled.

No Flash read should be done during change of this field.

20.5.2 EEFC Flash Command Register

Name: EEFC_FCR

Address: 0x400E0A04

Access: Write-only

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**

Value	Name	Description
0x00	GETD	Get Flash descriptor
0x01	WP	Write page
0x02	WPL	Write page and lock
0x03	EWP	Erase page and write page
0x04	EWPL	Erase page and write page then lock
0x05	EA	Erase all
0x07	EPA	Erase pages
0x08	SLB	Set lock bit
0x09	CLB	Clear lock bit
0x0A	GLB	Get lock bit
0x0B	SGPB	Set GPNVM bit
0x0C	CGPB	Clear GPNVM bit
0x0D	GGPB	Get GPNVM bit
0x0E	STUI	Start read unique identifier
0x0F	SPUI	Stop read unique identifier
0x10	GCALB	Get CALIB bit
0x11	ES	Erase sector
0x12	WUS	Write user signature
0x13	EUS	Erase user signature
0x14	STUS	Start read user signature
0x15	SPUS	Stop read user signature

- **FARG: Flash Command Argument**

GETD, GLB, GGPB, STUI, SPUI, GCALB, WUS, EUS, STUS, SPUS, EA	Commands requiring no argument, including Erase all command	FARG is meaningless, must be written with 0
ES	Erase sector command	FARG must be written with any page number within the sector to be erased
EPA	Erase pages command	FARG[1:0] defines the number of pages to be erased The start page must be written in FARG[15:2]. FARG[1:0] = 0: Four pages to be erased. FARG[15:2] = Page_Number / 4 FARG[1:0] = 1: Eight pages to be erased. FARG[15:3] = Page_Number / 8, FARG[2]=0 FARG[1:0] = 2: Sixteen pages to be erased. FARG[15:4] = Page_Number / 16, FARG[3:2]=0 FARG[1:0] = 3: Thirty-two pages to be erased. FARG[15:5] = Page_Number / 32, FARG[4:2]=0 Refer to Table 20-4 “EEFC_FCR.FARG Field for EPA Command”.
WP, WPL, EWP, EWPL	Programming commands	FARG must be written with the page number to be programmed
SLB, CLB	Lock bit commands	FARG defines the page number to be locked or unlocked
SGPB, CGPB	GPNVM commands	FARG defines the GPNVM number to be programmed

- **FKEY: Flash Writing Protection Key**

Value	Name	Description
0x5A	PASSWD	The 0x5A value enables the command defined by the bits of the register. If the field is written with a different value, the write is not performed and no action is started.

20.5.3 EEFC Flash Status Register

Name: EEFC_FSR

Address: 0x400E0A08

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
–	–	–	–	FLERR	FLOCKE	FCMDE	FRDY

- **FRDY: Flash Ready Status (cleared when Flash is busy)**

0: The EEFC is busy.

1: The EEFC is ready to start a new command.

When set, this flag triggers an interrupt if the FRDY flag is set in EEFC_FMR.

This flag is automatically cleared when the EEFC is busy.

- **FCMDE: Flash Command Error Status (cleared on read or by writing EEFC_FCR)**

0: No invalid commands and no bad keywords were written in EEFC_FMR.

1: An invalid command and/or a bad keyword was/were written in EEFC_FMR.

- **FLOCKE: Flash Lock Error Status (cleared on read)**

0: No programming/erase of at least one locked region has happened since the last read of EEFC_FSR.

1: Programming/erase of at least one locked region has happened since the last read of EEFC_FSR.

This flag is automatically cleared when EEFC_FSR is read or EEFC_FCR is written.

- **FLERR: Flash Error Status (cleared when a programming operation starts)**

0: No Flash memory error occurred at the end of programming (EraseVerify or WriteVerify test has passed).

1: A Flash memory error occurred at the end of programming (EraseVerify or WriteVerify test has failed).

20.5.4 EEFC Flash Result Register

Name: EEFC_FRR

Address: 0x400E0A0C

Access: Read-only

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, the next resulting value is accessible at the next register read.

21. Fast Flash Programming Interface (FFPI)

21.1 Description

The Fast Flash Programming Interface (FFPI) provides parallel high-volume programming using a standard gang programmer. The parallel interface is fully handshaked and the device is considered to be a standard EEPROM. Additionally, the parallel protocol offers an optimized access to all the embedded Flash functionalities.

Although the Fast Flash Programming mode is a dedicated mode for high volume programming, this mode is not designed for in-situ programming.

21.2 Embedded Characteristics

- Programming Mode for High-volume Flash Programming Using Gang Programmer
  - Offers Read and Write Access to the Flash Memory Plane
  - Enables Control of Lock Bits and General-purpose NVM Bits
  - Enables Security Bit Activation
  - Disabled Once Security Bit is Set
- Parallel Fast Flash Programming Interface
  - Provides an 16-bit Parallel Interface to Program the Embedded Flash
  - Full Handshake Protocol

21.3 Parallel Fast Flash Programming

21.3.1 Device Configuration

In Fast Flash Programming mode, the device is in a specific test mode. Only a certain set of pins is significant. The rest of the PIOs are used as inputs with a pull-up. The crystal oscillator is in bypass mode. Other pins must be left unconnected.

Figure 21-1. 16-bit Parallel Programming Interface

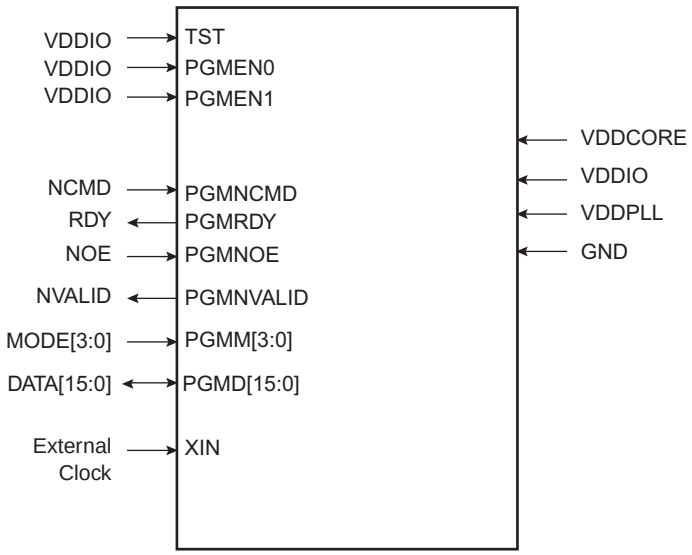


Table 21-1. Signal Description List

Signal Name	Function	Type	Active Level	Comments
Power				
VDDIO	I/O Lines Power Supply	Power	–	–
VDDCORE	Core Power Supply	Power	–	–
VDDPLL	PLL Power Supply	Power	–	–
GND	Ground	Ground	–	–
Clocks				
XIN	Main Clock Input	Input	–	–
Test				
TST	Test Mode Select	Input	High	Must be connected to VDDIO
PGMEN0	Test Mode Select	Input	High	Must be connected to VDDIO
PGMEN1	Test Mode Select	Input	High	Must be connected to VDDIO
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

Table 21-1. Signal Description List (Continued)

Signal Name	Function	Type	Active Level	Comments
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 21-2)	Input	–	Pulled-up input at reset
PGMD[15:0]	Bi-directional data bus	Input/Output	–	Pulled-up input at reset

21.3.2 Signal Names

Depending on the MODE settings, DATA is latched in different internal registers.

Table 21-2. Mode Coding

MODE[3:0]	Symbol	Data
0000	CMDE	Command Register
0001	ADDR0	Address Register LSBs
0010	ADDR1	–
0011	ADDR2	–
0100	ADDR3	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 21-3. Command Bit Coding

DATA[15:0]	Symbol	Command Executed
0x0011	READ	Read Flash
0x0012	WP	Write Page Flash
0x0022	WPL	Write Page and Lock Flash
0x0032	EWP	Erase Page and Write Page
0x0042	EWPL	Erase Page and Write Page then Lock
0x0013	EA	Erase All
0x0014	SLB	Set Lock Bit
0x0024	CLB	Clear Lock Bit
0x0015	GLB	Get Lock Bit
0x0034	SGPB	Set General Purpose NVM bit
0x0044	CGPB	Clear General Purpose NVM bit
0x0025	GGPB	Get General Purpose NVM bit
0x0054	SSE	Set Security Bit
0x0035	GSE	Get Security Bit
0x001F	WRAM	Write Memory
0x001E	GVE	Get Version

21.3.3 Entering Parallel Programming Mode

The following algorithm puts the device in Parallel Programming mode:

1. Apply the supplies as described in Table 21-1.
2. If an external clock is available, apply it to XIN within the VDDCORE POR reset time-out period, as defined in the section “Electrical Characteristics”.
3. Wait for the end of this reset period.
4. Start a read or write handshaking.

21.3.4 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 completed once the NCMD signal is high and RDY is high.

21.3.4.1 Write Handshaking

For details on the write handshaking sequence, refer to Figure 21-2 and Table 21-4.

Figure 21-2. Parallel Programming Timing, Write Sequence

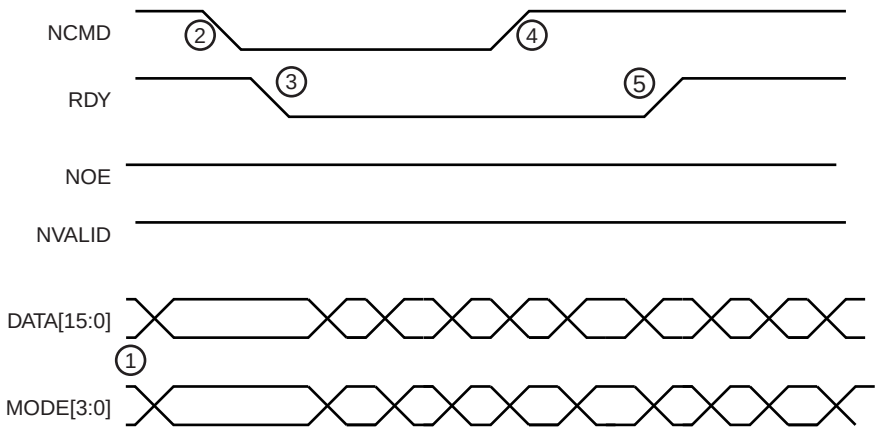


Table 21-4. Write Handshake

Step	Programmer Action	Device Action	Data I/O
1	Sets MODE and DATA signals	Waits for NCMD low	Input
2	Clears NCMD signal	Latches MODE and DATA	Input
3	Waits for RDY low	Clears RDY signal	Input
4	Releases MODE and DATA signals	Executes command and polls NCMD high	Input
5	Sets NCMD signal	Executes command and polls NCMD high	Input
6	Waits for RDY high	Sets RDY	Input

21.3.4.2 Read Handshaking

For details on the read handshaking sequence, refer to Figure 21-3 and Table 21-5.

Figure 21-3. Parallel Programming Timing, Read Sequence

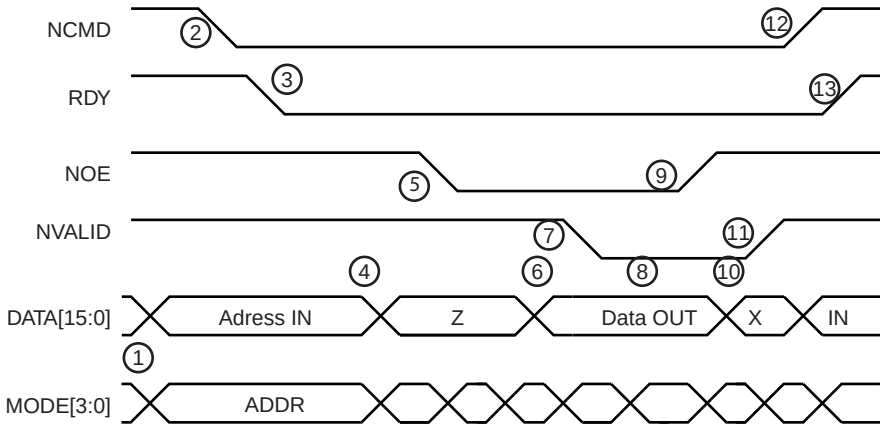


Table 21-5. Read Handshake

Step	Programmer Action	Device Action	DATA I/O
1	Sets MODE and DATA signals	Waits for NCMD low	Input
2	Clears NCMD signal	Latch MODE and DATA	Input
3	Waits for RDY low	Clears RDY signal	Input
4	Sets DATA signal in tristate	Waits for NOE Low	Input
5	Clears NOE signal	–	Tristate
6	Waits for NVALID low	Sets DATA bus in output mode and outputs the flash contents.	Output
7	–	Clears NVALID signal	Output
8	Reads value on DATA Bus	Waits for NOE high	Output
9	Sets NOE signal	–	Output
10	Waits for NVALID high	Sets DATA bus in input mode	X
11	Sets DATA in output mode	Sets NVALID signal	Input
12	Sets NCMD signal	Waits for NCMD high	Input
13	Waits for RDY high	Sets RDY signal	Input

21.3.5 Device Operations

Several commands on the Flash memory are available. These commands are summarized in Table 21-3. 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.

21.3.5.1 Flash Read Command

This command is used to read the contents of the Flash memory. The read command can start at any valid address in the memory plane and is optimized for consecutive reads. Read handshaking can be chained; an internal address buffer is automatically increased.

Table 21-6. Read Command

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	READ
2	Write handshaking	ADDR0	Memory Address LSB
3	Write handshaking	ADDR1	Memory Address
4	Read handshaking	DATA	*Memory Address++
5	Read handshaking	DATA	*Memory Address++
...	...	...	...
n	Write handshaking	ADDR0	Memory Address LSB
n+1	Write handshaking	ADDR1	Memory Address
n+2	Read handshaking	DATA	*Memory Address++
n+3	Read handshaking	DATA	*Memory Address++
...	...	...	...

21.3.5.2 Flash Write Command

This command is used to write the Flash contents.

The Flash memory plane is organized into several pages. Data to be written are stored in a load buffer that corresponds to a Flash memory page. The load buffer is automatically flushed to the Flash:

- before access to any page other than the current one
- when a new command is validated (MODE = CMDE)

The **Write Page** command **(WP)** is optimized for consecutive writes. Write handshaking can be chained; an internal address buffer is automatically increased.

Table 21-7. 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.

21.3.5.3 Flash Full Erase Command

This command is used to erase the Flash memory planes.

All lock regions must be unlocked before the Full Erase command by using the CLB command. Otherwise, the erase command is aborted and no page is erased.

Table 21-8. Full Erase Command

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	EA
2	Write handshaking	DATA	0

21.3.5.4 Flash Lock Commands

Lock bits can be set using WPL or EWPL commands. They can also be set by using the **Set Lock** command **(SLB)**. With this command, several lock bits can be activated. A Bit Mask is provided as argument to the command. When bit 0 of the bit mask is set, then the first lock bit is activated.

In the same way, the **Clear Lock** command **(CLB)** is used to clear lock bits.

Table 21-9. Set and Clear Lock Bit Command

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	SLB or CLB
2	Write handshaking	DATA	Bit Mask

Lock bits can be read using **Get Lock Bit** command **(GLB)**. The n^{th} lock bit is active when the bit n of the bit mask is set.

Table 21-10. 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

21.3.5.5 Flash General-purpose NVM Commands

General-purpose NVM bits (GP NVM bits) can be set using the **Set GPNVM** command **(SGPB)**. This command also activates GP NVM bits. A bit mask is provided as argument to the command. When bit 0 of the bit mask is set, then the first GP NVM bit is activated.

In the same way, the **Clear GPNVM** command **(CGPB)** is used to clear general-purpose NVM bits. The general-purpose NVM bit is deactivated when the corresponding bit in the pattern value is set to 1.

Table 21-11. Set/Clear GP NVM Command

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	SGPB or CGPB
2	Write handshaking	DATA	GP NVM bit pattern value

General-purpose NVM bits can be read using the **Get GPNVM Bit** command **(GGPB)**. The n^{th} GP NVM bit is active when bit n of the bit mask is set.

Table 21-12. 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

21.3.5.6 Flash Security Bit Command

A security bit can be set using the **Set Security Bit** command (SSE). Once the security bit is active, the Fast Flash programming is disabled. No other command can be run. An event on the Erase pin can erase the security bit once the contents of the Flash have been erased.

Table 21-13. 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.

To erase the Flash, perform the following steps:

1. Power-off the chip.
2. Power-on the chip with TST = 0.
3. Assert the ERASE pin for at least the ERASE pin assertion time as defined in the section “Electrical Characteristics”.
4. Power-off the chip.

Return to FFPI mode to check that the Flash is erased.

21.3.5.7 Memory Write Command

This command is used to perform a write access to any memory location.

The **Memory Write** command **(WRAM)** is optimized for consecutive writes. Write handshaking can be chained; an internal address buffer is automatically increased.

Table 21-14. 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++
...	...	...	...

21.3.5.8 Get Version Command

The **Get Version** (GVE) command retrieves the version of the FFPI interface.

Table 21-15. Get Version Command

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	GVE
2	Read handshaking	DATA	Version

22. Cortex-M Cache Controller (CMCC)

22.1 Description

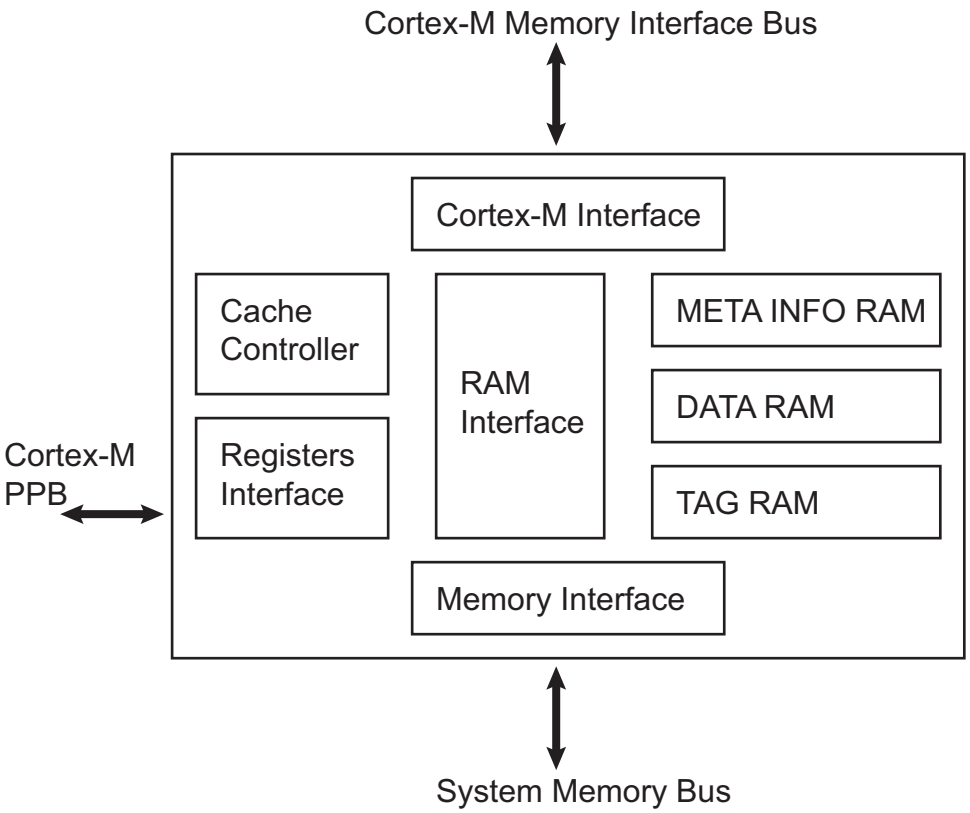
The Cortex-M Cache Controller (CMCC) is a 4-Way set associative unified cache controller. It integrates a controller, a tag directory, data memory, metadata memory and a configuration interface.

22.2 Embedded Characteristics

- Physically addressed and physically tagged
- L1 data cache set to 2 Kbytes
- L1 cache line size set to 16 Bytes
- L1 cache integrates 32 bus master interface
- Unified direct mapped cache architecture
- Unified 4-Way set associative cache architecture
- Write accesses forwarded, cache state not modified. Allocate on read.
- Round Robin victim selection policy
- Event Monitoring, with one programmable 32-bit counter
- Configuration registers accessible through Cortex-M Private Peripheral Bus (PPB)
- Cache interface includes cache maintenance operations registers

22.3 Block Diagram

Figure 22-1. Block Diagram



22.4 Functional Description

22.4.1 Cache Operation

On reset, the cache controller data entries are all invalidated and the cache is disabled. The cache is transparent to processor operations. The cache controller is activated with its configuration registers. The configuration interface is memory-mapped in the private peripheral bus.

Use the following sequence to enable the cache controller:

1. Verify that the cache controller is disabled by reading the value of the CSTS (Cache Controller Status) bit of the Status register (CMCC_SR).
2. Enable the cache controller by writing a one to the CEN (Cache Enable) bit of the Control register (CMCC_CTRL).

22.4.2 Cache Maintenance

If the contents seen by the cache have changed, the user must invalidate the cache entries. This can be done line-by-line or for all cache entries.

22.4.2.1 Cache Invalidate-by-Line Operation

When an invalidate-by-line command is issued, the cache controller resets the valid bit information of the decoded cache line. As the line is no longer valid, the replacement counter points to that line.

Use the following sequence to invalidate one line of cache:

1. Disable the cache controller by clearing the CEN bit of CMCC_CTRL.
2. Check the CSTS bit of CMCC_SR to verify that the cache is successfully disabled.
3. Perform an invalidate-by-line by configuring the bits INDEX and WAY in the Maintenance Register 1 (CMCC_MAINT1).
4. Enable the cache controller by writing a one the CEN bit of the CMCC_CTRL.

22.4.2.2 Cache Invalidate All Operation

To invalidate all cache entries, write a one to the INVALL bit of the Maintenance Register 0 (CMCC_MAINT0).

22.4.3 Cache Performance Monitoring

The Cortex-M cache controller includes a programmable 32-bit monitor counter. The monitor can be configured to count the number of clock cycles, the number of data hits or the number of instruction hits.

Use the following sequence to activate the counter:

1. Configure the monitor counter by writing to the MODE field of the Monitor Configuration register (CMCC_MCFG).
2. Enable the counter by writing a one to the MENABLE bit of the Monitor Enable register (CMCC_MEN).
3. If required, clear the counter by writing a one to the SWRST bit of the Monitor Control register (CMCC_MCTRL).
4. Check the value of the monitor counter by reading the EVENT_CNT field of the CMCC_MSR.

22.5 Cortex-M Cache Controller (CMCC) User Interface

Table 22-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Cache Controller Type Register	CMCC_TYPE	Read-only	–
0x04	Cache Controller Configuration Register	CMCC_CFG	Read/Write	0x00000000
0x08	Cache Controller Control Register	CMCC_CTRL	Write-only	–
0x0C	Cache Controller Status Register	CMCC_SR	Read-only	0x00000001
0x10–0x1C	Reserved	–	–	–
0x20	Cache Controller Maintenance Register 0	CMCC_MAINT0	Write-only	–
0x24	Cache Controller Maintenance Register 1	CMCC_MAINT1	Write-only	–
0x28	Cache Controller Monitor Configuration Register	CMCC_MCFG	Read/Write	0x00000000
0x2C	Cache Controller Monitor Enable Register	CMCC_MEN	Read/Write	0x00000000
0x30	Cache Controller Monitor Control Register	CMCC_MCTRL	Write-only	–
0x34	Cache Controller Monitor Status Register	CMCC_MSR	Read-only	0x00000000
0x38–0xFC	Reserved	–	–	–

22.5.1 Cache Controller Type Register

Name: CMCC_TYPE

Address: 0x400C4000

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–		CLSIZE			CSIZE		

7	6	5	4	3	2	1	0
LCKDOWN	WAYNUM		RRP	LRUP	RANDP	GCLK	AP

- **AP: Access Port Access Allowed**

0: Access Port Access is disabled.

1: Access Port Access is enabled.

- **GCLK: Dynamic Clock Gating Supported**

0: Cache controller does not support clock gating.

1: Cache controller uses dynamic clock gating.

- **RANDP: Random Selection Policy Supported**

0: Random victim selection is not supported.

1: Random victim selection is supported.

- **LRUP: Least Recently Used Policy Supported**

0: Least Recently Used Policy is not supported.

1: Least Recently Used Policy is supported.

- **RRP: Random Selection Policy Supported**

0: Random Selection Policy is not supported.

1: Random Selection Policy is supported.

- **WAYNUM: Number of Ways**

Value	Name	Description
0	DMAPPED	Direct Mapped Cache
1	ARCH2WAY	2-way set associative
2	ARCH4WAY	4-way set associative
3	ARCH8WAY	8-way set associative

- **LCKDOWN: Lockdown Supported**

0: Lockdown is not supported.

1: Lockdown is supported.

- **CSIZE: Data Cache Size**

Value	Name	Description
0	CSIZE_1KB	Data cache size is 1 Kbyte
1	CSIZE_2KB	Data cache size is 2 Kbytes
2	CSIZE_4KB	Data cache size is 4 Kbytes
3	CSIZE_8KB	Data cache size is 8 Kbytes

- **CLSIZE: Cache LIne Size**

Value	Name	Description
0	CLSIZE_1KB	Cache line size is 4 bytes
1	CLSIZE_2KB	Cache line size is 8 bytes
2	CLSIZE_4KB	Cache line size is 16 bytes
3	CLSIZE_8KB	Cache line size is 32 bytes

22.5.2 Cache Controller Configuration Register

Name: CMCC_CFG

Address: 0x400C4004

Access: Read/Write

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

- **GCLKDIS: Disable Clock Gating**

0: Clock gating is activated.

1: Clock gating is disabled.

22.5.3 Cache Controller Control Register

Name: CMCC_CTRL

Address: 0x400C4008

Access: Write-only

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

- **CEN: Cache Controller Enable**

0: The cache controller is disabled.

1: The cache controller is enabled.

22.5.4 Cache Controller Status Register

Name: CMCC_SR

Address: 0x400C400C

Access: Read-only

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

- **CSTS: Cache Controller Status**

0: The cache controller is disabled.

1: The cache controller is enabled.

22.5.5 Cache Controller Maintenance Register 0

Name: CMCC_MAINT0

Address: 0x400C4020

Access: Write-only

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

- **INVALL: Cache Controller Invalidate All**

0: No effect.

1: All cache entries are invalidated.

22.5.6 Cache Controller Maintenance Register 1

Name: CMCC_MAINT1

Address: 0x400C4024

Access: Write-only

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

- **INDEX: Invalidate Index**

This field indicates the cache line that is being invalidated.

The size of the INDEX field depends on the cache size:

For example:

– for 2 Kbytes: 5 bits

– for 4 Kbytes: 6 bits

– for 8 Kbytes: 7 bits

- **WAY: Invalidate Way**

Value	Name	Description
0	WAY0	Way 0 is selection for index invalidation
1	WAY1	Way 1 is selection for index invalidation
2	WAY2	Way 2 is selection for index invalidation
3	WAY3	Way 3 is selection for index invalidation

22.5.7 Cache Controller Monitor Configuration Register

Name: CMCC_MCFG

Address: 0x400C4028

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
–	–	–	–	–	–	MODE	

- **MODE: Cache Controller Monitor Counter Mode**

Value	Name	Description
0	CYCLE_COUNT	Cycle counter
1	IHIT_COUNT	Instruction hit counter
2	DHIT_COUNT	Data hit counter

22.5.8 Cache Controller Monitor Enable Register

Name: CMCC_MEN

Address: 0x400C402C

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
–	–	–	–	–	–	–	MENABLE

- **MENABLE: Cache Controller Monitor Enable**

0: The monitor counter is disabled.

1: The monitor counter is enabled.

22.5.9 Cache Controller Monitor Control Register

Name: CMCC_MCTRL

Address: 0x400C4030

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: Monitor**

0: No effect.

1: Resets the event counter register.

22.5.10 Cache Controller Monitor Status Register

Name: CMCC_MSR

Address: 0x400C4034

Access: Read-only

31	30	29	28	27	26	25	24
EVENT_CNT							

23	22	21	20	19	18	17	16
EVENT_CNT							

15	14	13	12	11	10	9	8
EVENT_CNT							

7	6	5	4	3	2	1	0
EVENT_CNT							

- **EVENT_CNT: Monitor Event Counter**

23. SAM-BA Boot Program for SAM4E Microcontrollers

23.1 Description

The SAM-BA Boot Program integrates an array of programs permitting download and/or upload into the different memories of the product.

23.2 Embedded Characteristics

- Default Boot Program Stored in SAM4E Series Products
- Interface with SAM-BA Graphic User Interface
- SAM-BA Boot
  - Supports Several Communication Media
    - Serial Communication on UART0
    - USB Device Port Communication up to 1M Byte/s
  - USB Requirements
    - External Crystal or Clock with the frequency of:
      11.289 MHz
      12.000 MHz
      16.000 MHz
      18.432 MHz

23.3 Hardware and Software Constraints

- SAM-BA Boot uses the first 2048 bytes of the SRAM for variables and stacks. The remaining available size can be used for user's code.
- USB Requirements:
  - External Crystal or External Clock(1) with frequency of:
    - 11.289 MHz
    - 12.000 MHz
    - 16.000 MHz
    - 18.432 MHz
- UART0 requirements: None

Note: 1. must be 2500 ppm and 1.8V Square Wave Signal.

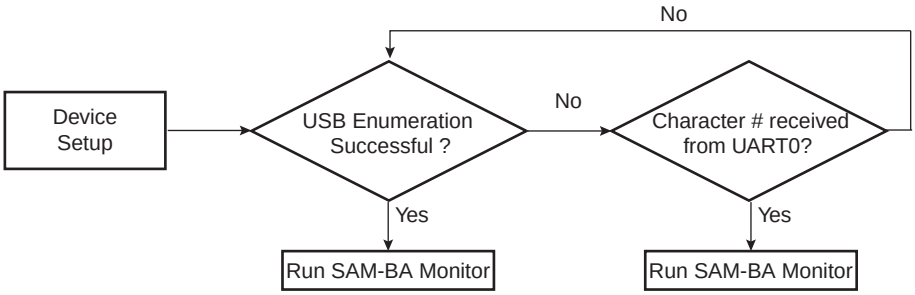
Table 23-1. Pins Driven during Boot Program Execution

Peripheral	Pin	PIO Line
UART0	URXD0	PA9
UART0	UTXD0	PA10

23.4 Flow Diagram

The Boot Program implements the algorithm illustrated in Figure 23-1.

Figure 23-1. Boot Program Algorithm Flow Diagram



The SAM-BA Boot program seeks to detect a source clock either from the embedded main oscillator with external crystal (main oscillator enabled) or from a supported frequency signal applied to the XIN pin (Main oscillator in bypass mode).

If a clock is found from the two possible sources above, the boot program checks to verify that the frequency is one of the supported external frequencies. If the frequency is one of the supported external frequencies, USB activation is allowed, else (no clock or frequency other than one of the supported external frequencies), the internal 12 MHz RC oscillator is used as main clock and USB clock is not allowed due to frequency drift of the 12 MHz RC oscillator.

23.5 Device Initialization

The initialization sequence is the following:

1. Stack setup
2. Set up the Embedded Flash Controller
3. External Clock detection (crystal or external clock on XIN)
4. If external crystal or clock with supported frequency, allow USB activation
5. Else, does not allow USB activation and use internal 12 MHz RC oscillator
6. Main oscillator frequency detection if no external clock detected
7. Switch Master Clock on Main Oscillator
8. C variable initialization
9. PLLA setup: PLLA is initialized to generate a 48 MHz clock
10. Disable of the Watchdog
11. Initialization of UART0 (115200 bauds, 8, N, 1)
12. Initialization of the USB Device Port (in case of USB activation allowed)
13. Wait for one of the following events:
    1. Check if USB device enumeration has occurred
    2. Check if characters have been received in UART0
14. Jump to SAM-BA Monitor (see Section 23.6 ”SAM-BA Monitor”)

23.6 SAM-BA Monitor

Once the communication interface is identified, the monitor runs in an infinite loop waiting for different commands as shown in Table 23-2.

Table 23-2. 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
- Read commands: Read a byte (**o**), a halfword (**h**) or a word (**w**) from the target
  - *Address*: Address in hexadecimal
  - *Output*: The byte, halfword or word read in hexadecimal
- Send a file (**S**): Send a file to a specified address
  - *Address*: Address in hexadecimal

Note: There is a time-out on this command which is reached when the prompt '>' appears before the end of the command execution.

- Receive a file (**R**): Receive data into a file from a specified address
  - *Address*: Address in hexadecimal
  - *NbOfBytes*: Number of bytes in hexadecimal to receive
- Go (**G**): Jump to a specified address and execute the code
  - *Address*: Address to jump in hexadecimal
- Get Version (**V**): Return the SAM-BA boot version

Note: In Terminal mode, when the requested command is performed, SAM-BA Monitor adds the following prompt sequence to its answer: <LF>+<CR>+'>'.

23.6.1 UART0 Serial Port

Communication is performed through the UART0 initialized to 115200 Baud, 8, n, 1.

The Send and Receive File commands use the Xmodem protocol to communicate. Any terminal performing this protocol can be used to send the application file to the target. The size of the binary file to send depends on the SRAM size embedded in the product. In all cases, the size of the binary file must be lower than the SRAM size because the Xmodem protocol requires some SRAM memory to work. See Section 23.3 ”Hardware and Software Constraints”.

23.6.2 Xmodem Protocol

The Xmodem protocol supported is the 128-byte length block. This protocol uses a two-character CRC-16 to guarantee detection of a maximum bit error.

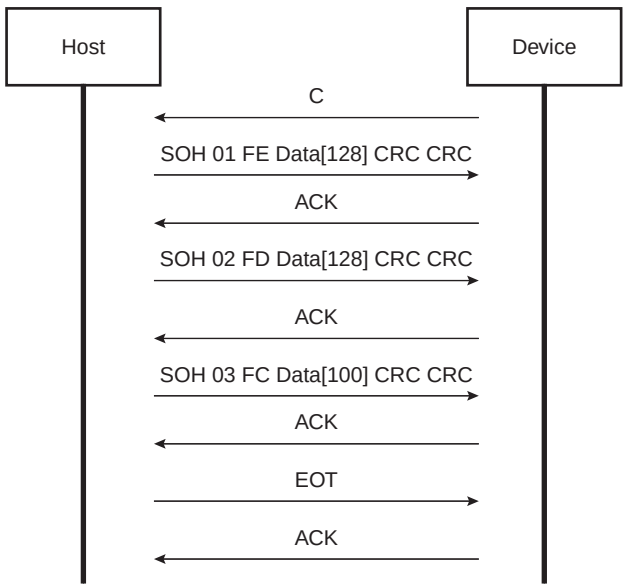
Xmodem protocol with CRC is accurate provided both sender and receiver report successful transmission. Each block of the transfer looks like:

<SOH><blk #><255-blk #><--128 data bytes--><checksum> in which:

- <SOH> = 01 hex
- <blk #> = binary number, starts at 01, increments by 1, and wraps 0FFH to 00H (not to 01)
- <255-blk #> = 1’s complement of the blk#.
- <checksum> = 2 bytes CRC16

Figure 23-2 shows a transmission using this protocol.

Figure 23-2. Xmodem Transfer Example



23.6.3 USB Device Port

The device uses the USB communication device class (CDC) drivers to take advantage of the installed PC RS-232 software to talk over the USB. The CDC class is implemented in all releases of Windows®, 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 “USB Basic Application”, Atmel literature number 6123, for more details.

23.6.3.1 Enumeration Process

The USB protocol is a master/slave protocol. This is the host that starts the enumeration sending requests to the device through the control endpoint. The device handles standard requests as defined in the USB Specification.

Table 23-3. Handled Standard Requests

Request	Definition
GET_DESCRIPTOR	Returns the current device configuration value.
SET_ADDRESS	Sets the device address for all future device access.
SET_CONFIGURATION	Sets the device configuration.
GET_CONFIGURATION	Returns the current device configuration value.
GET_STATUS	Returns status for the specified recipient.
SET_FEATURE	Set or Enable a specific feature.
CLEAR_FEATURE	Clear or Disable a specific feature.

The device also handles some class requests defined in the CDC class.

Table 23-4. Handled Class Requests

Request	Definition
SET_LINE_CODING	Configures DTE rate, stop bits, parity and number of character bits.
GET_LINE_CODING	Requests current DTE rate, stop bits, parity and number of character bits.
SET_CONTROL_LINE_STATE	RS-232 signal used to tell the DCE device the DTE device is now present.

Unhandled requests are STALLed.

23.6.3.2 Communication Endpoints

There are two communication endpoints and endpoint 0 is used for the enumeration process. Endpoint 1 is a 64-byte Bulk OUT endpoint and endpoint 2 is a 64-byte Bulk IN endpoint. SAM-BA Boot commands are sent by the host through endpoint 1. If required, the message is split by the host into several data payloads by the host driver.

If the command requires a response, the host can send IN transactions to pick up the response.

23.6.4 In Application Programming (IAP) Feature

The IAP feature is a function located in ROM that can be called by any software application.

When called, this function sends the desired FLASH command to the EEFC and waits for the Flash to be ready (looping while the FRDY bit is not set in the 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 (0x00800008).

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);

}
```

24. Bus Matrix (MATRIX)

24.1 Description

The Bus Matrix (MATRIX) implements a multi-layer AHB, based on the AHB-Lite protocol, that enables parallel access paths between multiple AHB masters and slaves in a system, thus increasing the overall bandwidth. The Bus Matrix interconnects up to 7 AHB masters to up to 6 AHB slaves. The normal latency to connect a master to a slave is one cycle except for the default master of the accessed slave which is connected directly (zero cycle latency). The Bus Matrix user interface is compliant with ARM Advanced Peripheral Bus.

24.2 Embedded Characteristics

- Configurable Number of Masters (up to 7)
- Configurable Number of Slaves (up to 6)
- One Decoder for Each Master
- Several Possible Boot Memories for Each Master before Remap
- One Remap Function for Each Master
- Support for Long Bursts of 32, 64, 128 and up to the 256-beat Word Burst AHB Limit
- Enhanced Programmable Mixed Arbitration for Each Slave
  - Round-Robin
  - Fixed Priority
- Programmable Default Master for Each Slave
  - No Default Master
  - Last Accessed Default Master
  - Fixed Default Master
- Deterministic Maximum Access Latency for Masters
- Zero or One Cycle Arbitration Latency for the First Access of a Burst
- Bus Lock Forwarding to Slaves
- Master Number Forwarding to Slaves
- One Special Function Register for Each Slave (not dedicated)
- Write Protection of User Interface Registers

24.2.1 Matrix Masters

The Bus Matrix manages 7 masters, which means that each master can perform an access concurrently with others, to an available slave.

Each master has its own decoder, which is defined specifically for each master. In order to simplify the addressing, all the masters have the same decodings.

24.2.2 Matrix Slaves

The Bus Matrix manages 6 slaves. Each slave has its own arbiter, allowing a different arbitration per slave.

Table 24-1. List of Bus Matrix Slaves

Slave 0	Internal SRAM
Slave 1	Internal ROM
Slave 2	Internal Flash

Table 24-1. List of Bus Matrix Slaves

Slave 3	Peripheral Bridge 0
Slave 4	Peripheral Bridge 1
Slave 5	External Bus Interface (EBI)

24.2.3 Master to Slave Access

All the Masters can normally access all the Slaves. However, some paths do not make sense, for example allowing access from the Cortex-M4 S Bus to the Internal SRAM. Thus, these paths are forbidden or simply not wired, and shown as "–" in Table 24-2.

Table 24-2. Master to Slave Access

		0	1	2	3	4	5	6
Slaves	**Masters**	**Cortex-M4 I/D Bus**	**Cortex-M4 S Bus**	**PDC0**	**PDC1**	**DMAC**	**Reserved**	**EMAC**
0	Internal SRAM	–	X	X	X	X	–	X
1	Internal ROM	X	–	X	X	X	–	X
2	Internal Flash	X	–	–	–	–	–	–
3	Peripheral Bridge 0	–	X	X	–	X	–	–
4	Peripheral Bridge 1	–	X	–	X	X	–	–
5	External Bus Interface (EBI)	–	X	X	X	X	–	X

24.3 Memory Mapping

The Bus Matrix provides one decoder for every AHB master interface. The decoder offers each AHB master several memory mappings. Each memory area may be assigned to several slaves. Booting at the same address while using different AHB slaves (i.e. external RAM, internal ROM or internal Flash, etc.) becomes possible.

The Bus Matrix user interface provides the Master Remap Control Register (MATRIX_MRCR), that performs remap action for every master independently.

The Bus Matrix user interface provides Master Remap Control Register (MATRIX_MRCR) that performs remap action for every master independently.

24.4 Special Bus Granting Mechanism

The Bus Matrix provides some speculative bus granting techniques in order to anticipate access requests from masters. This mechanism reduces latency at first access of a burst, or for a single transfer, as long as the slave is free from any other master access. It does not provide any benefit if the slave is continuously accessed by more than one master, since arbitration is pipelined and has no negative effect on the slave bandwidth or access latency.

This bus granting mechanism sets a different default master for every slave.

At the end of the current access, if no other request is pending, the slave remains connected to its associated default master. A slave can be associated with three kinds of default masters:

- No default master
- Last access master
- Fixed default master

To change from one type of default master to another, the Bus Matrix user interface provides the Slave Configuration Registers, one for every slave, that set a default master for each slave. The Slave Configuration Register contains two fields: DEFMSTR_TYPE and FIXED_DEFMSTR. The 2-bit DEFMSTR_TYPE field selects the default master type (no default, last access master, fixed default master), whereas the 4-bit FIXED_DEFMSTR field selects a fixed default master provided that DEFMSTR_TYPE is set to fixed default master. Please refer to Section 24.12.2 “Bus Matrix Slave Configuration Registers”.

24.5 No Default Master

After the end of the current access, if no other request is pending, the slave is disconnected from all masters.

This configuration incurs one latency clock cycle for the first access of a burst after bus Idle. Arbitration without default master may be used for masters that perform significant bursts or several transfers with no Idle in between, or if the slave bus bandwidth is widely used by one or more masters.

This configuration provides no benefit on access latency or bandwidth when reaching maximum slave bus throughput whatever the number of requesting masters.

24.6 Last Access Master

After the end of the current access, if no other request is pending, the slave remains connected to the last master that performed an access request.

This allows the Bus Matrix to remove the one latency cycle for the last master that accessed the slave. Other non privileged masters still get one latency clock cycle if they want to access the same slave. This technique is useful for masters that mainly perform single accesses or short bursts with some Idle cycles in between.

This configuration provides no benefit on access latency or bandwidth when reaching maximum slave bus throughput whatever is the number of requesting masters.

24.7 Fixed Default Master

After the end of the current access, if no other request is pending, the slave connects to its fixed default master. Unlike the last access master, the fixed default master does not change unless the user modifies it by software (FIXED_DEFMSTR field of the related MATRIX_SCFG).

This allows the Bus Matrix arbiters to remove the one latency clock cycle for the fixed default master of the slave. All requests attempted by the fixed default master do not cause any arbitration latency, whereas other non-privileged masters will get one latency cycle. This technique is useful for a master that mainly performs single accesses or short bursts with Idle cycles in between.

This configuration provides no benefit on access latency or bandwidth when reaching maximum slave bus throughput, regardless of the number of requesting masters.

24.8 Arbitration

The Bus Matrix provides an arbitration mechanism that reduces latency when conflict cases occur, i.e. when two or more masters try to access the same slave at the same time. One arbiter per AHB slave is provided, thus arbitrating each slave specifically.

The Bus Matrix provides the user with the possibility of choosing between 2 arbitration types or mixing them for each slave:

1. Round-robin Arbitration (default)
2. Fixed Priority Arbitration

The resulting algorithm may be complemented by selecting a default master configuration for each slave.

When re-arbitration must be done, specific conditions apply. See Section 24.8.1 “Arbitration Scheduling”.

24.8.1 Arbitration Scheduling

Each arbiter has the ability to arbitrate between two or more different master requests. In order to avoid burst breaking and also to provide the maximum throughput for slave interfaces, arbitration may only take place during the following cycles:

1. Idle Cycles: When a slave is not connected to any master or is connected to a master which is not currently accessing it.
2. Single Cycles: When a slave is currently doing a single access.
3. End of Burst Cycles: When the current cycle is the last cycle of a burst transfer. For defined length burst, predicted end of burst matches the size of the transfer but is managed differently for undefined length burst. See Section 24.8.1.1 “Undefined Length Burst Arbitration”
4. Slot Cycle Limit: When the slot cycle counter has reached the limit value indicating that the current master access is too long and must be broken. See Section 24.8.1.2 “Slot Cycle Limit Arbitration”

24.8.1.1 Undefined Length Burst Arbitration

In order to prevent long AHB burst lengths that can lock the access to the slave for an excessive period of time, the user can trigger the re-arbitration before the end of the incremental bursts. The re-arbitration period can be selected from the following Undefined Length Burst Type (ULBT) possibilities:

1. Unlimited: no predetermined end of burst is generated. This value enables 1-Kbyte burst lengths.
2. 1-beat bursts: predetermined end of burst is generated at each single transfer during the INCR transfer.
3. 4-beat bursts: predetermined end of burst is generated at the end of each 4-beat boundary during INCR transfer.
4. 8-beat bursts: predetermined end of burst is generated at the end of each 8-beat boundary during INCR transfer.
5. 16-beat bursts: predetermined end of burst is generated at the end of each 16-beat boundary during INCR transfer.
6. 32-beat bursts: predetermined end of burst is generated at the end of each 32-beat boundary during INCR transfer.
7. 64-beat bursts: predetermined end of burst is generated at the end of each 64-beat boundary during INCR transfer.
8. 128-beat bursts: predetermined end of burst is generated at the end of each 128-beat boundary during INCR transfer.

The use of undefined length16-beat bursts, or less, is discouraged since this generally decreases significantly the overall bus bandwidth due to arbitration and slave latencies at each first access of a burst.

If the master does not permanently and continuously request the same slave or has an intrinsically limited average throughput, the ULBT should be left at its default unlimited value, knowing that the AHB specification natively limits all word bursts to 256 beats and double-word bursts to 128 beats because of its 1 Kilobyte address boundaries.

Unless duly needed, the ULBT should be left at its default value of 0 for power saving.

This selection can be done through the ULBT field of the Master Configuration Registers (MATRIX_MCFG).

24.8.1.2 Slot Cycle Limit Arbitration

The Bus Matrix contains specific logic to break long accesses, such as very long bursts on a very slow slave (e.g., an external low speed memory). At each arbitration time, a counter is loaded with the value previously written in the SLOT_CYCLE field of the related Slave Configuration Register (MATRIX_SCFG) and decreased at each clock cycle. When the counter elapses, the arbiter has the ability to re-arbitrate at the end of the current AHB bus access cycle.

Unless a master has a very tight access latency constraint, which could lead to data overflow or underflow due to a badly undersized internal FIFO with respect to its throughput, the Slot Cycle Limit should be disabled (SLOT_CYCLE = 0) or set to its default maximum value in order not to inefficiently break long bursts performed by some Atmel masters.

In most cases, this feature is not needed and should be disabled for power saving.

Warning: This feature cannot prevent any slave from locking its access indefinitely.

24.8.2 Arbitration Priority Scheme

The bus Matrix arbitration scheme is organized in priority pools.

Round-robin priority is used in the highest and lowest priority pools, whereas fixed level priority is used between priority pools and in the intermediate priority pools.

For each slave, each master is assigned to one of the slave priority pools through the priority registers for slaves (MxPR fields of MATRIX_PRAS and MATRIX_PRBS). When evaluating master requests, this programmed priority level always takes precedence.

After reset, all the masters belong to the lowest priority pool (MxPR = 0) and are therefore granted bus access in a true round-robin order.

The highest priority pool must be specifically reserved for masters requiring very low access latency. If more than one master belongs to this pool, they will be granted bus access in a biased round-robin manner which allows tight and deterministic maximum access latency from AHB bus requests. In the worst case, any currently occurring high-priority master request will be granted after the current bus master access has ended and other high priority pool master requests, if any, have been granted once each.

The lowest priority pool shares the remaining bus bandwidth between AHB Masters.

Intermediate priority pools allow fine priority tuning. Typically, a moderately latency-critical master or a bandwidth-only critical master will use such a priority level. The higher the priority level (MxPR value), the higher the master priority.

All combinations of MxPR values are allowed for all masters and slaves. For example, some masters might be assigned the highest priority pool (round-robin), and remaining masters the lowest priority pool (round-robin), with no master for intermediate fix priority levels.

If more than one master requests the slave bus, regardless of the respective masters priorities, no master will be granted the slave bus for two consecutive runs. A master can only get back-to-back grants so long as it is the only requesting master.

24.8.2.1 Fixed Priority Arbitration

Fixed priority arbitration algorithm is the first and only arbitration algorithm applied between masters from distinct priority pools. It is also used in priority pools other than the highest and lowest priority pools (intermediate priority pools).

Fixed priority arbitration allows the Bus Matrix arbiters to dispatch the requests from different masters to the same slave by using the fixed priority defined by the user in the MxPR field for each master in the Priority Registers, MATRIX_PRAS and MATRIX_PRBS. If two or more master requests are active at the same time, the master with the highest priority MxPR number is serviced first.

In intermediate priority pools, if two or more master requests with the same priority are active at the same time, the master with the highest number is serviced first.

24.8.2.2 Round-Robin Arbitration

This algorithm is only used in the highest and lowest priority pools. It allows the Bus Matrix arbiters to properly dispatch requests from different masters to the same slave. If two or more master requests are active at the same time in the priority pool, they are serviced in a round-robin increasing master number order.

24.9 System I/O Configuration

The System I/O Configuration register (CCFG_SYSIO) allows to configure some I/O lines in System I/O mode (such as JTAG, ERASE, USB, etc...) or as general purpose I/O lines. Enabling or disabling the corresponding I/O lines in peripheral mode or in PIO mode (PIO_PER or PIO_PDR registers) in the PIO controller as no effect. However, the direction (input or output), pull-up, pull-down and other mode control is still managed by the PIO controller.

24.10 SMC NAND Flash Chip Select Configuration

The SMC Nand Flash Chip Select Configuration Register (CCFG_SMCNFCS) allow to manage the chip select signal (NCSx) as assigned to NAND Flash or not.

Each NCSx can be individually assigned to Nand Flash or not. When the NCSx is assigned to NANDFLASH, the signals NANDOE and NANDWE are used for the NCSx signals selected.

24.11 Write Protect Registers

To prevent any single software error that may corrupt the Bus Matrix behavior, the entire Bus Matrix address space can be write-protected by setting the WPEN bit in the Bus Matrix Write Protect Mode Register (MATRIX_WPMR).

If WPEN is at one and a write access in the Bus Matrix address space is detected, then the WPVS flag in the Bus Matrix Write Protect Status Register (MATRIX_WPSR) is set and the field WPVSRC indicates in which register the write access has been attempted.

The WPVS flag is reset by writing the Bus Matrix Write Protect Mode Register (MATRIX_WPMR) with the appropriate access key WPKEY.

The protected registers are:

"Bus Matrix Master Configuration Registers"

"Bus Matrix Slave Configuration Registers"

"Bus Matrix Priority Registers A For Slaves"

"Bus Matrix Master Remap Control Register"

"Write Protect Mode Register"

24.12 Bus Matrix (MATRIX) User Interface

Table 24-3. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	Master Configuration Register 0	MATRIX_MCFG0	Read-write	0x00000001
0x0004	Master Configuration Register 1	MATRIX_MCFG1	Read-write	0x00000000
0x0008	Master Configuration Register 2	MATRIX_MCFG2	Read-write	0x00000000
0x000C	Master Configuration Register 3	MATRIX_MCFG3	Read-write	0x00000000
0x0010	Master Configuration Register 4	MATRIX_MCFG4	Read-write	0x00000000
0x0014	Master Configuration Register 5	MATRIX_MCFG5	Read-write	0x00000000
0x0018	Master Configuration Register 6	MATRIX_MCFG6	Read-write	0x00000000
0x001C-0x003C	Reserved	–	–	–
0x0040	Slave Configuration Register 0	MATRIX_SCFG0	Read-write	0x000001FF
0x0044	Slave Configuration Register 1	MATRIX_SCFG1	Read-write	0x000001FF
0x0048	Slave Configuration Register 2	MATRIX_SCFG2	Read-write	0x000001FF
0x004C	Slave Configuration Register 3	MATRIX_SCFG3	Read-write	0x000001FF
0x0050	Slave Configuration Register 4	MATRIX_SCFG4	Read-write	0x000001FF
0x0054	Slave Configuration Register 5	MATRIX_SCFG5	Read-write	0x000001FF
0x0058-0x007C	Reserved	–	–	–
0x0080	Priority Register A for Slave 0	MATRIX_PRAS0	Read-write	0x33333333(1)
0x0084	Reserved	–	–	–
0x0088	Priority Register A for Slave 1	MATRIX_PRAS1	Read-write	0x33333333(1)
0x008C	Reserved	–	–	–
0x0090	Priority Register A for Slave 2	MATRIX_PRAS2	Read-write	0x33333333(1)
0x0094	Reserved	–	–	–
0x0098	Priority Register A for Slave 3	MATRIX_PRAS3	Read-write	0x33333333(1)
0x009C	Reserved	–	–	–
0x00A0	Priority Register A for Slave 4	MATRIX_PRAS4	Read-write	0x33333333(1)
0x00A4	Reserved	–	–	–
0x00A8	Priority Register A for Slave 5	MATRIX_PRAS5	Read-write	0x33333333(1)
0x00AC	Reserved	–	–	–
0x00B4-0x00FC	Reserved	–	–	–
0x0100	Master Remap Control Register	MATRIX_MRCR	Read-write	0x00000000
0x0104 - 0x010C	Reserved	–	–	–
0x0110	Reserved	–	–	–
0x0114	System I/O Configuration Register	CCFG_SYSIO	Read-write	0x00000000
0x0118 - 0x0120	Reserved	–	–	–
0x0124	SMC NAND Flash Chip Select Configuration Register	CCFG_SMCNFCS	Read-write	0x00000000

Table 24-3. Register Mapping (Continued)

Offset	Register	Name	Access	Reset
0x0128 - 0x01E0	Reserved	–	–	–
0x01E4	Write Protect Mode Register	MATRIX_WPMR	Read-write	0x00000000
0x01E8	Write Protect Status Register	MATRIX_WPSR	Read-only	0x00000000

Notes: 1. Values in the Bus Matrix Priority Registers are product dependent.

24.12.1 Bus Matrix Master Configuration Registers

Name: MATRIX_MCFGx [x=0..6]

Address: 0x400E0200

Access: Read-write

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

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

- **ULBT: Undefined Length Burst Type**

0: Unlimited Length Burst

No predicted end of burst is generated, therefore INCR bursts coming from this master can only be broken if the Slave Slot Cycle Limit is reached. If the Slot Cycle Limit is not reached, the burst is normally completed by the master, at the latest, on the next AHB 1-Kbyte address boundary, allowing up to 256-beat word bursts or 128-beat double-word bursts.

This value should not be used in the very particular case of a master capable of performing back-to-back undefined length bursts on a single slave, since this could indefinitely freeze the slave arbitration and thus prevent another master from accessing this slave.

1: Single Access

The undefined length burst is treated as a succession of single accesses, allowing re-arbitration at each beat of the INCR burst or bursts sequence.

2: 4-beat Burst

The undefined length burst or bursts sequence is split into 4-beat bursts or less, allowing re-arbitration every 4 beats.

3: 8-beat Burst

The undefined length burst or bursts sequence is split into 8-beat bursts or less, allowing re-arbitration every 8 beats.

4: 16-beat Burst

The undefined length burst or bursts sequence is split into 16-beat bursts or less, allowing re-arbitration every 16 beats.

5: 32-beat Burst

The undefined length burst or bursts sequence is split into 32-beat bursts or less, allowing re-arbitration every 32 beats.

6: 64-beat Burst

The undefined length burst or bursts sequence is split into 64-beat bursts or less, allowing re-arbitration every 64 beats.

7: 128-beat Burst

The undefined length burst or bursts sequence is split into 128-beat bursts or less, allowing re-arbitration every 128 beats.

Unless duly needed, the ULBT should be left at its default 0 value for power saving.

24.12.2 Bus Matrix Slave Configuration Registers

Name: MATRIX_SCFGx [x=0..5]

Address: 0x400E0240

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	FIXED_DEFMSTR				DEFMSTR_TYPE	

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	SLOT_CYCLE

7	6	5	4	3	2	1	0
SLOT_CYCLE							

This register can only be written if the WPEN bit is cleared in the "Write Protect Mode Register" .

- **SLOT_CYCLE: Maximum Bus Grant Duration for Masters**

When SLOT_CYCLE AHB clock cycles have elapsed since the last arbitration, a new arbitration takes place to let another master access this slave. If another master is requesting the slave bus, then the current master burst is broken.

If SLOT_CYCLE = 0, the Slot Cycle Limit feature is disabled and bursts always complete unless broken according to the ULBT.

This limit has been placed in order to enforce arbitration so as to meet potential latency constraints of masters waiting for slave access.

This limit must not be too small. Unreasonably small values break every burst and the Bus Matrix arbitrates without performing any data transfer. The default maximum value is usually an optimal conservative choice.

In most cases, this feature is not needed and should be disabled for power saving.

See "Slot Cycle Limit Arbitration" for details.

- **DEFMSTR_TYPE: Default Master Type**

0: No Default Master

At the end of the current slave access, if no other master request is pending, the slave is disconnected from all masters.

This results in a one clock cycle latency for the first access of a burst transfer or for a single access.

1: Last Default Master

At the end of the current slave access, if no other master request is pending, the slave stays connected to the last master having accessed it.

This results in not having one clock cycle latency when the last master tries to access the slave again.

2: Fixed Default Master

At the end of the current slave access, if no other master request is pending, the slave connects to the fixed master the number that has been written in the FIXED_DEFMSTR field.

This results in not having one clock cycle latency when the fixed master tries to access the slave again.

- **FIXED_DEFMSTR: Fixed Default Master**

This is the number of the Default Master for this slave. Only used if DEFMSTR_TYPE is 2. Specifying the number of a master which is not connected to the selected slave is equivalent to setting DEFMSTR_TYPE to 0.

24.12.3 Bus Matrix Priority Registers A For Slaves

Name: MATRIX_PRASx [x=0..5]

Address: 0x400E0280 [0], 0x400E0288 [1], 0x400E0290 [2], 0x400E0298 [3], 0x400E02A0 [4], 0x400E02A8 [5]

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	M6PR	

23	22	21	20	19	18	17	16
–	–	M5PR		–	–	M4PR	

15	14	13	12	11	10	9	8
–	–	M3PR		–	–	M2PR	

7	6	5	4	3	2	1	0
–	–	M1PR		–	–	M0PR	

This register can only be written if the WPE bit is cleared in the "Write Protect Mode Register" .

- **MxPR: Master x Priority**

Fixed priority of Master x for accessing the selected slave. The higher the number, the higher the priority.

All the masters programmed with the same MxPR value for the slave make up a priority pool.

Round-robin arbitration is used in the lowest (MxPR = 0) and highest (MxPR = 3) priority pools.

Fixed priority is used in intermediate priority pools (MxPR = 1) and (MxPR = 2).

See "Arbitration Priority Scheme" for details.

24.12.4 Bus Matrix Master Remap Control Register

Name: MATRIX_MRCR

Address: 0x400E0300

Access: Read-write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	RCB6	RCB5	RCB4	RCB3	RCB2	RCB1	RCB0

This register can only be written if the WPEN bit is cleared in the "Write Protect Mode Register" .

- **RCBx: Remap Command Bit for Master x**

0: Disable remapped address decoding for the selected Master

1: Enable remapped address decoding for the selected Master

24.12.5 System I/O Configuration Register

Name: CCFG_SYSIO

Address: 0x400E0314

Access: Read-write

Reset: 0x0000_0000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	SYSIO12	SYSIO11	SYSIO10	–	–
7	6	5	4	3	2	1	0
SYSIO7	SYSIO6	SYSIO5	SYSIO4	–	–	–	–

- **SYSIO4: PB4 or TDI Assignment**

0 = TDI function selected.

1 = PB4 function selected.

- **SYSIO5: PB5 or TDO/TRACESWO Assignment**

0 = TDO/TRACESWO function selected.

1 = PB5 function selected.

- **SYSIO6: PB6 or TMS/SWDIO Assignment**

0 = TMS/SWDIO function selected.

1 = PB6 function selected.

- **SYSIO7: PB7 or TCK/SWCLK Assignment**

0 = TCK/SWCLK function selected.

1 = PB7 function selected.

- **SYSIO10: PB10 or DDM Assignment**

0 = DDM function selected.

1 = PB10 function selected.

- **SYSIO11: PB11 or DDP Assignment**

0 = DDP function selected.

1 = PB11 function selected.

- **SYSIO12: PB12 or ERASE Assignment**

0 = ERASE function selected.

1 = PB12 function selected.

24.12.6 SMC NAND Flash Chip Select Configuration Register

Name: CCFG_SMCNFCS

Address: 0x400E0324

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
–	–	–	–	SMC_NFCS3	SMC_NFCS2	SMC_NFCS1	SMC_NFCS0

- **SMC_NFCS0: SMC NAND Flash Chip Select 0 Assignment**

0 = NCS0 is not assigned to a NAND Flash (NANDOE and NANWE not used for NCS0)

1 = NCS0 is assigned to a NAND Flash (NANDOE and NANWE used for NCS0)

- **SMC_NFCS1: SMC NAND Flash Chip Select 1 Assignment**

0 = NCS1 is not assigned to a NAND Flash (NANDOE and NANWE not used for NCS1)

1 = NCS1 is assigned to a NAND Flash (NANDOE and NANWE used for NCS1)

- **SMC_NFCS2: SMC NAND Flash Chip Select 2 Assignment**

0 = NCS2 is not assigned to a NAND Flash (NANDOE and NANWE not used for NCS2)

1 = NCS2 is assigned to a NAND Flash (NANDOE and NANWE used for NCS2)

- **SMC_NFCS3: SMC NAND Flash Chip Select 3 Assignment**

0 = NCS3 is not assigned to a NAND Flash (NANDOE and NANWE not used for NCS3)

1 = NCS3 is assigned to a NAND Flash (NANDOE and NANWE used for NCS3)

24.12.7 Write Protect Mode Register

Name: MATRIX_WPMR

Address: 0x400E03E4

Access: Read-write

31	30	29	28	27	26	25	24
WPKEY							

23	22	21	20	19	18	17	16
WPKEY							

15	14	13	12	11	10	9	8
WPKEY							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WPEN

For more details on MATRIX_WPMR, please refer to Section 24.11 “Write Protect Registers”.

The protected registers are:

“Bus Matrix Master Configuration Registers”

“Bus Matrix Slave Configuration Registers”

“Bus Matrix Priority Registers A For Slaves”

“Bus Matrix Master Remap Control Register”

“Write Protect Mode Register”

- **WPEN: Write Protect Enable**

0: Disables the Write Protect if WPKEY corresponds to 0x4D4154 (“MAT” in ASCII).

1: Enables the Write Protect if WPKEY corresponds to 0x4D4154 (“MAT” in ASCII).

Protects the entire Bus Matrix address space from address offset 0x000 to 0x1FC.

- **WPKEY: Write Protect KEY** (Write-only)

Value	Name	Description
0x4D4154	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

24.12.8 Write Protect Status Register

Name: MATRIX_WPSR

Address: 0x400E03E8

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
WPVSRC							

15	14	13	12	11	10	9	8
WPVSRC							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WPVS

For more details on MATRIX_WPSR, please refer to Section 24.11 “Write Protect Registers”.

- **WPVS: Write Protect Violation Status**

0: No Write Protect Violation has occurred since the last write of the MATRIX_WPMR.

1: At least one Write Protect Violation has occurred since the last write of the MATRIX_WPMR.

- **WPVSRC: Write Protect Violation Source**

When WPVS is active, this field indicates the register address offset in which a write access has been attempted.

Otherwise it reads as 0.

25. DMA Controller (DMAC)

25.1 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.

25.2 Embedded Characteristics

- 1 AHB-Lite Master Interfaces
- DMA Module Supports the Following Transfer Schemes: Peripheral-to-Memory, Memory-to-Peripheral, Peripheral-to-Peripheral and Memory-to-Memory
- Source and Destination Operate independently on BYTE (8-bit), HALF-WORD (16-bit) and WORD (32-bit)
- Supports Hardware and Software Initiated Transfers
- Supports Multiple Buffer Chaining Operations
- Supports Incrementing/decrementing/fixed Addressing Mode Independently for Source and Destination
- Programmable Arbitration Policy, Modified Round Robin and Fixed Priority are Available
- Supports Specified Length and Unspecified Length AMBA AHB Burst Access to Maximize Data Bandwidth
- AMBA APB Interface Used to Program the DMA Controller
- 4 DMA Channels 16 External Request Lines
- Embedded FIFO
- Channel Locking and Bus Locking Capability
- Register Write Protection

25.3 DMA Controller Peripheral Connections

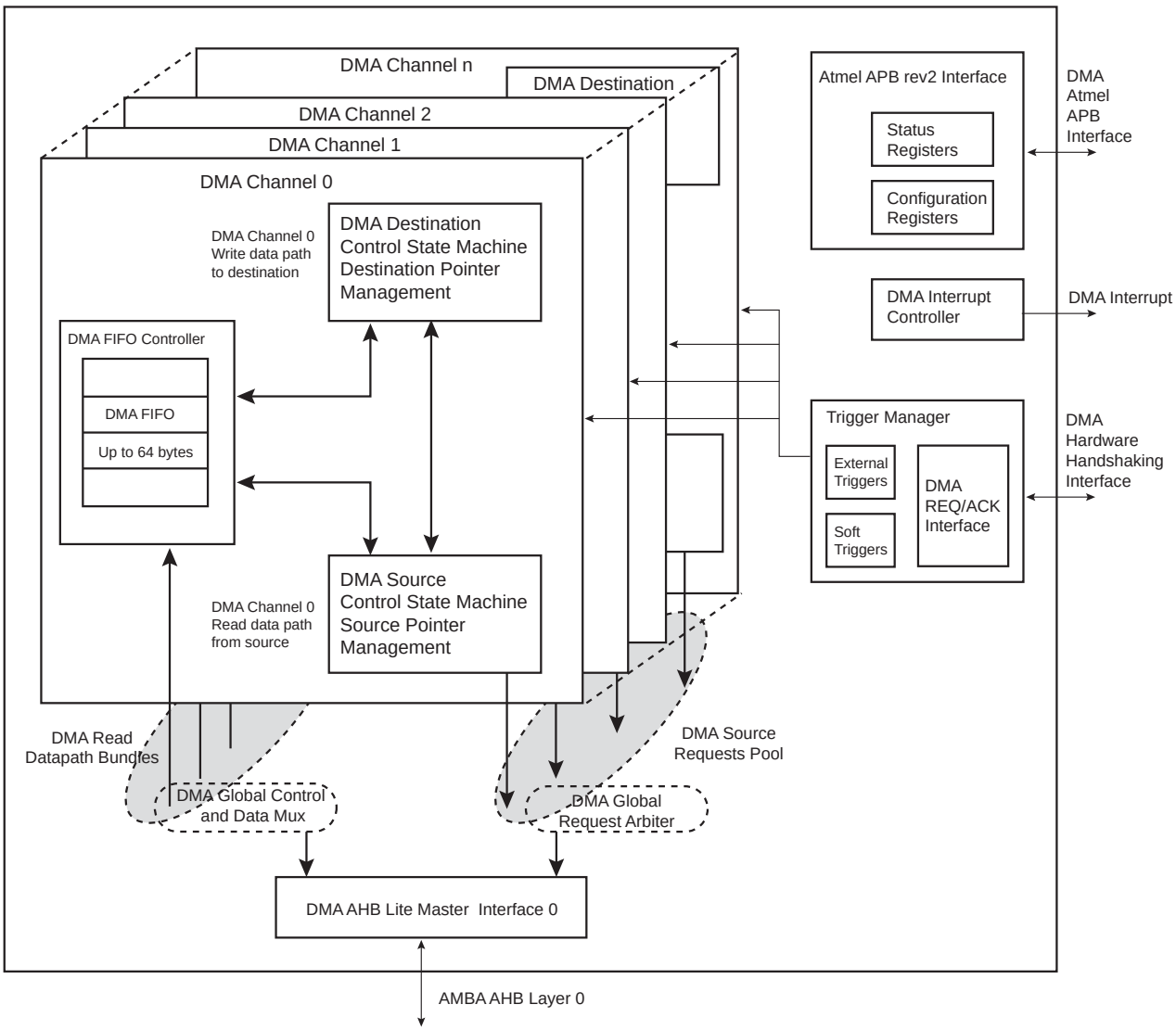
The DMA Controller handles the transfer between peripherals and memory and receives triggers from the peripherals listed in the following table.

Table 25-1. DMA Channel Definition

Instance Name	Transmit/Receive	DMA Channel Number
HSMCI	Transmit/Receive	0
SPI	Transmit	1
SPI	Receive	2
USART0	Transmit	3
USART0	Receive	4
USART1	Transmit	5
USART1	Receive	6
AES	Transmit	11
AES	Receive	12
PWM	Transmit	13

25.4 Block Diagram

Figure 25-1. DMA Controller (DMAC) Block Diagram



25.5 Product Dependencies

25.5.1 Interrupt Sources

The DMAC interrupt line is connected to one of the internal sources of the interrupt controller. Using the DMAC interrupt requires prior programming of the interrupt controller.

Table 25-2. Peripheral IDs

Instance	ID
DMAC	20

25.6 Functional Description

25.6.1 Basic Definitions

Source peripheral: Device on an AMBA layer from where the DMAC reads data, which is then stored in the channel FIFO. The source peripheral teams up with a destination peripheral to form a channel.

Destination peripheral: Device to which the DMAC writes the stored data from the FIFO (previously read from the source peripheral).

Memory: Source or destination that is always “ready” for a DMAC transfer and does not require a handshaking interface to interact with the DMAC.

Programmable Arbitration Policy: Modified Round Robin and Fixed Priority are available by means of the ARB_CFG bit in the Global Configuration Register (DMAC_GCFG). The fixed priority is linked to the channel number. The highest DMAC channel number has the highest priority.

Channel: Read/write datapath between a source peripheral on one configured AMBA layer and a destination peripheral on the same or different AMBA layer that occurs through the channel FIFO. If the source peripheral is not memory, then a source handshaking interface is assigned to the channel. If the destination peripheral is not memory, then a destination handshaking interface is assigned to the channel. Source and destination handshaking interfaces can be assigned dynamically by programming the channel registers.

Master interface: DMAC is a master on the AHB bus reading data from the source and writing it to the destination over the AHB bus.

Slave interface: The APB interface over which the DMAC is programmed. The slave interface in practice could be on the same layer as any of the master interfaces or on a separate layer.

Handshaking interface: A set of signal registers that conform to a protocol and *handshake* between the DMAC and source or destination peripheral to control the transfer of a single or chunk transfer between them. This interface is used to request, acknowledge, and control a DMAC transaction. A channel can receive a request through one of two types of handshaking interface: hardware or software.

Hardware handshaking interface: Uses hardware signals to control the transfer of a single or chunk transfer between the DMAC and the source or destination peripheral.

Software handshaking interface: Uses software registers to control the transfer of a single or chunk transfer between the DMAC and the source or destination peripheral. No special DMAC handshaking signals are needed on the I/O of the peripheral. This mode is useful for interfacing an existing peripheral to the DMAC without modifying it.

Transfer hierarchy: Figure 25-2 illustrates the hierarchy between DMAC transfers, buffer transfers, chunk or single, and AMBA transfers (single or burst) for non-memory peripherals. Figure 25-3 shows the transfer hierarchy for memory.

Figure 25-2. DMAC Transfer Hierarchy for Non-Memory Peripheral

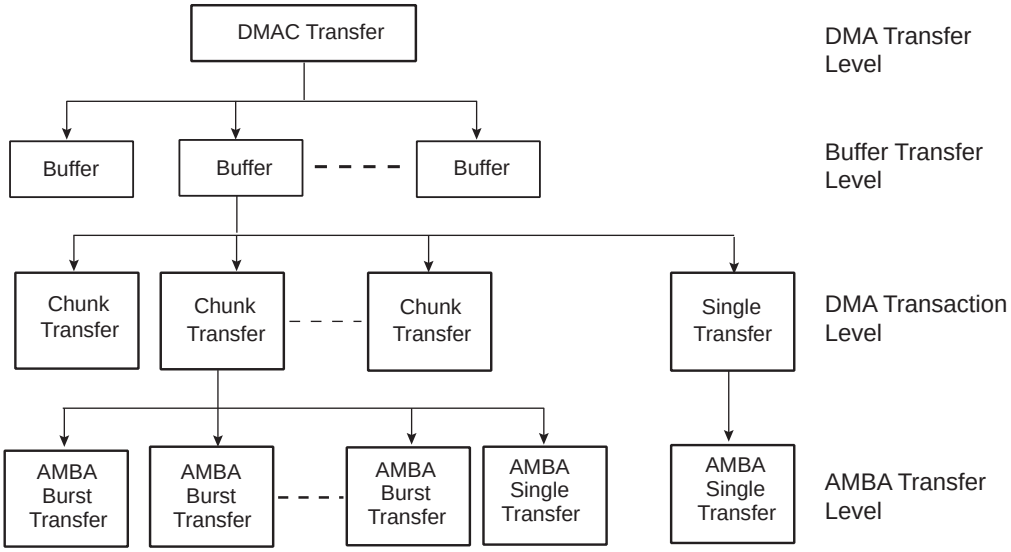
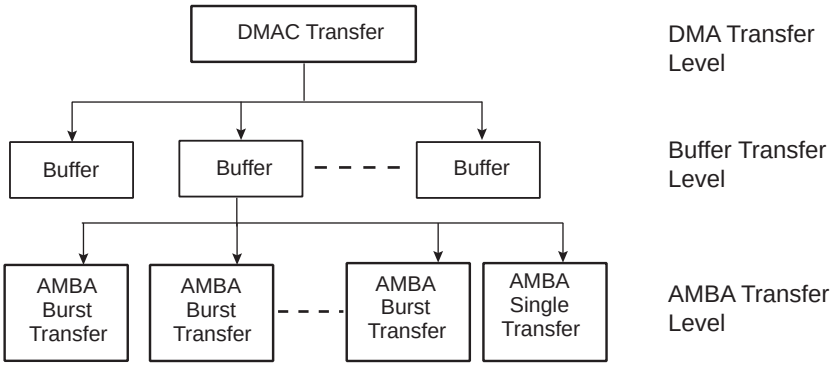


Figure 25-3. DMAC Transfer Hierarchy for Memory



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. It is then possible to reprogram 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.

25.6.2 Memory Peripherals

Figure 25-3 on page 470 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.

25.6.3 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.

25.6.3.1 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 Channel Configuration Register (DMAC_CFGx) must be cleared to enable software handshaking.

When the peripheral is not the flow controller, then the Software Last Transfer Flag Register (DMAC_LAST) is not used, and the values in these registers are ignored.

Chunk Transactions

Writing a ‘1’ to the Software Chunk Transfer Request Register (DMAC_CREQ[2x]) 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 Software Single Request Register (DMAC_SREQ[2x]) 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.

25.6.4 DMAC Transfer Types

A DMAC transfer may consist of single or multi-buffer transfers. On successive buffers of a multi-buffer transfer, DMAC_SADDRx/DMAC_DADDRx 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 reprogrammed 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, DMAC_DSCRx in the DMAC is reprogrammed using the following method:

- Buffer chaining using linked lists

A buffer descriptor (LLI) consists of the following registers: DMAC_SADDRx, DMAC_DADDRx, DMAC_DSCRx, DMAC_CTRLAx, and DMAC_CTRLBx. These registers, along with DMAC_CFGx, are used by the DMAC to set up and describe the buffer transfer.

25.6.4.1 Multi-buffer Transfers

Buffer Chaining Using Linked Lists

In this case, the DMAC reprograms 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).

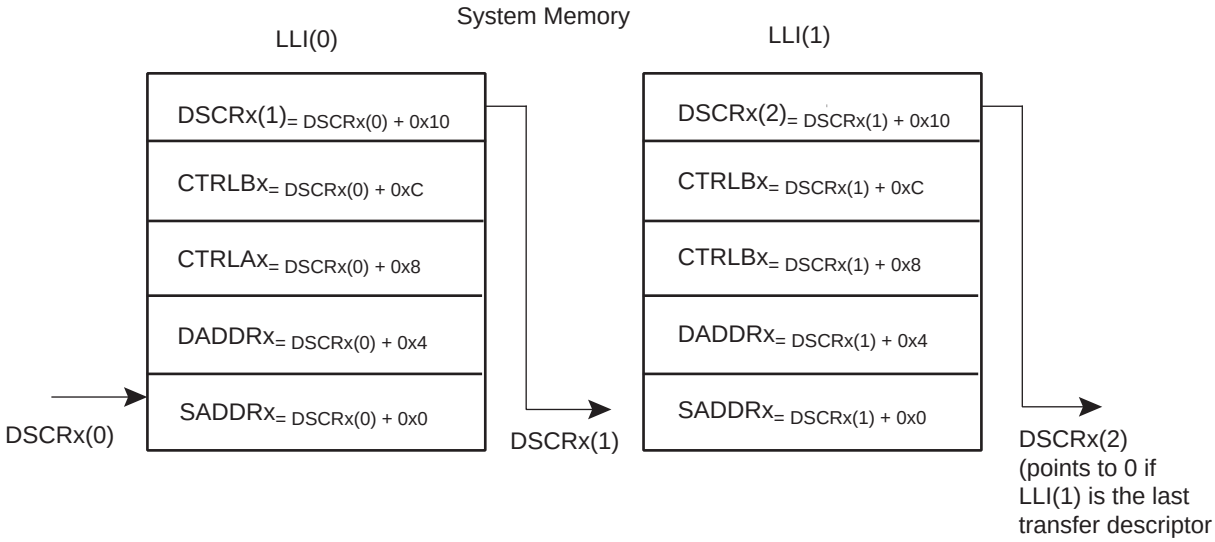
To set up buffer chaining, a sequence of linked lists must be programmed in memory.

DMAC_SADDRx, DMAC_DADDRx, DMAC_DSCRx, DMAC_CTRLAx and DMAC_CTRLBx are fetched from system memory on an LLI update. The updated content of DMAC_CTRLAx is written back to memory on buffer completion. Figure 25-4 on page 473 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 25-4. Multi-Buffer Transfer Using Linked List



25.6.4.2 Programming DMAC for Multiple Buffer Transfers

Table 25-3. Multiple Buffers Transfer Management

	Transfer Type	SRC_DSCR	DST_DSCR	BTSIZE	DSCR	SADDR	DADDR	Other Fields
1	Single Buffer or Last Buffer of a multiple buffer transfer	–	–	USR(1)	0	USR(1)	USR(1)	USR(1)
2	Multi-buffer transfer with contiguous DADDR	0	1	LLI(3)	USR(1)	LLI(3)	CONT(2)	LLI(3)
3	Multi-buffer transfer with contiguous SADDR	1	0	LLI(3)	USR(1)	CONT(2)	LLI(3)	LLI(3)
4	Multi-buffer transfer with LLI support	0	0	LLI(3)	USR(1)	LLI(3)	LLI(3)	LLI(3)

Notes: 1. USR means that the register field is manually programmed by the user.
2. CONT means that address are contiguous.
3. 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 the fields DMAC_CTRLAx.SRC_DSCR and DMAC_CTRLAx.DST_DSCR.

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.

25.6.4.3 Ending Multi-buffer Transfers

All multi-buffer transfers must end as shown in Row 1 of Table 25-3 on page 473. 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_DSCRx is set to 0.

25.6.5 Programming a Channel

Four registers, DMAC_DSCRx, DMAC_CTRLAx, 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 25-3 on page 473.

The “BTSIZE”, “SADDR” and “DADDR” columns in the table indicate where the values of DMAC_SADDRx, DMAC_DADDRx, DMAC_CTRLAx, DMAC_CTRLBx, and DMAC_DSCRx are obtained for the next buffer transfer when multi-buffer DMAC transfers are enabled.

25.6.5.1 Programming Examples

Single-buffer Transfer (Row 1)

1. Read the ENAx bit in the DMAC Channel Handler Status Register (DMAC_CHSR) to choose a free (disabled) channel.
2. Clear any pending interrupts on the channel from the previous DMAC transfer by reading the DMAC Error, Buffer Transfer and Chained Buffer Transfer Status Register (DMAC_EBCISR).
3. Program the following channel registers:
   a. Write the starting source address in DMAC_SADDRx for channel x.
   b. Write the starting destination address in DMAC_DADDRx for channel x.
   c. Write the next descriptor address in DMA_DSCRx for channel x with 0x0.
   d. Program DMAC_CTRLAx, DMAC_CTRLBx and DMAC_CFGx according to Row 1 as shown in Table 25-3 on page 473.
   e. Write the control information for the DMAC transfer in DMAC_CTRLAx and DMAC_CTRLBx for channel x. For example, in the register, it is possible to 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 field in DMAC_CTRLBx.
      – 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_INCR field.
      – Incrementing/decrementing or fixed address for destination in DST_INCR field.
   f. Write the channel configuration information into DMAC_CFGx 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 setting the ENAx bit in the DMAC Channel Handler Enable Register (DMAC_CHER), where x is the channel number. Make sure that the ENABLE bit (register bit 0) in DMAC_EN is set.
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 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 DMAC_CHSR 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 25-5 on page 476) for channel x. For example, in the register, it is possible to 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 field in DMAC_CTRLBx.
   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 DMAC_CFGx 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 25-3 on page 473. The LLI.DMAC_CTRLBx register of the last Linked List Item must be set as described in Row 1 of Table 25-3. Figure 25-4 on page 473 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 bit 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 DMAC_EBCISR.
9. Program DMAC_CTRLBx and DMAC_CFGx according to Row 4 as shown in Table 25-3 on page 473.
10. Program DMAC_DSCRx with DMAC_DSCRx(0), the pointer to the first Linked List item.
11. Finally, enable the channel by setting 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).

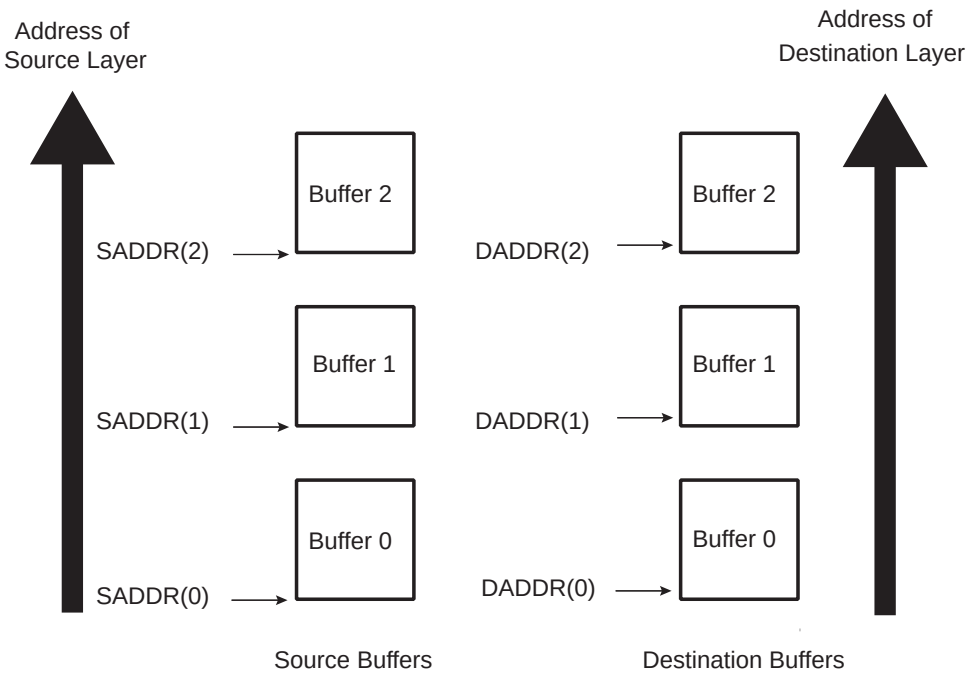
Note: The LLI.DMAC_SADDRx, LLI.DMAC_DADDRx, LLI.DMAC_DSCRx, LLI.DMAC_CTRLAx and LLI.DMAC_CTRLBx registers are fetched. The DMAC automatically reprograms the DMAC_SADDRx, DMAC_DADDRx, DMAC_DSCRx, DMAC_CTRLBx and DMAC_CTRLAx channel registers from the DMAC_DSCRx(0).

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.

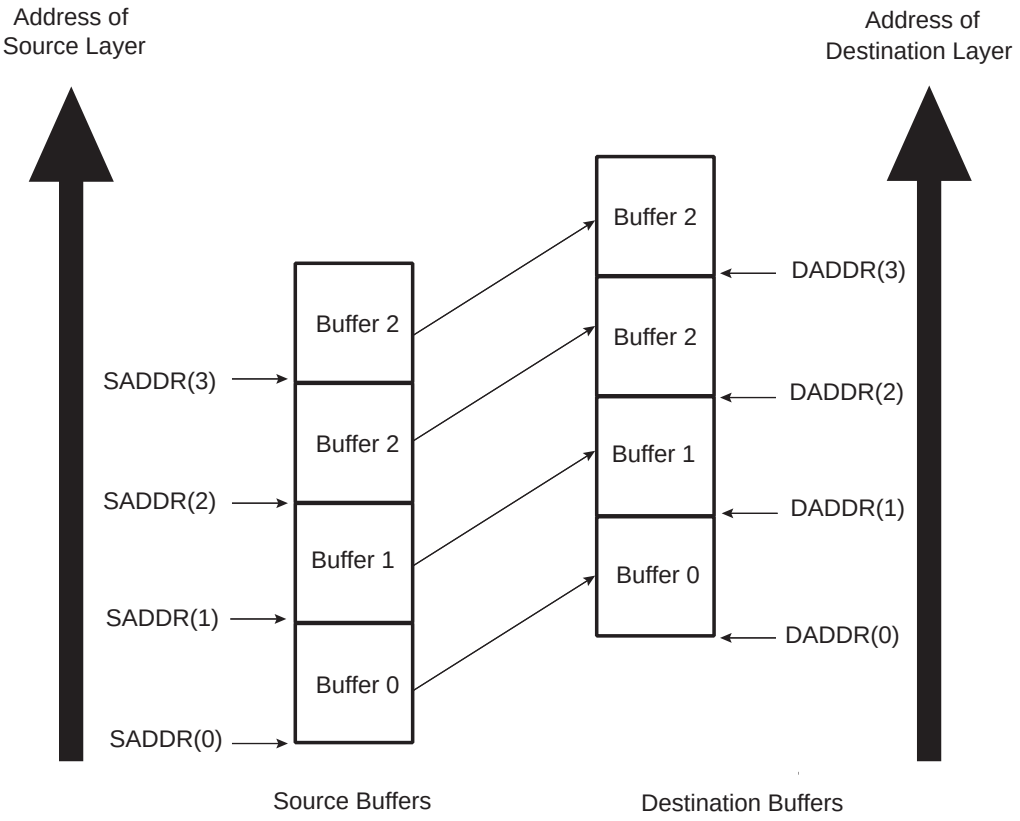
15. The DMAC does not wait for the buffer interrupt to be cleared, but continues fetching the next LLI from the memory location pointed to by current DMAC_DSCRx and automatically reprograms the DMAC_SADDRx, DMAC_DADDRx, DMAC_DSCRx, DMAC_CTRLAx and DMAC_CTRLBx channel registers. The DMAC transfer continues until the DMAC determines that the DMAC_CTRLBx and DMAC_DSCRx registers at the end of a buffer transfer match as described in Row 1 of Table 25-3 on page 473. 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 25-5 on page 476.

Figure 25-5. Multi-buffer with Linked List Address for Source and Destination



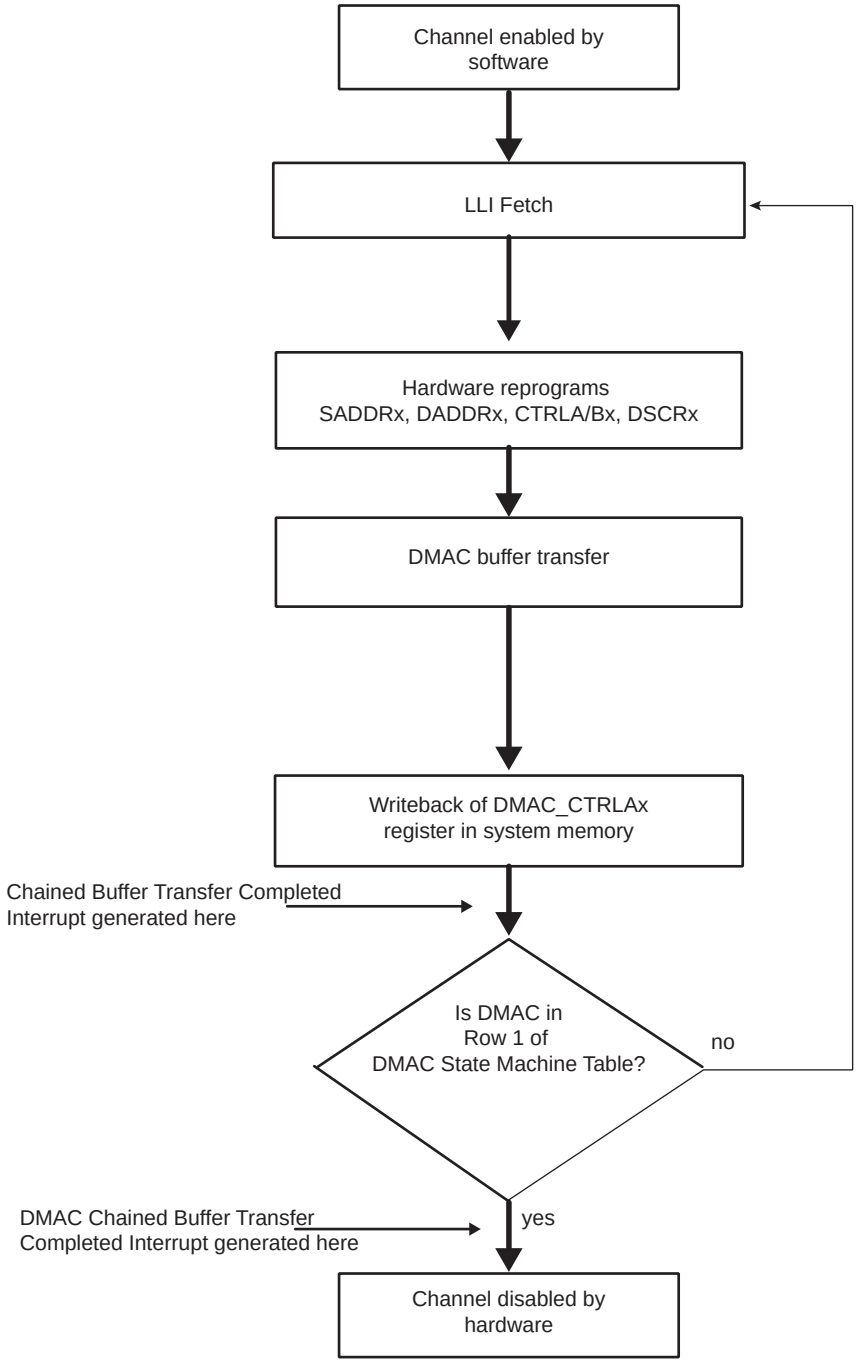
If the user needs to execute a DMAC transfer where the source and destination address are contiguous but the amount of data to be transferred is greater than the maximum buffer size DMAC_CTRLAx.BTSIZE, then this can be achieved using the type of multi-buffer transfer as shown in Figure 25-6 on page 477.

Figure 25-6. Multi-buffer with Linked Address for Source and Destination Buffers are Contiguous



The DMAC transfer flow is shown in Figure 25-7 on page 478.

Figure 25-7. DMAC Transfer Flow for Source and Destination Linked List Address



Multi-buffer DMAC Transfer with Linked List for Source and Contiguous Destination Address (Row 2)

1. Read the DMAC_CHSR 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, it is possible to 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 field in DMAC_CTRLBx.

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 DMAC_DADDRx 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 DMAC_CFGx 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 25-3 on page 473, while the LLI.DMAC_CTRLBx register of the last Linked List item must be set as described in Row 1 of Table 25-3. Figure 25-4 on page 473 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 bit 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 DMAC_CTRLAx, DMAC_CTRLBx and DMAC_CFGx according to Row 2 as shown in Table 25-3 on page 473.
11. Program DMAC_DSCRx with DMAC_DSCRx(0), the pointer to the first Linked List item.
12. Finally, enable the channel by setting the DMAC_CHER.ENAx bit. The transfer is performed. Make sure that the ENABLE bit (register bit 0) in DMAC_EN is set.
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.

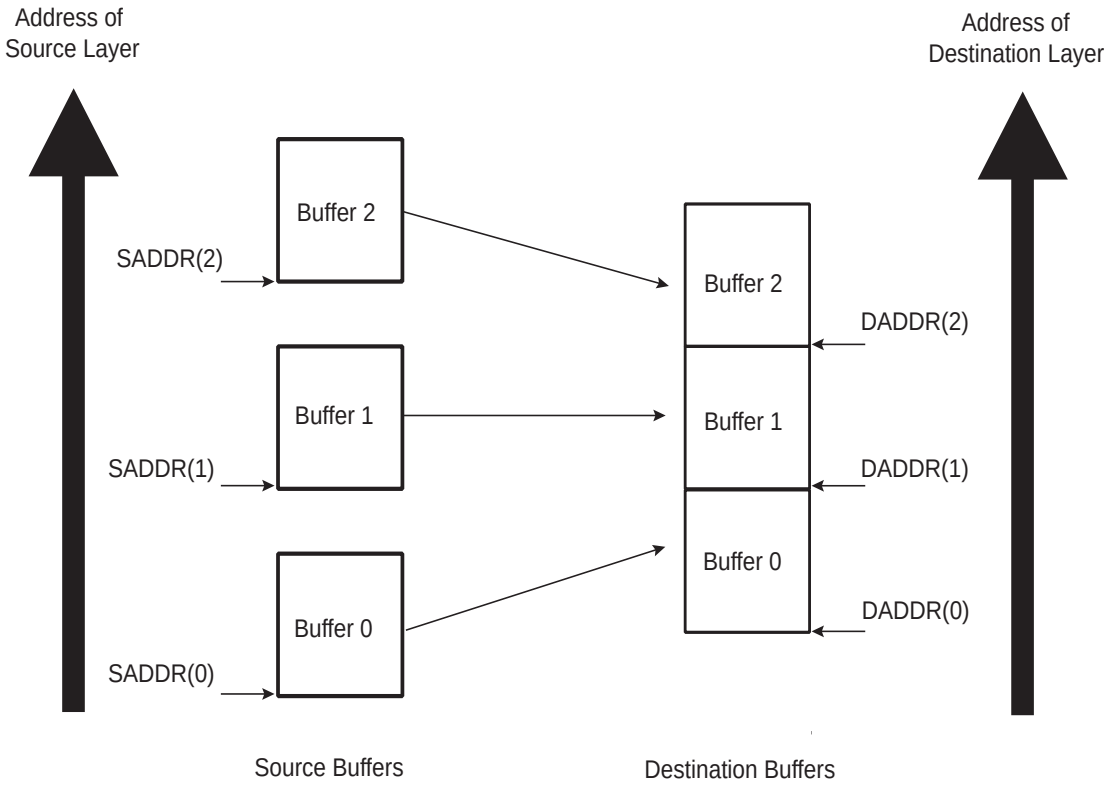
14. Source and destination requests single and chunk DMAC transactions to transfer the buffer of data (assuming non-memory peripherals). The DMAC acknowledges at the completion of every transaction (chunk and single) in the buffer and carries out the buffer transfer.
15. Once the buffer of data is transferred, the DMAC_CTRLAx register is written out to the system memory at the same location and on the same layer 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. DMAC_DADDRx 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 25-3 on page 473. The DMAC then knows that the previous buffer transferred was the last buffer in the DMAC transfer.

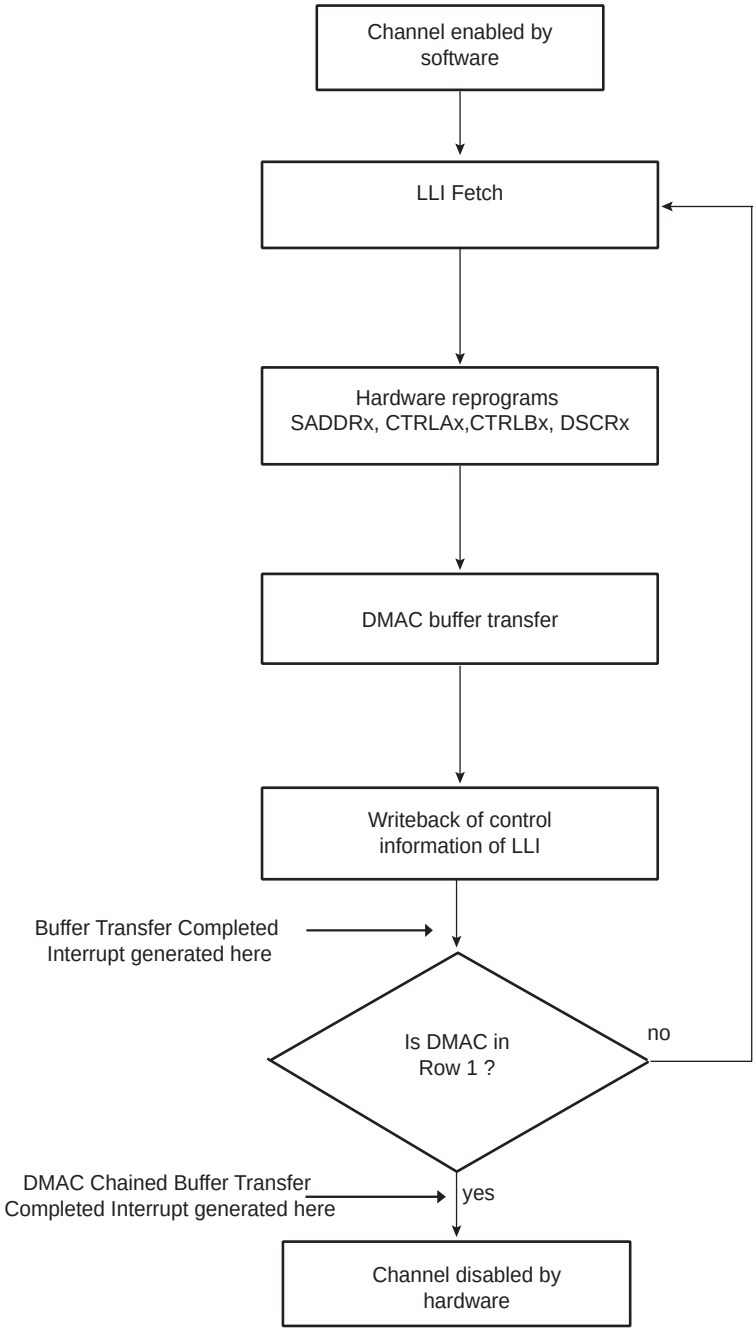
The DMAC transfer might look like that shown in Figure 25-8. Note that the destination address is decrementing.

Figure 25-8. DMAC Transfer with Linked List Source Address and Contiguous Destination Address



The DMAC transfer flow is shown in Figure 25-9 on page 481.

Figure 25-9. DMAC Transfer Flow for Linked List Source Address and Contiguous Destination Address



25.6.6 Disabling a Channel Prior to Transfer Completion

Under normal operation, the software enables a channel by setting the DMAC_CHER.ENAx bit, and the hardware disables a channel on transfer completion by clearing the DMAC_CHSR.ENAx 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 DMAC_CHSR.

1. If the software chooses to disable a channel n prior to the DMAC transfer completion, then it can set the DMAC_CHER.SUSPx bit to instruct 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_CTRLAx.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 by setting the DMAC_CHDR.RESx bit. 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.

25.6.6.1 Abnormal Transfer Termination

A DMAC transfer may be terminated abruptly by software by clearing the channel enable bit, DMAC_CHER.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 general enable bit in the DMAC Enable Register (DMAC_EN.ENABLE). Again, this does not mean that all channels are disabled immediately after the DMAC_EN.ENABLE bit 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.

25.6.7 Register Write Protection

To prevent any single software error from corrupting DMAC behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the “DMAC Write Protection Mode Register” (DMAC_WPMR).

If a write access to a write-protected register is detected, the WPVS bit in the “DMAC Write Protection Status Register” (DMAC_WPSR) is set and the field WPVSRC indicates the register in which the write access has been attempted.

The WPVS bit is automatically cleared after reading the DMAC_WPSR.

The following registers can be write-protected:

- “DMAC Global Configuration Register”
- “DMAC Enable Register”
- “DMAC Channel x [x = 0..3] Source Address Register”
- “DMAC Channel x [x = 0..3] Destination Address Register”
- “DMAC Channel x [x = 0..3] Descriptor Address Register”
- “DMAC Channel x [x = 0..3] Control A Register”

- “DMAC Channel x [x = 0..3] Control B Register”
- “DMAC Channel x [x = 0..3] Configuration Register”

25.7 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.
- The channel registers DMAC_SADDRx and DMAC_DADDRx must be programmed 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 DMAC Channel Handler Disable Register, it must re-enable the channel only after it has polled a ‘0’ in the DMAC Channel Handler Status Register. This is because the current AHB Burst must terminate properly.
- If the value of field DMAC_CTRLAx.BTSIZE is configured to zero and the DMAC has been defined as the flow controller, the channel is automatically disabled.
- 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 this is not done and the first DMAC request is also the last transfer, the DMAC channel might miss a Last Transfer Flag.

25.8 DMA Controller (DMAC) User Interface

Table 25-4. Register Mapping

Offset	Register	Name	Access	Reset
0x000	DMAC Global Configuration Register	DMAC_GCFG	Read/Write	0x10
0x004	DMAC Enable Register	DMAC_EN	Read/Write	0x0
0x008	DMAC Software Single Request Register	DMAC_SREQ	Read/Write	0x0
0x00C	DMAC Software Chunk Transfer Request Register	DMAC_CREQ	Read/Write	0x0
0x010	DMAC Software Last Transfer Flag Register	DMAC_LAST	Read/Write	0x0
0x014	Reserved	–	–	–
0x018	DMAC Error, Chained Buffer Transfer Completed Interrupt and Buffer Transfer Completed Interrupt Enable Register	DMAC_EBCIER	Write-only	–
0x01C	DMAC Error, Chained Buffer Transfer Completed Interrupt and Buffer Transfer Completed Interrupt Disable Register	DMAC_EBCIDR	Write-only	–
0x020	DMAC Error, Chained Buffer Transfer Completed Interrupt and Buffer transfer completed Mask Register	DMAC_EBCIMR	Read-only	0x0
0x024	DMAC Error, Chained Buffer Transfer Completed Interrupt and Buffer transfer completed Status Register	DMAC_EBCISR	Read-only	0x0
0x028	DMAC Channel Handler Enable Register	DMAC_CHER	Write-only	–
0x02C	DMAC Channel Handler Disable Register	DMAC_CHDR	Write-only	–
0x030	DMAC Channel Handler Status Register	DMAC_CHSR	Read-only	0x00FF0000
0x034–0x038	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 Protection Mode Register	DMAC_WPMR	Read/Write	0x0
0x1E8	DMAC Write Protection Status Register	DMAC_WPSR	Read-only	0x0
0x1EC–0x1FC	Reserved	–	–	–

25.8.1 DMAC Global Configuration Register

Name: DMAC_GCFG

Address: 0x400C0000

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
–	–	–	ARB_CFG	–	–	–	–

Note: Bit fields 0, 1, 2, and 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 Protection Mode Register” .

- **ARB_CFG: Arbiter Configuration**

Value	Name	Description
0	FIXED	Fixed priority arbiter (see “Basic Definitions”)
1	ROUND_ROBIN	Modified round robin arbiter.

25.8.2 DMAC Enable Register

Name: DMAC_EN

Address: 0x400C0004

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
–	–	–	–	–	–	–	ENABLE

This register can only be written if the WPEN bit is cleared in “DMAC Write Protection Mode Register” .

- **ENABLE: General Enable of DMA**

0: DMA Controller is disabled.

1: DMA Controller is enabled.

25.8.3 DMAC Software Single Request Register

Name: DMAC_SREQ

Address: 0x400C0008

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
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.

25.8.4 DMAC Software Chunk Transfer Request Register

Name: DMAC_CREQ

Address: 0x400C000C

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
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.

25.8.5 DMAC Software Last Transfer Flag Register

Name: DMAC_LAST

Address: 0x400C0010

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
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.

25.8.6 DMAC Error, Buffer Transfer and Chained Buffer Transfer Interrupt Enable Register

Name: DMAC_EBCIER

Address: 0x400C0018

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	ERR3	ERR2	ERR1	ERR0
15	14	13	12	11	10	9	8
–	–	–	–	CBTC3	CBTC2	CBTC1	CBTC0
7	6	5	4	3	2	1	0
–	–	–	–	BTC3	BTC2	BTC1	BTC0

- **BTCx: Buffer Transfer Completed [3:0]**

Buffer Transfer Completed Interrupt Enable Register. Set the relevant bit in the BTC field to enable the interrupt for channel i.

- **CBTCx: Chained Buffer Transfer Completed [3:0]**

Chained Buffer Transfer Completed Interrupt Enable Register. Set the relevant bit in the CBTC field to enable the interrupt for channel i.

- **ERRx: Access Error [3:0]**

Access Error Interrupt Enable Register. Set the relevant bit in the ERR field to enable the interrupt for channel i.

25.8.7 DMAC Error, Buffer Transfer and Chained Buffer Transfer Interrupt Disable Register

Name: DMAC_EBCIDR

Address: 0x400C001C

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	ERR3	ERR2	ERR1	ERR0
15	14	13	12	11	10	9	8
–	–	–	–	CBTC3	CBTC2	CBTC1	CBTC0
7	6	5	4	3	2	1	0
–	–	–	–	BTC3	BTC2	BTC1	BTC0

- **BTCx: Buffer Transfer Completed [3:0]**

Buffer transfer completed Disable Interrupt Register. When set, a bit of the BTC field disables the interrupt from the relevant DMAC channel.

- **CBTCx: Chained Buffer Transfer Completed [3:0]**

Chained Buffer transfer completed Disable Register. When set, a bit of the CBTC field disables the interrupt from the relevant DMAC channel.

- **ERRx: Access Error [3:0]**

Access Error Interrupt Disable Register. When set, a bit of the ERR field disables the interrupt from the relevant DMAC channel.

25.8.8 DMAC Error, Buffer Transfer and Chained Buffer Transfer Interrupt Mask Register

Name: DMAC_EBCIMR

Address: 0x400C0020

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	ERR3	ERR2	ERR1	ERR0
15	14	13	12	11	10	9	8
–	–	–	–	CBTC3	CBTC2	CBTC1	CBTC0
7	6	5	4	3	2	1	0
–	–	–	–	BTC3	BTC2	BTC1	BTC0

- **BTCx: Buffer Transfer Completed [3:0]**

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 [3:0]**

0: Chained Buffer Transfer interrupt is disabled for channel i.

1: Chained Buffer Transfer interrupt is enabled for channel i.

- **ERRx: Access Error [3:0]**

0: Transfer Error Interrupt is disabled for channel i.

1: Transfer Error Interrupt is enabled for channel i.

25.8.9 DMAC Error, Buffer Transfer and Chained Buffer Transfer Status Register

Name: DMAC_EBCISR

Address: 0x400C0024

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	ERR3	ERR2	ERR1	ERR0
15	14	13	12	11	10	9	8
–	–	–	–	CBTC3	CBTC2	CBTC1	CBTC0
7	6	5	4	3	2	1	0
–	–	–	–	BTC3	BTC2	BTC1	BTC0

- **BTCx: Buffer Transfer Completed [3:0]**

When BTC[*i*] is set, Channel *i* buffer transfer has terminated.

- **CBTCx: Chained Buffer Transfer Completed [3:0]**

When CBTC[*i*] is set, Channel *i* Chained buffer has terminated. LLI Fetch operation is disabled.

- **ERRx: Access Error [3:0]**

When ERR[*i*] is set, Channel *i* has detected an AHB Read or Write Error Access.

25.8.10 DMAC Channel Handler Enable Register

Name: DMAC_CHER

Address: 0x400C0028

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	KEEP3	KEEP2	KEEP1	KEEP0
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	SUSP3	SUSP2	SUSP1	SUSP0
7	6	5	4	3	2	1	0
–	–	–	–	ENA3	ENA2	ENA1	ENA0

- **ENAx: Enable [3:0]**

When set, a bit of the ENA field enables the relevant channel.

- **SUSPx: Suspend [3:0]**

When set, a bit of the SUSP field freezes the relevant channel and its current context.

- **KEEPx: Keep on [3:0]**

When set, a bit of the KEEP field resumes the current channel from an automatic stall state.

25.8.11 DMAC Channel Handler Disable Register

Name: DMAC_CHDR

Address: 0x400C002C

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
–	–	–	–	RES3	RES2	RES1	RES0
7	6	5	4	3	2	1	0
–	–	–	–	DIS3	DIS2	DIS1	DIS0

- **DISx: Disable [3:0]**

Write one to this field to disable the relevant DMAC Channel. The content of the FIFO is lost and the current AHB access is terminated. Software must poll DIS[3:0] field in the DMAC_CHSR register to be sure that the channel is disabled.

- **RESx: Resume [3:0]**

Write one to this field to resume the channel transfer restoring its context.

25.8.12 DMAC Channel Handler Status Register

Name: DMAC_CHSR

Address: 0x400C0030

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	STAL3	STAL2	STAL1	STAL0
23	22	21	20	19	18	17	16
–	–	–	–	EMPT3	EMPT2	EMPT1	EMPT0
15	14	13	12	11	10	9	8
–	–	–	–	SUSP3	SUSP2	SUSP1	SUSP0
7	6	5	4	3	2	1	0
–	–	–	–	ENA3	ENA2	ENA1	ENA0

- **ENAx: Enable [3:0]**

A one in any position of this field indicates that the relevant channel is enabled.

- **SUSPx: Suspend [3:0]**

A one in any position of this field indicates that the channel transfer is suspended.

- **EMPTx: Empty [3:0]**

A one in any position of this field indicates that the relevant channel is empty.

- **STALx: Stalled [3:0]**

A one in any position of this field indicates that the relevant channel is stalling.

25.8.13 DMAC Channel x [x = 0..3] Source Address Register

Name: DMAC_SADDRx [x = 0..3]

Address: 0x400C003C [0], 0x400C0064 [1], 0x400C008C [2], 0x400C00B4 [3]

Access: Read/Write

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 Protection Mode Register” .

- **SADDR: Channel x Source Address**

This register must be aligned with the source transfer width.

25.8.14 DMAC Channel x [x = 0..3] Destination Address Register

Name: DMAC_DADDRx [x = 0..3]

Address: 0x400C0040 [0], 0x400C0068 [1], 0x400C0090 [2], 0x400C00B8 [3]

Access: Read/Write

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 Protection Mode Register” .

- **DADDR: Channel x Destination Address**

This register must be aligned with the destination transfer width.

25.8.15 DMAC Channel x [x = 0..3] Descriptor Address Register

Name: DMAC_DSCRx [x = 0..3]

Address: 0x400C0044 [0], 0x400C006C [1], 0x400C0094 [2], 0x400C00BC [3]

Access: Read/Write

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 Protection Mode Register” .

- **DSCR: Buffer Transfer Descriptor Address**

This address is word aligned.

25.8.16 DMAC Channel x [x = 0..3] Control A Register

Name: DMAC_CTRLAx [x = 0..3]

Address: 0x400C0048 [0], 0x400C0070 [1], 0x400C0098 [2], 0x400C00C0 [3]

Access: Read/Write

31	30	29	28	27	26	25	24
DONE	–	DST_WIDTH		–	–	SRC_WIDTH	

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
BTSIZE							

7	6	5	4	3	2	1	0
BTSIZE							

This register can only be written if the WPEN bit is cleared in “DMAC Write Protection Mode Register” on page 506.

- **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 cleared, the DMAC module is automatically disabled when the relevant channel is enabled.

- **SRC_WIDTH: Transfer Width for the Source**

Value	Name	Description
00	BYTE	The transfer size is set to 8-bit width
01	HALF_WORD	The transfer size is set to 16-bit width
1X	WORD	The transfer size is set to 32-bit width

- **DST_WIDTH: Transfer Width for the Destination**

Value	Name	Description
00	BYTE	The transfer size is set to 8-bit width
01	HALF_WORD	The transfer size is set to 16-bit width
1X	WORD	The transfer size is set to 32-bit width

- **DONE: Current Descriptor Stop Command and Transfer Completed Memory Indicator**

0: The transfer is performed.

1: If SOD bit in DMAC_CFG is set to true, then the DMAC is automatically disabled when an LLI updates the content of this register.

The DONE bit is written back to memory at the end of the current descriptor transfer.

25.8.17 DMAC Channel x [x = 0..3] Control B Register

Name: DMAC_CTRLBx [x = 0..3]

Address: 0x400C004C [0], 0x400C0074 [1], 0x400C009C [2], 0x400C00C4 [3]

Access: Read/Write

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 Protection Mode Register” .

- **SRC_DSCR: Source Address Descriptor**

0 (FETCH_FROM_MEM): Source address is updated when the descriptor is fetched from the memory.

1 (FETCH_DISABLE): Buffer Descriptor Fetch operation is disabled for the source.

- **DST_DSCR: Destination Address Descriptor**

0 (FETCH_FROM_MEM): Destination address is updated when the descriptor is fetched from the memory.

1 (FETCH_DISABLE): Buffer Descriptor Fetch operation is disabled for the destination.

- **FC: Flow Control**

This field defines which device controls the size of the buffer transfer, also referred to as the Flow Controller.

Value	Name	Description
00	MEM2MEM_DMA_FC	Memory-to-Memory Transfer DMAC is flow controller
01	MEM2PER_DMA_FC	Memory-to-Peripheral Transfer DMAC is flow controller
10	PER2MEM_DMA_FC	Peripheral-to-Memory Transfer DMAC is flow controller
11	PER2PER_DMA_FC	Peripheral-to-Peripheral Transfer DMAC is flow controller

- **SRC_INCR: Incrementing, Decrementing or Fixed Address for the Source**

Value	Name	Description
00	INCREMENTING	The source address is incremented
01	DECREMENTING	The source address is decremented
10	FIXED	The source address remains unchanged

- **DST_INCR: Incrementing, Decrementing or Fixed Address for the Destination**

Value	Name	Description
00	INCREMENTING	The destination address is incremented
01	DECREMENTING	The destination address is decremented
10	FIXED	The destination address remains unchanged

- **IEN: Interrupt Enable Not**

0: When the buffer transfer is completed, the BTCx flag is set in the DMAC_EBCISR. This bit is active low.

1: When the buffer transfer is completed, the BTCx flag is not set.

If this bit is cleared, when the buffer transfer is completed, the BTCx flag is set in the DMAC_EBCISR.

25.8.18 DMAC Channel x [x = 0..3] Configuration Register

Name: DMAC_CFGx [x = 0..3]

Address: 0x400C0050 [0], 0x400C0078 [1], 0x400C00A0 [2], 0x400C00C8 [3]

Access: Read/Write

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 Protection Mode Register” on page 506

- **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 the DMAC_CTRLAx.DONE bit.

1 (ENABLE): STOP ON DONE activated, the DMAC module is automatically disabled if DMAC_CTRLAx.DONE bit is set.

- **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.

25.8.19 DMAC Write Protection Mode Register

Name: DMAC_WPMR

Address: 0x400C01E4

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 0x444D41 (“DMA” in ASCII).

1: Enables the Write Protection if WPKEY corresponds to 0x444D41 (“DMA” in ASCII).

See Section 25.6.7 ”Register Write Protection” for the list of registers that can be write-protected.

- **WPKEY: Write Protection Key**

Value	Name	Description
0x444D41	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

25.8.20 DMAC Write Protection Status Register

Name: DMAC_WPSR

Address: 0x400C01E8

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
WPVSRC							

15	14	13	12	11	10	9	8
WPVSRC							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WPVS

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of the DMAC_WPSR.

1: A write protection violation has occurred since the last read of the DMAC_WPSR. 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**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

26. Peripheral DMA Controller (PDC)

26.1 Description

The Peripheral DMA Controller (PDC) transfers data between on-chip serial peripherals and the target 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 the current transfer and one set (pointer, counter) for the next transfer. The bidirectional channel user interface contains four 32-bit memory pointers and four 16-bit counters. Each set (pointer, counter) is used by the current transmit, next transmit, current receive and next receive.

Using the PDC decreases processor overhead by reducing its intervention during the transfer. This lowers significantly the number of clock cycles required for a data transfer, improving microcontroller performance.

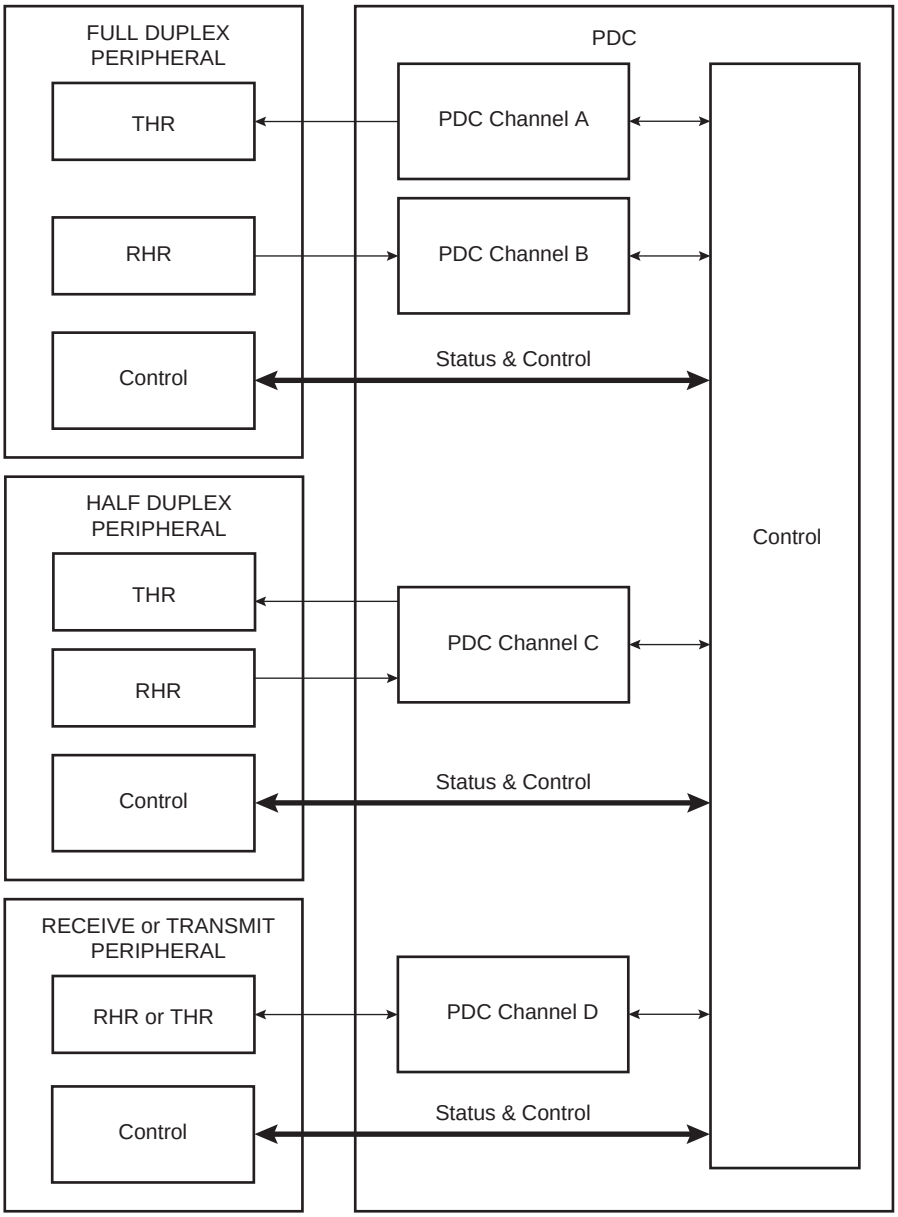
To launch a transfer, the peripheral triggers its associated PDC channels by using transmit and receive signals. When the programmed data is transferred, an end of transfer interrupt is generated by the peripheral itself.

26.2 Embedded Characteristics

- Performs Transfers to/from APB Communication Serial Peripherals
- Supports Half-duplex and Full-duplex Peripherals

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 the current and next transfer, whether it is for read (transmit) or write (receive). 16-bit counters define the size of the current and next transfers. It is possible, at any moment, to read the number of transfers remaining 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.

The peripheral where a PDC transfer is configured must have its peripheral clock enabled. The peripheral clock must be also enabled to access the PDC register set associated to this peripheral.

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 to a receive area and to a transmit area, respectively, in the target 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 the current transfer and the one for the 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 the 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 the next counter is zero, the channel stops transferring data and sets the appropriate flag. If the next counter value is greater than zero, the values of the next pointer/next counter are copied into the current pointer/current counter and the channel resumes the transfer, whereas next pointer/next counter get zero/zero as values.At the end of this transfer, the PDC channel sets the appropriate flags in the Peripheral Status register.

The following list gives an overview of how status register flags behave depending on the counters' values:

- ENDRX flag is set when the PDC Receive Counter Register (PERIPH_RCR) reaches zero.
- RXBUFF flag is set when both PERIPH_RCR and the PDC Receive Next Counter Register (PERIPH_RNCR) reach zero.

- ENDTX flag is set when the PDC Transmit Counter Register (PERIPH_TCR) reaches zero.
- TXBUFE flag is set when both PERIPH_TCR and the PDC Transmit Next Counter Register (PERIPH_TNCR) reach zero.

These status flags are described in the Transfer Status Register (PERIPH_PTSR).

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 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 transfers the data to the Transmit Holding register (THR) of its associated peripheral. The same peripheral sends data depending on 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 returns flags to the peripheral. All these flags are only visible in the peripheral's Status register.

Depending on whether the peripheral is half- or full-duplex, the flags belong to either one single channel or two different channels.

26.4.5.1 Receive Transfer End

The receive transfer end flag is set when PERIPH_RCR reaches zero and the last data has been transferred to memory.

This flag is reset by writing a non-zero value to PERIPH_RCR or PERIPH_RNCR.

26.4.5.2 Transmit Transfer End

The transmit transfer end flag is set when PERIPH_TCR reaches zero and the last data has been written to the peripheral THR.

This flag is reset by writing a non-zero value to PERIPH_TCR or PERIPH_TNCR.

26.4.5.3 Receive Buffer Full

The receive buffer full flag is set when PERIPH_RCR reaches zero, with PERIPH_RNCR also set to zero and the last data transferred to memory.

This flag is reset by writing a non-zero value to PERIPH_TCR or PERIPH_TNCR.

26.4.5.4 Transmit Buffer Empty

The transmit buffer empty flag is set when PERIPH_TCR reaches zero, with PERIPH_TNCR also set to zero and the last data written to peripheral THR.

This flag is reset by writing a non-zero value to PERIPH_TCR or PERIPH_TNCR.

26.5 Peripheral DMA Controller (PDC) User Interface

Table 26-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Receive Pointer Register	PERIPH(1)_RPR	Read/Write	0
0x04	Receive Counter Register	PERIPH_RCR	Read/Write	0
0x08	Transmit Pointer Register	PERIPH_TPR	Read/Write	0
0x0C	Transmit Counter Register	PERIPH_TCR	Read/Write	0
0x10	Receive Next Pointer Register	PERIPH_RNPR	Read/Write	0
0x14	Receive Next Counter Register	PERIPH_RNCR	Read/Write	0
0x18	Transmit Next Pointer Register	PERIPH_TNPR	Read/Write	0
0x1C	Transmit Next Counter Register	PERIPH_TNCR	Read/Write	0
0x20	Transfer Control Register	PERIPH_PTCR	Write-only	–
0x24	Transfer Status Register	PERIPH_PTSR	Read-only	0

Note: 1. PERIPH: Ten registers are mapped in the peripheral memory space at the same offset. These can be defined by the user depending on 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 the 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 the 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 the 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 the 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 the 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 the 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. Static Memory Controller (SMC)

27.1 Description

The External Bus Interface (EBI) is designed to ensure the successful data transfer between several external devices and the ARM-based microcontroller. The Static Memory Controller (SMC) is part of the EBI.

The SMC handles 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 the external memory devices or peripheral devices. It has 4 chip selects, a 24-bit address bus, and an 8-bit data bus. Separate read and write control signals allow for direct memory and peripheral interfacing. Read and write signal waveforms are fully adjustable.

The SMC can manage wait requests from external devices to extend the current access. The SMC is provided with an automatic Slow clock mode. In Slow clock mode, it switches from user-programmed waveforms to slow-rate specific waveforms on read and write signals. The SMC supports asynchronous burst read in Page mode access for page sizes up to 32 bytes.

The external data bus can be scrambled/unscrambled by means of user keys.

27.2 Embedded Characteristics

- Four Chip Selects Available
- 16-Mbyte Address Space per Chip Select
- 8-bit Data Bus
- Zero Wait State Scrambling/Unscrambling Function with User Key
- Word, Halfword, Byte Transfers
- Programmable Setup, Pulse And Hold Time for Read Signals per Chip Select
- Programmable Setup, Pulse And Hold Time for Write Signals per Chip Select
- Programmable Data Float Time per Chip Select
- External Wait Request
- Automatic Switch to Slow Clock Mode
- Asynchronous Read in Page Mode Supported: Page Size Ranges from 4 to 32 Bytes
- Register Write Protection

27.3 I/O Lines Description

Table 27-1. I/O Line Description

Name	Description	Type	Active Level
NCS[3:0]	Static Memory Controller Chip Select Lines	Output	Low
NRD	Read Signal	Output	Low
NWE	Write Enable Signal	Output	Low
A[23:0]	Address Bus	Output	–
D[7:0]	Data Bus	I/O	–
NWAIT	External Wait Signal	Input	Low
NANDCS	NAND Flash Chip Select Line	Output	Low
NANDOE	NAND Flash Output Enable	Output	Low
NANDWE	NAND Flash Write Enable	Output	Low
NANDALE	NAND Flash Address Latch Enable	Output	–
NANDCLE	NAND Flash Command Latch Enable	Output	–

27.4 Multiplexed Signals

Table 27-2. Static Memory Controller (SMC) Multiplexed Signals

Multiplexed Signals		Related Function
A22	NANDCLE	NAND Flash Command Latch Enable
A21	NANDALE	NAND Flash Address Latch Enable

27.5 Product Dependencies

27.5.1 I/O Lines

The pins used for interfacing the SMC are multiplexed with the PIO lines. The programmer must first program the PIO controller to assign the SMC 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.

Table 27-3. I/O Lines

Instance	Signal	I/O Line	Peripheral
SMC	A0	PC18	A
SMC	A1	PC19	A
SMC	A2	PC20	A
SMC	A3	PC21	A
SMC	A4	PC22	A
SMC	A5	PC23	A
SMC	A6	PC24	A
SMC	A7	PC25	A
SMC	A8	PC26	A
SMC	A9	PC27	A

Table 27-3. I/O Lines

SMC	A10	PC28	A
SMC	A11	PC29	A
SMC	A12	PC30	A
SMC	A13	PC31	A
SMC	A14	PA18	C
SMC	A15	PA19	C
SMC	A16	PA20	C
SMC	A17	PA0	C
SMC	A18	PA1	C
SMC	A19	PA23	C
SMC	A20	PA24	C
SMC	A21/NANDALE	PC16	A
SMC	A22/NANDCLE	PC17	A
SMC	A23	PA25	C
SMC	D0	PC0	A
SMC	D1	PC1	A
SMC	D2	PC2	A
SMC	D3	PC3	A
SMC	D4	PC4	A
SMC	D5	PC5	A
SMC	D6	PC6	A
SMC	D7	PC7	A
SMC	NANDOE	PC9	A
SMC	NANDWE	PC10	A
SMC	NCS0	PC14	A
SMC	NCS1	PC15	A
SMC	NCS1	PD18	A
SMC	NCS2	PA22	C
SMC	NCS3	PC12	A
SMC	NCS3	PD19	A
SMC	NRD	PC11	A
SMC	NWAIT	PC13	A
SMC	NWE	PC8	A

27.5.2 Power Management

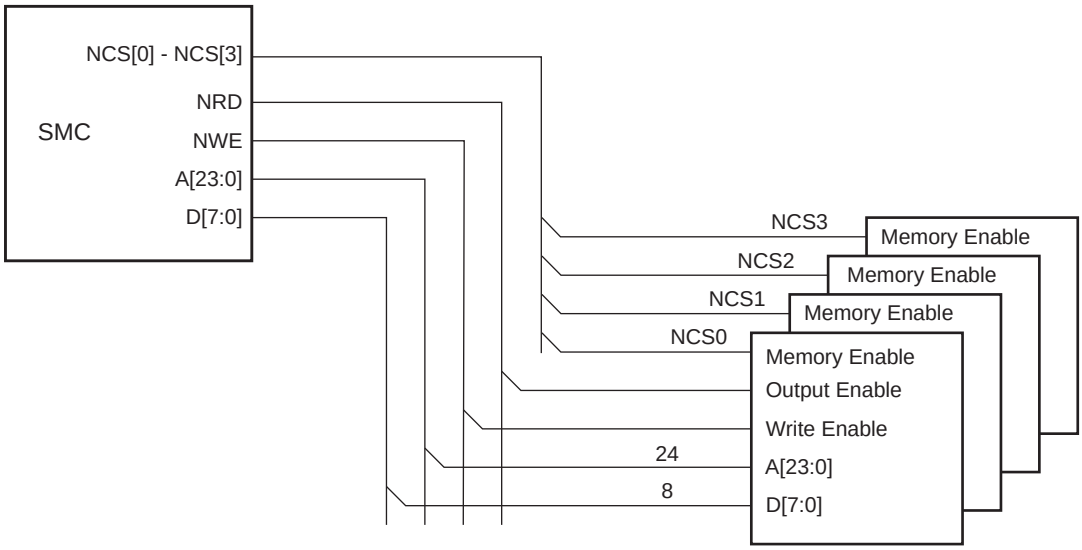
The SMC is clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the SMC clock.

27.6 External Memory Mapping

The SMC provides up to 24 address lines, A[23:0]. This allows each chip select line to address up to 16 Mbytes of memory.

If the physical memory device connected on one chip select is smaller than 16 Mbytes, it wraps around and appears to be repeated within this space. The SMC correctly handles any valid access to the memory device within the page (see Figure 27-1).

Figure 27-1. Memory Connections for Four External Devices



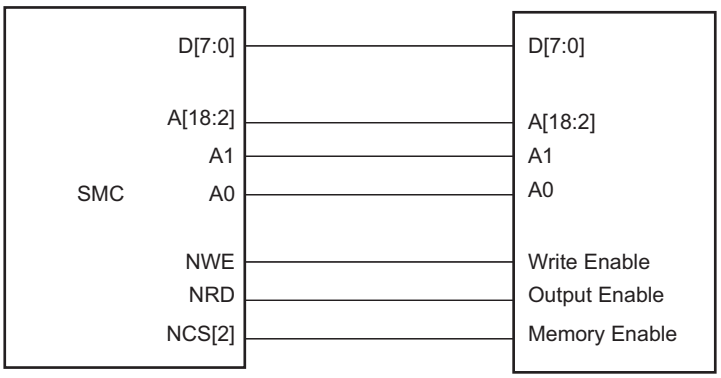
27.7 Connection to External Devices

27.7.1 Data Bus Width

The data bus width is 8 bits.

Figure 27-2 shows how to connect a 512-Kbyte × 8-bit memory on NCS2.

Figure 27-2. Memory Connection for an 8-bit Data Bus



27.7.2 NAND Flash Support

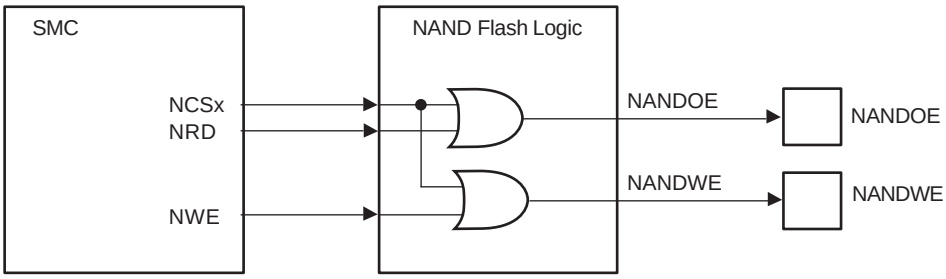
The SMC integrates circuitry that interfaces to NAND Flash devices.

The NAND Flash logic is driven by the SMC. Configuration is done via the SMC_NFCSx field in the CCFG_SMCNFCS register in the Bus Matrix. For details on this register, refer to the section “Bus Matrix (MATRIX)” of this datasheet. The external NAND Flash device is accessed via the address space reserved for the chip select programmed.

The user can connect up to four NAND Flash devices with separate chip selects.

The NAND Flash logic drives the read and write command signals of the SMC on the NANDOE and NANDWE signals when the NCSx programmed is active. NANDOE and NANDWE are disabled as soon as the transfer address fails to lie in the NCSx programmed address space.

Figure 27-3. NAND Flash Signal Multiplexing on SMC Pins



Notes: 1. NCSx is active when CCFG_SMCNFCS.SMC_NFCSx=1.

2. When the NAND Flash logic is activated, (SMC_NFCSx=1), the NWE pin can be used only in Peripheral mode (NWE function). If the NWE function is not used for other external memories (SRAM, LCD), it must be configured in one of the following modes:
   – PIO Input with pull-up enabled (default state after reset)
   – PIO Output set at level 1

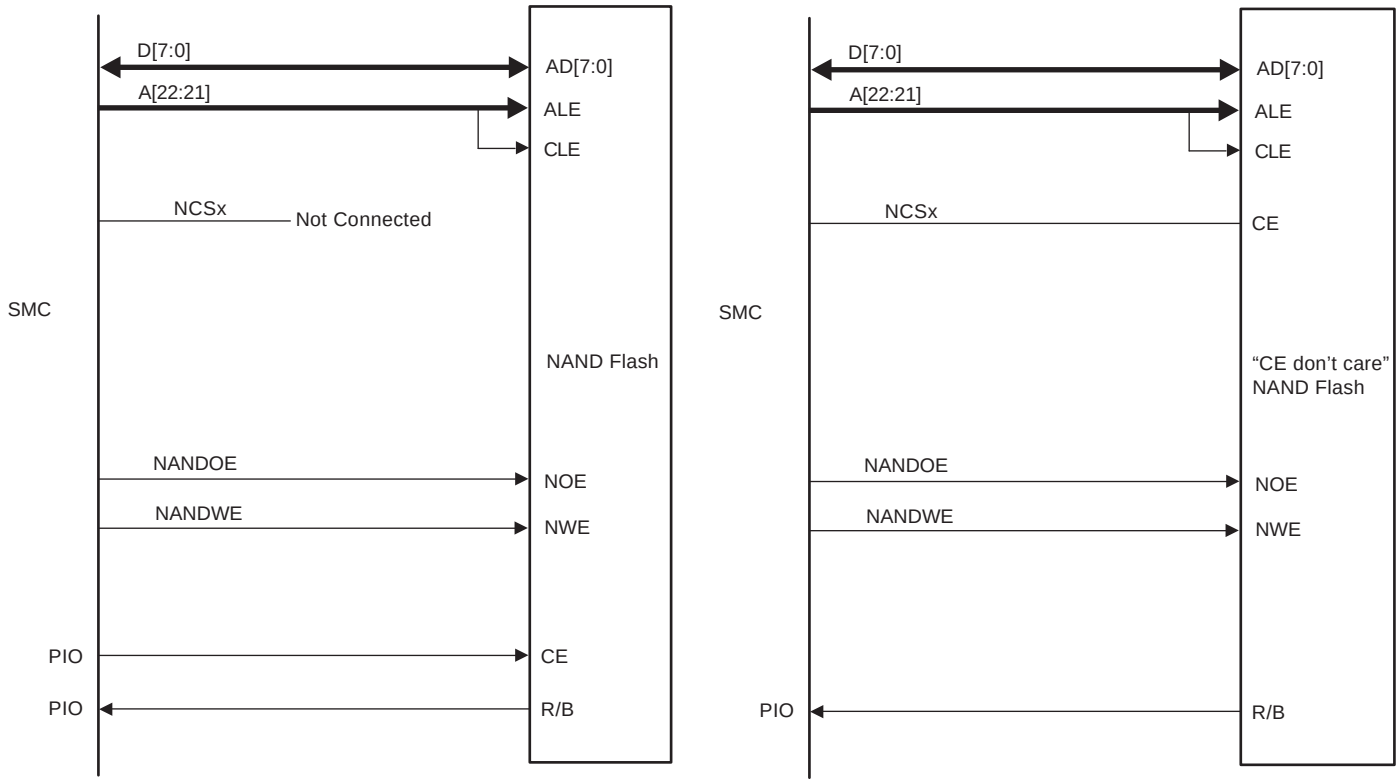
The address latch enable and command latch enable signals on the NAND Flash device are driven by address bits A22 and A21of the address bus. Any bit of the address bus can also be used for this purpose. The command, address or data words on the data bus of the NAND Flash device use their own addresses within the NCSx address space (configured in the register CCFG_SMCNFCS in the Bus Matrixe). The chip enable (CE) signal of the device and the ready/busy (R/B) signals are connected to PIO lines. The CE signal then remains asserted even when NAND Flash chip select is not selected, preventing the device from returning to Standby mode. The NANDCS output signal should be used in accordance with the external NAND Flash device type.

Two types of CE behavior exist depending on the NAND Flash device:

- Standard NAND Flash devices require that the CE pin remains asserted low continuously during the read busy period to prevent the device from returning to Standby mode. Since the SMC asserts the NCSx signal high, it is necessary to connect the CE pin of the NAND Flash device to a GPIO line, in order to hold it low during the busy period preceding data read out.
- This restriction has been removed for “CE don’t care” NAND Flash devices. The NCSx signal can be directly connected to the CE pin of the NAND Flash device.

Figure 27-4 illustrates both topologies: Standard and “CE don’t care” NAND Flash.

Figure 27-4. Standard and “CE don’t care” NAND Flash Application Examples



27.8 Application Example

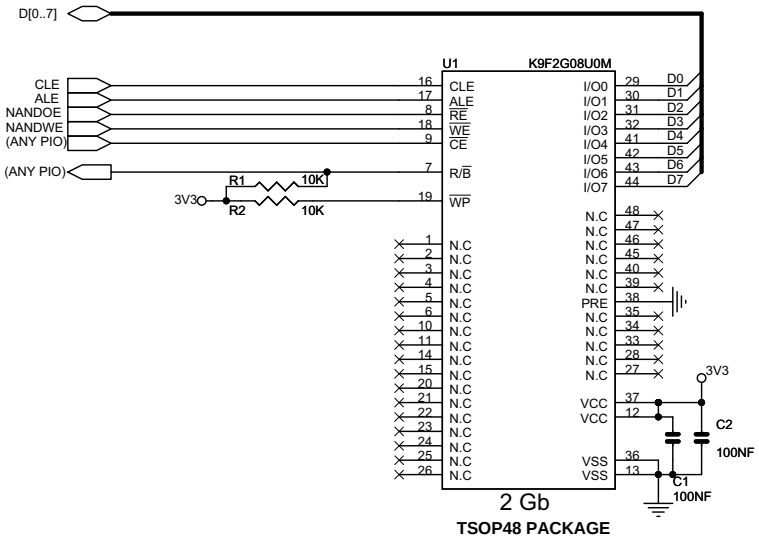
27.8.1 Implementation Examples

Hardware configurations are given for illustration only. The user should refer to the manufacturer web site to check for memory device availability.

For hardware implementation examples, refer to the evaluation kit schematics for this microcontroller, which show examples of a connection to an LCD module and NAND Flash.

27.8.1.1 8-bit NAND Flash

Hardware Configuration



Software Configuration

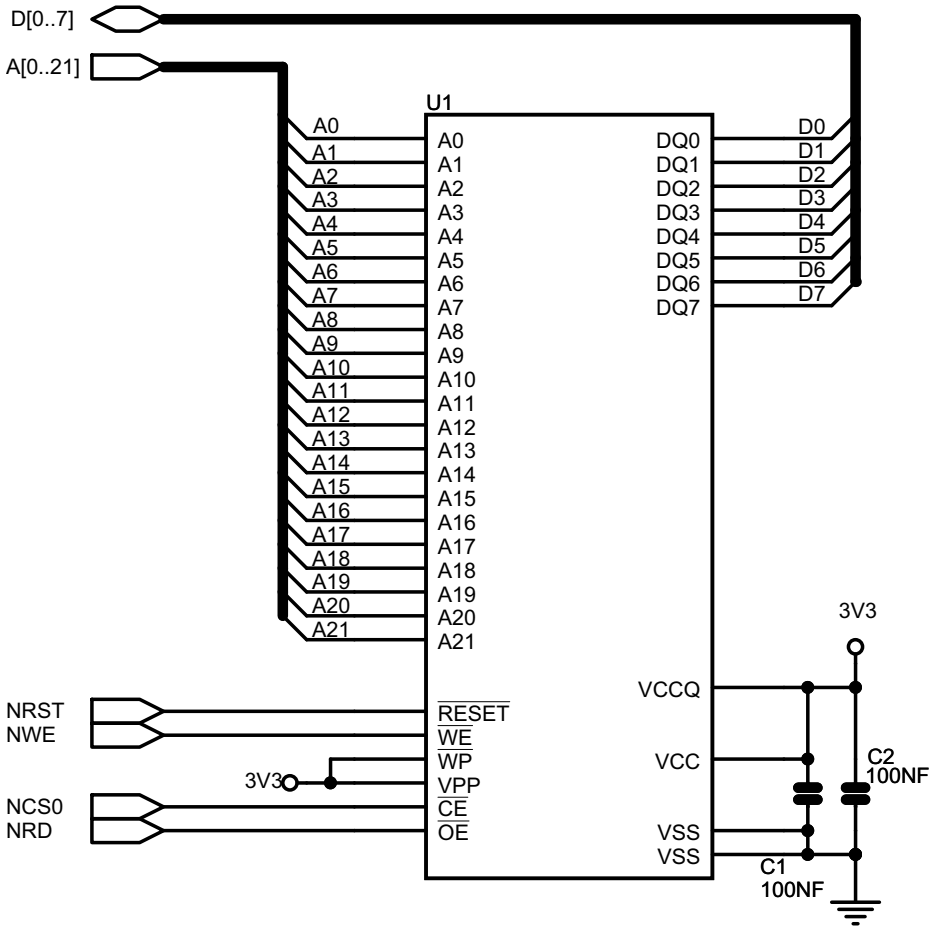
Perform the following configuration:

1. Select the chip select used to drive the NAND Flash by setting the bit CCFG_SMCNFCS.SMC_NFCSx.
2. Reserve A21 / A22 for ALE / CLE functions. Address and Command Latches are controlled by setting the address bits A21 and A22, respectively, during accesses.
3. NANDOE and NANDWE signals are multiplexed with PIO lines. Thus, the dedicated PIOs must be programmed in Peripheral mode in the PIO controller.
4. Configure a PIO line as an input to manage the Ready/Busy signal.
5. Configure SMC CS3 Setup, Pulse, Cycle and Mode according to NAND Flash timings, the data bus width and the system bus frequency.

In this example, the NAND Flash is not addressed as a “CE don’t care”. To address it as a “CE don’t care”, connect NCS3 (if SMC_NFCS3 is set) to the NAND Flash CE.

27.8.1.2 NOR Flash

Hardware Configuration



Software Configuration

Configure the SMC CS0 Setup, Pulse, Cycle and Mode depending on Flash timings and system bus frequency.

27.9 Standard Read and Write Protocols

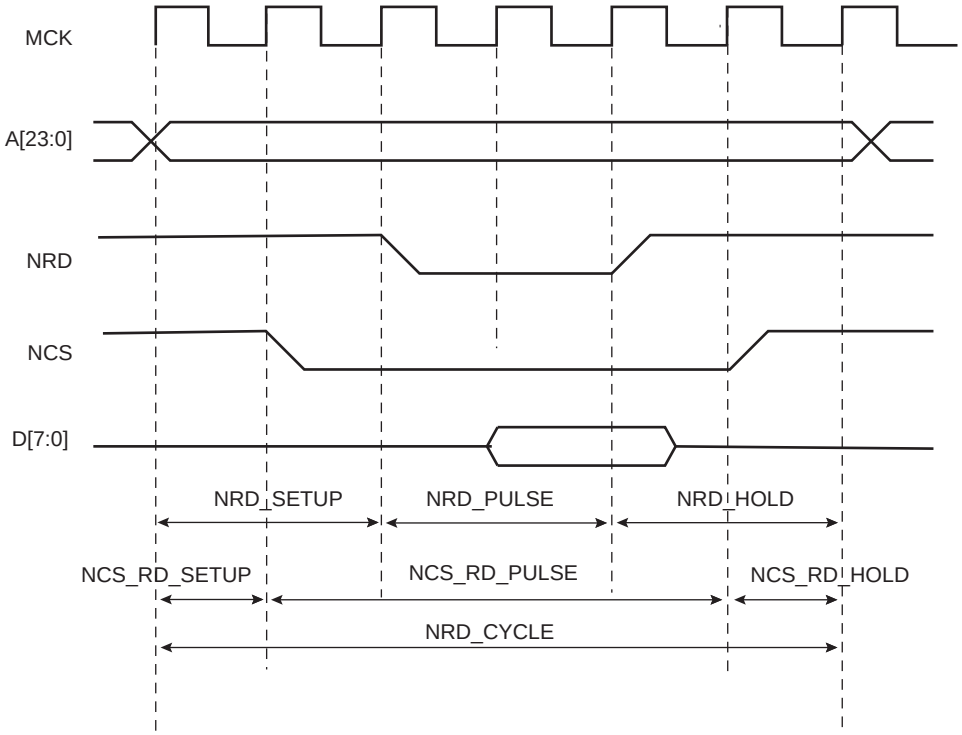
In the following sections, NCS represents one of the NCS[0..3] chip select lines.

27.9.1 Read Waveforms

The read cycle is shown in Figure 27-5.

The read cycle starts with the address setting on the memory address bus.

Figure 27-5. Standard Read Cycle



27.9.1.1 NRD Waveform

The NRD signal is characterized by a setup timing, a pulse width and a hold timing.

- NRD_SETUP— NRD setup time is defined as the setup of address before the NRD falling edge;
- NRD_PULSE—NRD pulse length is the time between NRD falling edge and NRD rising edge;
- NRD_HOLD—NRD hold time is defined as the hold time of address after the NRD rising edge.

27.9.1.2 NCS Waveform

The NCS signal can be divided into a setup time, pulse length and hold time:

- NCS_RD_SETUP—NCS setup time is defined as the setup time of address before the NCS falling edge.
- NCS_RD_PULSE—NCS pulse length is the time between NCS falling edge and NCS rising edge;
- NCS_RD_HOLD—NCS hold time is defined as the hold time of address after the NCS rising edge.

27.9.1.3 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 defined as:

NRD_CYCLE = NRD_SETUP + NRD_PULSE + NRD_HOLD,

as well as

NRD_CYCLE = 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. The NRD_CYCLE field is common to both the NRD and NCS signals, thus the timing period is of the same duration.

NRD_CYCLE, NRD_SETUP, and NRD_PULSE implicitly define the NRD_HOLD value as:

NRD_HOLD = NRD_CYCLE - NRD SETUP - NRD PULSE

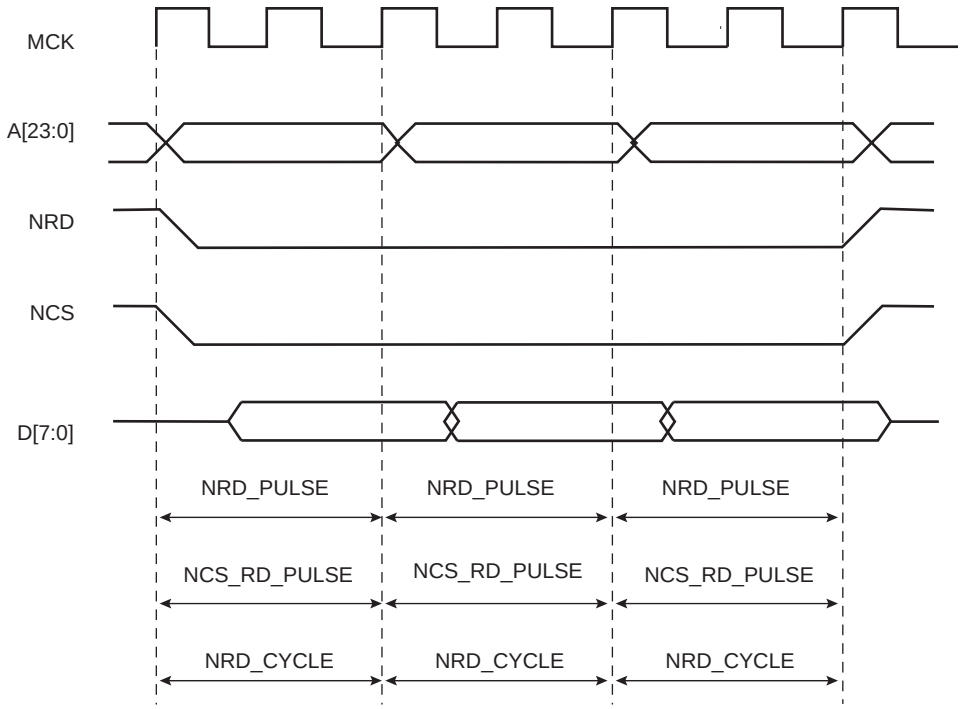
NRD_CYCLE, NCS_RD_SETUP, and NCS_RD_PULSE implicitly define the NCS_RD_HOLD value as:

NCS_RD_HOLD = NRD_CYCLE - NCS_RD_SETUP - NCS_RD_PULSE

27.9.1.4 Null Delay Setup and Hold

If null setup and hold parameters are programmed for NRD and/or NCS, NRD and NCS remain active continuously in case of consecutive read cycles in the same memory (see Figure 27-6).

Figure 27-6. No Setup, No Hold on NRD and NCS Read Signals



27.9.1.5 Null Pulse

Programming a null pulse is not permitted. The pulse must be at least set to 1. A null value leads to unpredictable behavior.

27.9.2 Read Mode

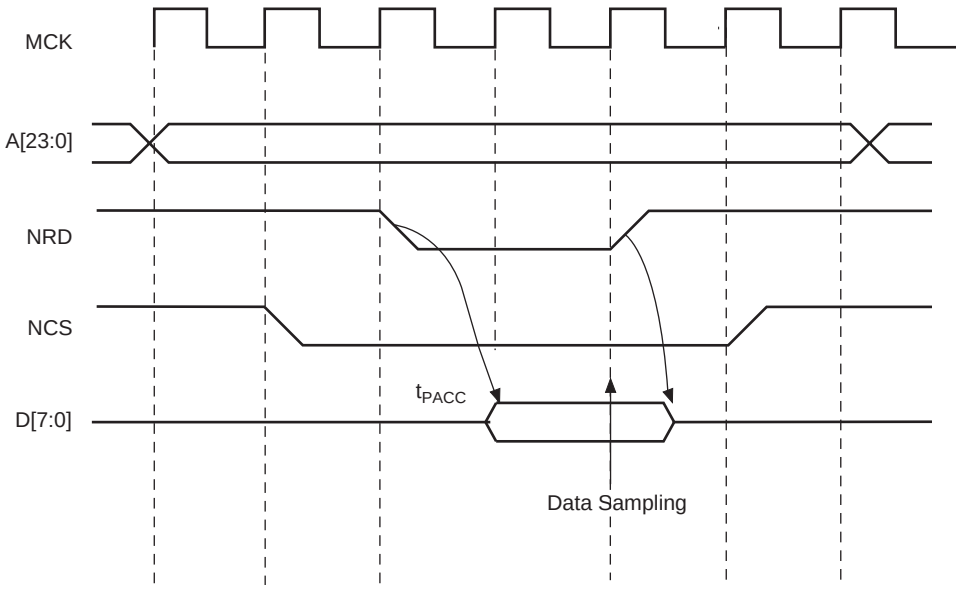
As NCS and NRD waveforms are defined independently of one other, the SMC needs to know when the read data is available on the data bus. The SMC does not compare NCS and NRD timings to know which signal rises first. The READ_MODE bit in the SMC_MODE register of the corresponding chip select indicates which signal of NRD and NCS controls the read operation.

27.9.2.1 Read is Controlled by NRD (SMC_MODE.READ_MODE = 1):

Figure 27-7 shows the waveforms of a read operation of a typical asynchronous RAM. The read data is available t_{PACC} after the falling edge of NRD, and turns to 'Z' after the rising edge of NRD. In this case, SMC_MODE.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.

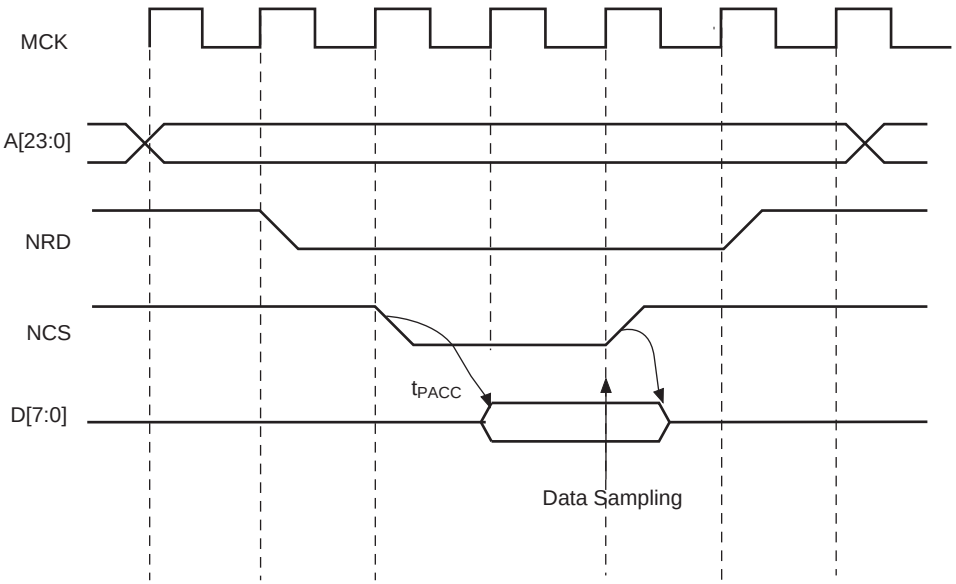
Figure 27-7. SMC_MODE.READ_MODE = 1: Data is sampled by SMC before the rising edge of NRD



27.9.2.2 Read is Controlled by NCS (SMC_MODE.READ_MODE = 0)

Figure 27-8 shows the typical read cycle of an LCD module. The read data is valid t_{PACC} after the falling edge of the NCS signal and remains valid until the rising edge of NCS. Data must be sampled when NCS is raised. In this case, the SMC_MODE.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 27-8. SMC_MODE.READ_MODE = 0: Data is Sampled by SMC Before the Rising Edge of NCS



27.9.3 Write Waveforms

The write protocol is similar to the read protocol. It is depicted in Figure 27-9. The write cycle starts with the address setting on the memory address bus.

27.9.3.1 NWE Waveforms

The NWE signal is characterized by a setup timing, a pulse width and a hold timing.

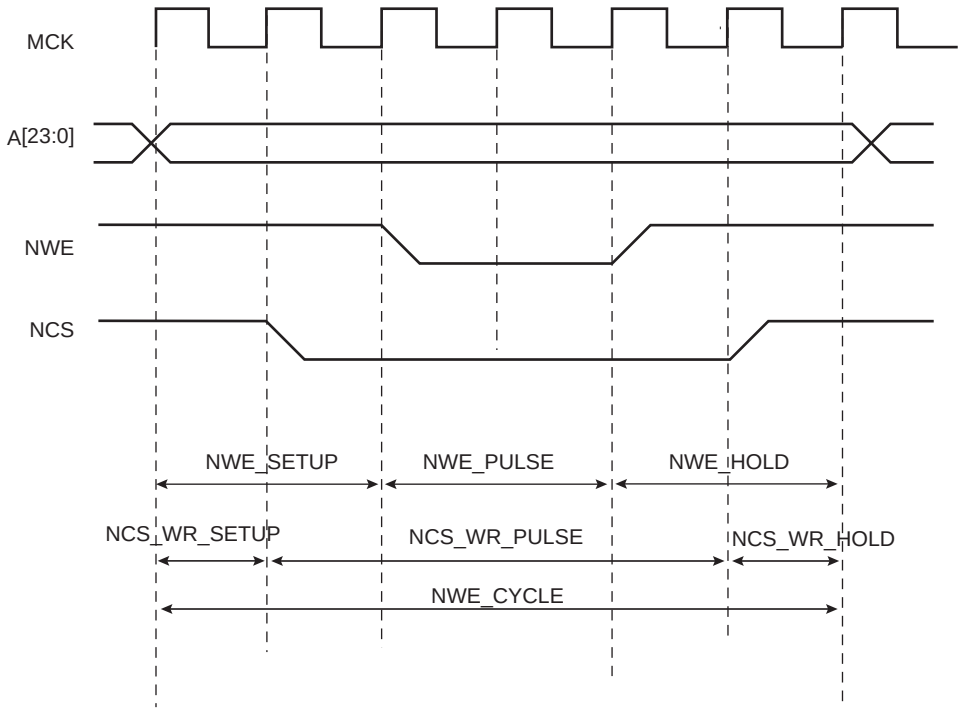
- NWE_SETUP—the NWE setup time is defined as the setup of address and data before the NWE falling edge;
- NWE_PULSE—the NWE pulse length is the time between NWE falling edge and NWE rising edge;
- NWE_HOLD—the NWE hold time is defined as the hold time of address and data after the NWE rising edge.

27.9.3.2 NCS Waveforms

The NCS signal waveforms in write operation are not the same that those applied in read operations, but are separately defined:

- NCS_WR_SETUP—the NCS setup time is defined as the setup time of address before the NCS falling edge.
- NCS_WR_PULSE—the NCS pulse length is the time between NCS falling edge and NCS rising edge;
- NCS_WR_HOLD—the NCS hold time is defined as the hold time of address after the NCS rising edge.

Figure 27-9. Write Cycle



27.9.3.3 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 defined as:

NWE_CYCLE = NWE_SETUP + NWE_PULSE + NWE_HOLD,

as well as

NWE_CYCLE = 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. The NWE_CYCLE field is common to both the NWE and NCS signals, thus the timing period is of the same duration.

NWE_CYCLE, NWE_SETUP, and NWE_PULSE implicitly define the NWE_HOLD value as:

NWE_HOLD = NWE_CYCLE - NWE_SETUP - NWE_PULSE

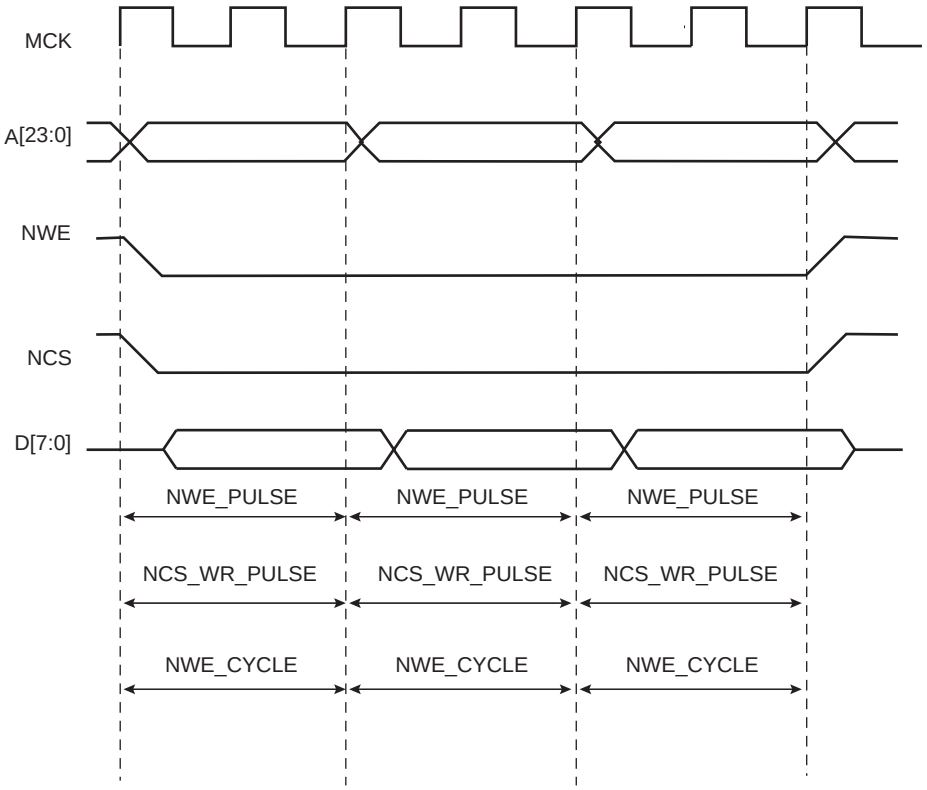
NWE_CYCLE, NCS_WR_SETUP, and NCS_WR_PULSE implicitly define the NCS_WR_HOLD value as:

NCS_WR_HOLD = NWE_CYCLE - NCS_WR_SETUP - NCS_WR_PULSE

27.9.3.4 Null Delay Setup and Hold

If null setup parameters are programmed for NWE and/or NCS, NWE and/or NCS remain active continuously in case of consecutive write cycles in the same memory (see Figure 27-10). However, for devices that perform write operations on the rising edge of NWE or NCS, such as SRAM, either a setup or a hold must be programmed.

Figure 27-10. Null Setup and Hold Values of NCS and NWE in Write Cycle



27.9.3.5 Null Pulse

Programming null pulse is not permitted. Pulse must be at least set to 1. A null value leads to unpredictable behavior.

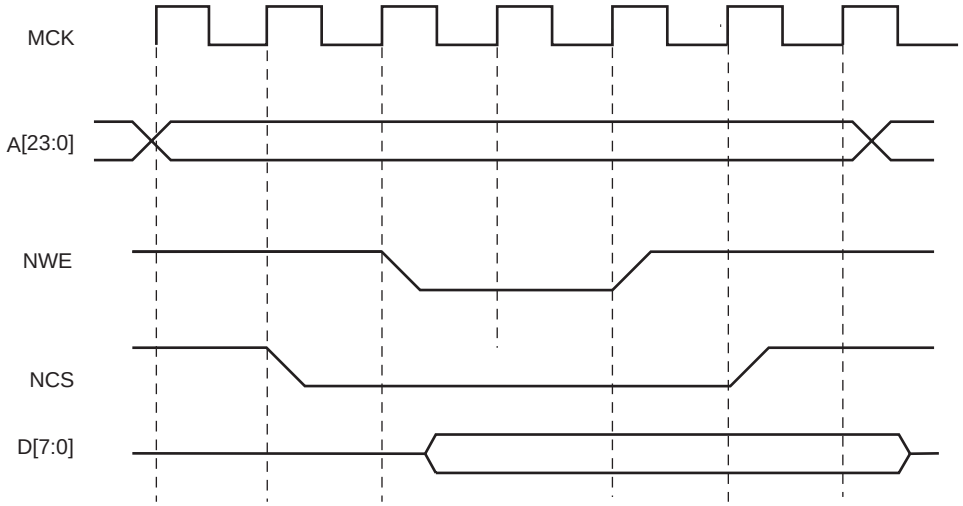
27.9.4 Write Mode

The bit WRITE_MODE in the SMC_MODE register of the corresponding chip select indicates which signal controls the write operation.

27.9.4.1 Write is Controlled by NWE (SMC.MODE.WRITE_MODE = 1):

Figure 27-11 shows the waveforms of a write operation with SMC_MODE.WRITE_MODE set . The data is put on the bus during the pulse and hold steps of the NWE signal. The internal data buffers are switched to Output mode after the NWE_SETUP time, and until the end of the write cycle, regardless of the programmed waveform on NCS.

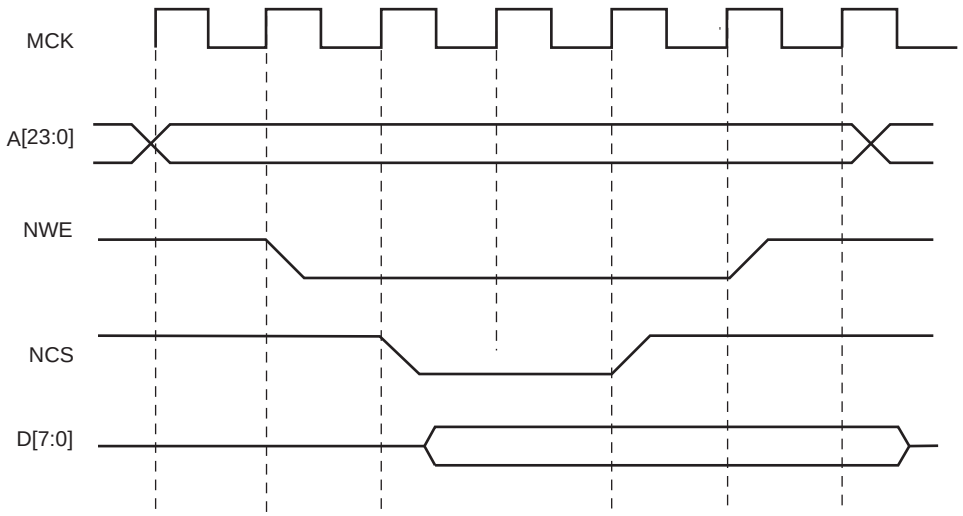
Figure 27-11. SMC_MODE.WRITE_MODE = 1. Write Operation is Controlled by NWE



27.9.4.2 Write is Controlled by NCS (SMC.MODE.WRITE_MODE = 0)

Figure 27-12 shows the waveforms of a write operation with SMC_MODE.WRITE_MODE cleared. The data is put on the bus during the pulse and hold steps of the NCS signal. The internal data buffers are switched to Output mode after the NCS_WR_SETUP time, and until the end of the write cycle, regardless of the programmed waveform on NWE.

Figure 27-12. WRITE_MODE = 0. Write Operation is Controlled by NCS



27.9.5 Register Write Protection

To prevent any single software error that may corrupt SMC behavior, the registers listed below can be write-protected by setting the WPEN bit in the SMC Write Protection Mode register (SMC_WPMR).

If a write access in a write-protected register is detected, the WPVS flag in the SMC Write Protection Status register (SMC_WPSR) is set and the field WPVSRC indicates in which register the write access has been attempted.

The WPVS flag is automatically cleared after reading the SSMC_WPSR.

The following registers can be write-protected:

- "SMC Setup Register"
- "SMC Pulse Register"
- "SMC Cycle Register"
- "SMC Mode Register"

27.9.6 Coding Timing Parameters

All timing parameters are defined for one chip select and are grouped together in one 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 27-4 shows how the timing parameters are coded and their permitted range.

Table 27-4. Coding and Range of Timing Parameters

Coded Value	Number of Bits	Effective Value	Permitted Range	
			Coded Value	Effective Value
setup [5:0]	6	128 × setup[5] + setup[4:0]	0 ≤ 31	0 ≤ 128+31
pulse [6:0]	7	256 × pulse[6] + pulse[5:0]	0 ≤ 63	0 ≤ 256+63
cycle [8:0]	9	256 × cycle[8:7] + cycle[6:0]	0 ≤ 127	0 ≤ 256+127 0 ≤ 512+127 0 ≤ 768+127

27.9.7 Reset Values of Timing Parameters

Table 27-5 gives the default value of timing parameters at reset.

Table 27-5. Reset Values of Timing Parameters

Parameter	Reset Value	Definition
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 operations continue for 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.

27.9.8 Usage Restriction

The SMC does not check the validity of the user-programmed parameters. If the sum of SETUP and PULSE parameters is larger than the corresponding CYCLE parameter, this leads to unpredictable behavior of the SMC.

- For read operations:

  Null but positive setup and hold of address and NRD and/or NCS can not be guaranteed at the memory interface because of the propagation delay of 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.

- For write operations:

  If a null hold value is programmed on NWE, the SMC can guarantee a positive hold of address and NCS signal after the rising edge of NWE. This is true for SMC_MODE.WRITE_MODE = 1 only. See Section 27.11.2 ”Early Read Wait State”.

- 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.

27.10 Scrambling/Unscrambling Function

The external data bus can be scrambled to prevent recovery of intellectual property data located in off-chip memories by means of data analysis at the package pin level of either the microcontroller or the memory device.

The scrambling and unscrambling are performed on-the-fly without additional wait states.

The scrambling/unscrambling function can be enabled or disabled by configuring the CSxSE bits in the SMC Off-Chip Memory Scrambling Register (SMC_OCMS).

When multiple chip selects are handled, the scrambling function per chip select is configurable using the CSxSE bits in the SMC_OCMS register.

The scrambling method depends on two user-configurable key registers, SMC_KEY1 and SMC_KEY2. These key registers cannot be read. They can be written once after a system reset.

The scrambling user key or the seed for key generation 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.

27.11 Automatic Wait States

Under certain circumstances, the SMC automatically inserts idle cycles between accesses to avoid bus contention or operation conflict.

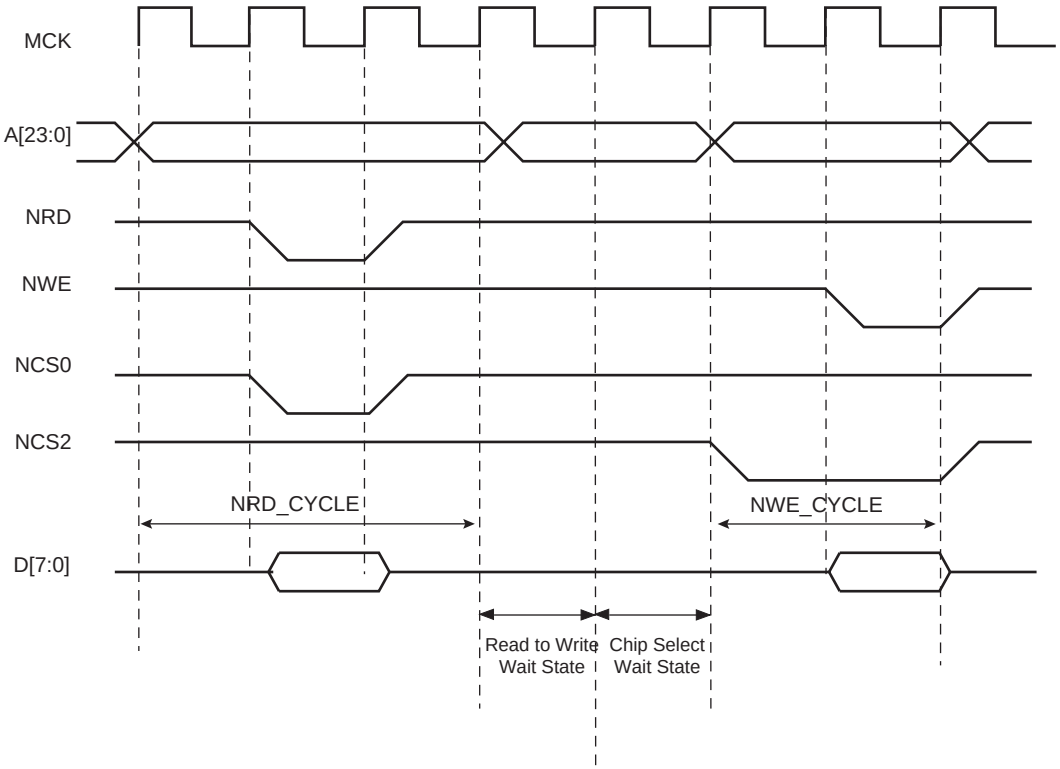
27.11.1 Chip Select Wait States

The SMC always inserts an idle cycle between two transfers on separate chip selects. This idle cycle ensures that there is no bus contention between the de-activation of one device and the activation of the next one.

During chip select wait state, all control lines are turned inactive: NWR, NCS[0..3], NRD lines are all set to 1.

Figure 27-13 illustrates a chip select wait state between access on chip select 0 and chip select 2.

Figure 27-13. Chip Select Wait State between a Read Access on NCS0 and a Write Access on NCS2



27.11.2 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 27-14).
- in NCS Write controlled mode (SMC_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 27-15). 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 (SMC_MODE.WRITE_MODE = 1) and if there is no hold timing (NWE_HOLD = 0), the feedback of the write control signal is used to control address, data, and chip select lines. If the external

write control signal is not inactivated as expected due to load capacitances, an Early Read Wait State is inserted and address, data and control signals are maintained one more cycle. See Figure 27-16.

Figure 27-14. Early Read Wait State: Write with No Hold Followed by Read with No Setup

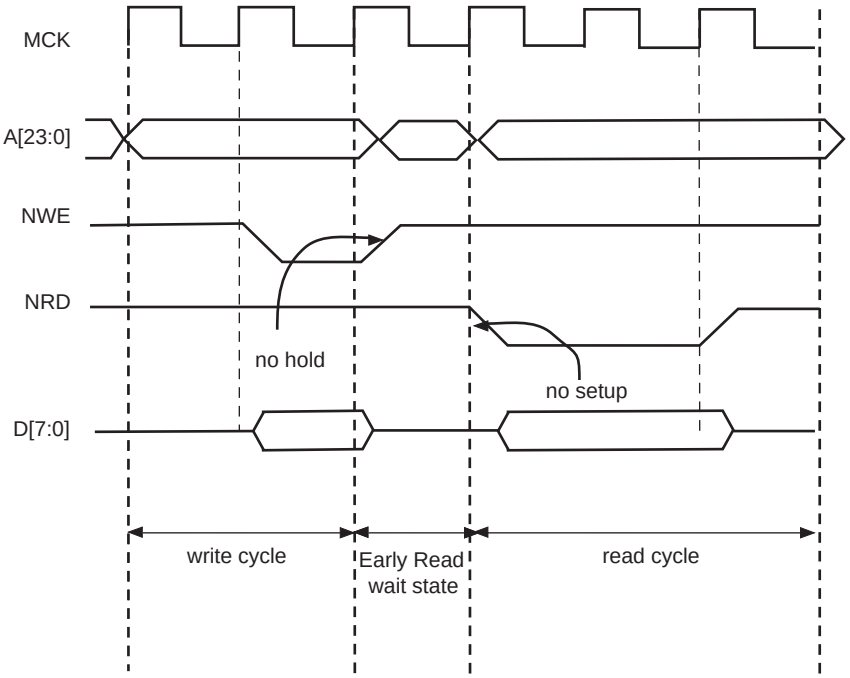


Figure 27-15. Early Read Wait State: NCS-controlled write with no hold followed by a read with no NCS setup

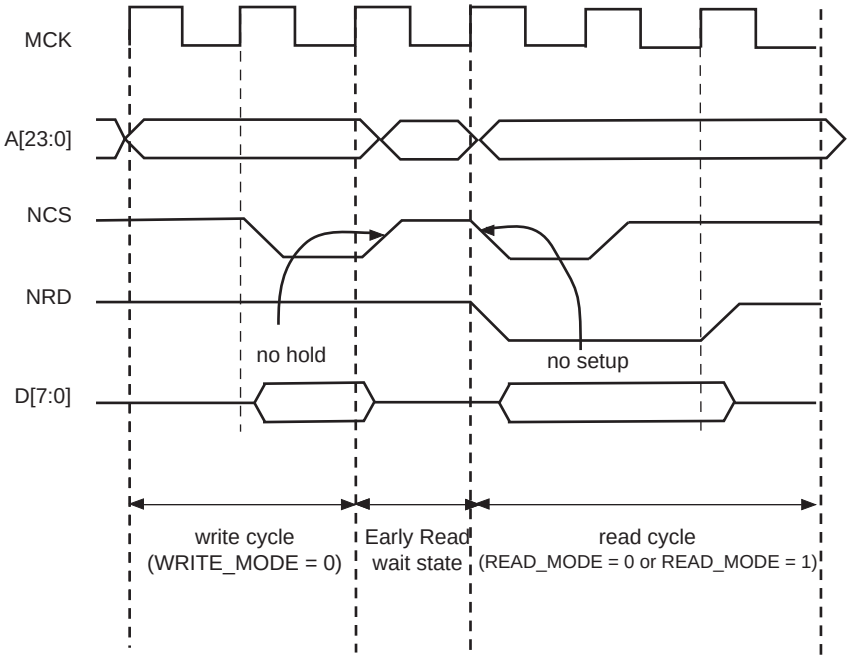
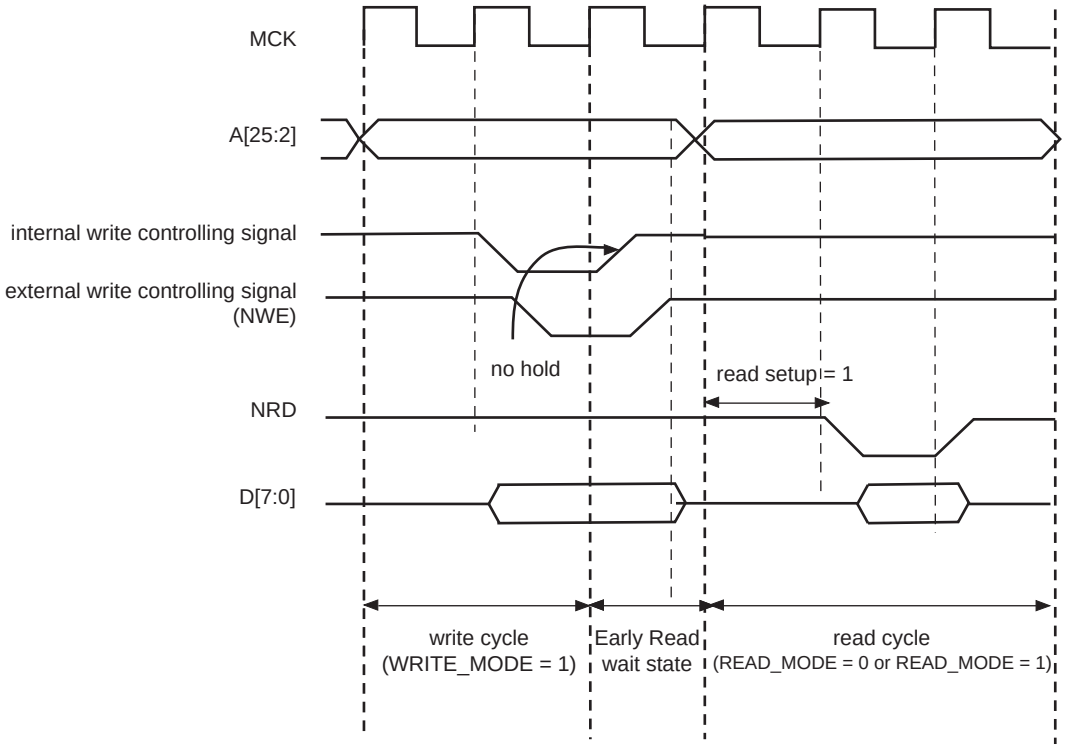


Figure 27-16. Early Read Wait State: NWE-controlled write with no hold followed by a read with one set-up cycle



27.11.3 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. This “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.

27.11.3.1 User Procedure

To insert a reload configuration wait state, the SMC detects a write access to any SMC_MODE register of the user interface. If the user only modifies timing registers (SMC_SETUP, SMC_PULSE, SMC_CYCLE registers) in the user interface, he must validate the modification by writing the SMC_MODE, even if no change was made on the mode parameters.

The user must not change the configuration parameters of an SMC chip select (Setup, Pulse, Cycle, Mode) if accesses are performed on this CS during the modification. Any change of the chip select parameters, while fetching the code from a memory connected on this CS, may lead to unpredictable behavior. The instructions used to modify the parameters of an SMC chip select can be executed from the internal RAM or from a memory connected to another CS.

27.11.3.2 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 Section 27.14 ”Slow Clock Mode”).

27.11.4 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 27-13.

27.12 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 SMC_MODE.TDF_CYCLES field for the corresponding chip select. The value of SMC_MODE.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 SMC_MODE.READ_MODE and the SMC_MODE.TDF_MODE fields for the corresponding chip select.

27.12.1 SMC_MODE.READ_MODE

Setting SMC_MODE.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 SMC_MODE.TDF_CYCLES MCK cycles.

When the read operation is controlled by the NCS signal (SMC_MODE.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 27-17 illustrates the Data Float Period in NRD-controlled mode (SMC_MODE.READ_MODE =1), assuming a data float period of 2 cycles (SMC_MODE.TDF_CYCLES = 2). Figure 27-18 shows the read operation when controlled by NCS (SMC_MODE.READ_MODE = 0) and SMC_MODE.TDF_CYCLES = 3.

Figure 27-17. TDF Period in NRD Controlled Read Access (TDF = 2)

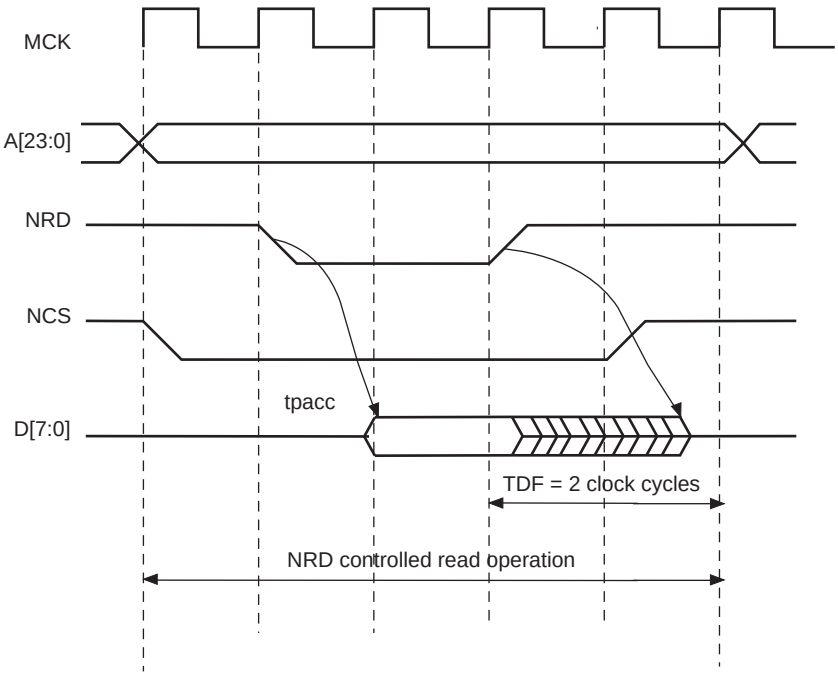
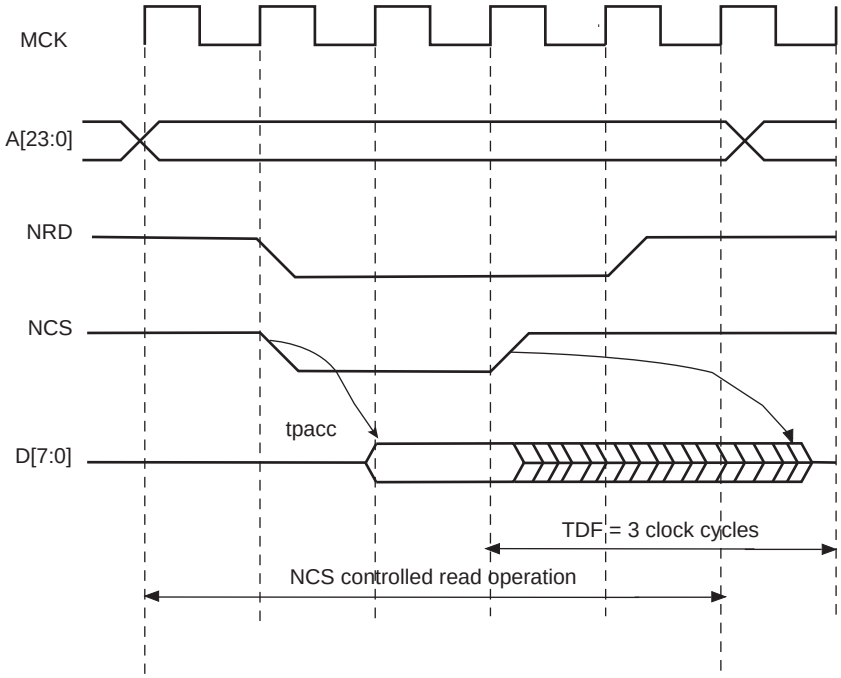


Figure 27-18. TDF Period in NCS Controlled Read Operation (TDF = 3)



27.12.2 TDF Optimization Enabled (SMC_MODE.TDF_MODE = 1)

When SMC_MODE.TDF_MODE 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.

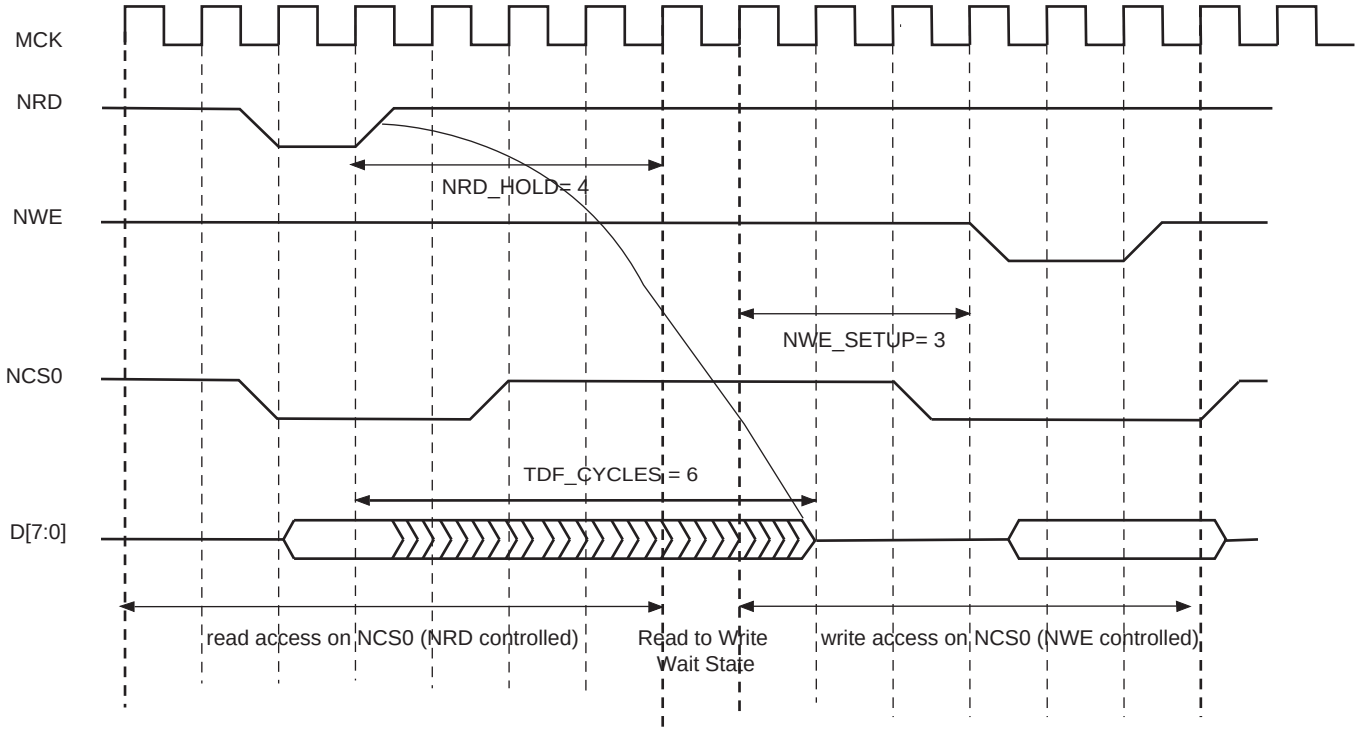
Figure 27-19 shows a read access controlled by NRD, followed by a write access controlled by NWE, on chip select 0. Chip select 0 has been programmed with:

NRD_HOLD = 4; SMC_MODE.READ_MODE = 1 (NRD controlled)

NWE_SETUP = 3; SMC_MODE.WRITE_MODE = 1 (NWE controlled)

SMC_MODE.TDF_CYCLES = 6; SMC_MODE.TDF_MODE = 1 (optimization enabled).

Figure 27-19. TDF Optimization: No TDF wait states are inserted if the TDF period is over when the next access begins



27.12.3 TDF Optimization Disabled (SMC_MODE.TDF_MODE = 0)

When optimization is disabled, TDF wait states are inserted at the end of the read transfer, so that the data float period is ended when the second access begins. If the hold period of the read1 controlling signal overlaps the data float period, no additional tdf wait states will be inserted.

Figure 27-20, Figure 27-21 and Figure 27-22 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 27-20. TDF Optimization Disabled (TDF Mode = 0): TDF wait states between 2 read accesses on different chip selects

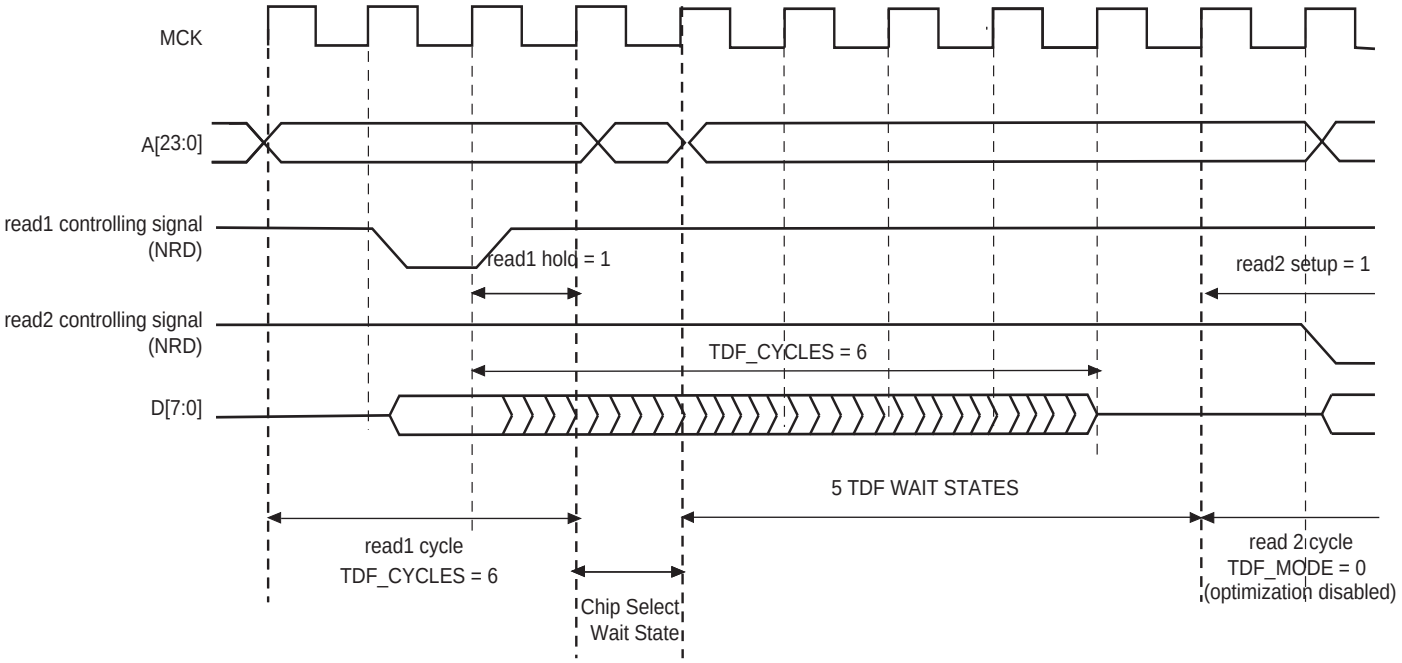


Figure 27-21. TDF Mode = 0: TDF wait states between a read and a write access on different chip selects

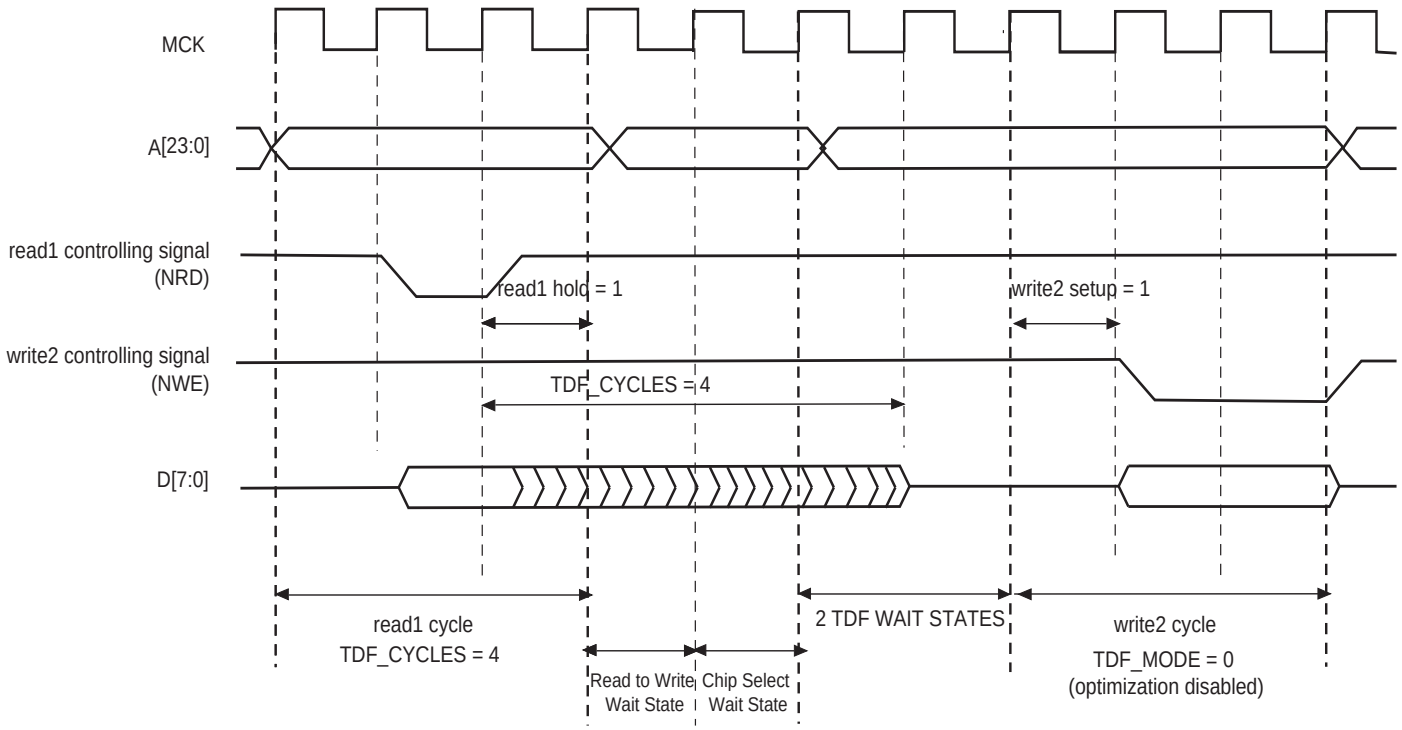
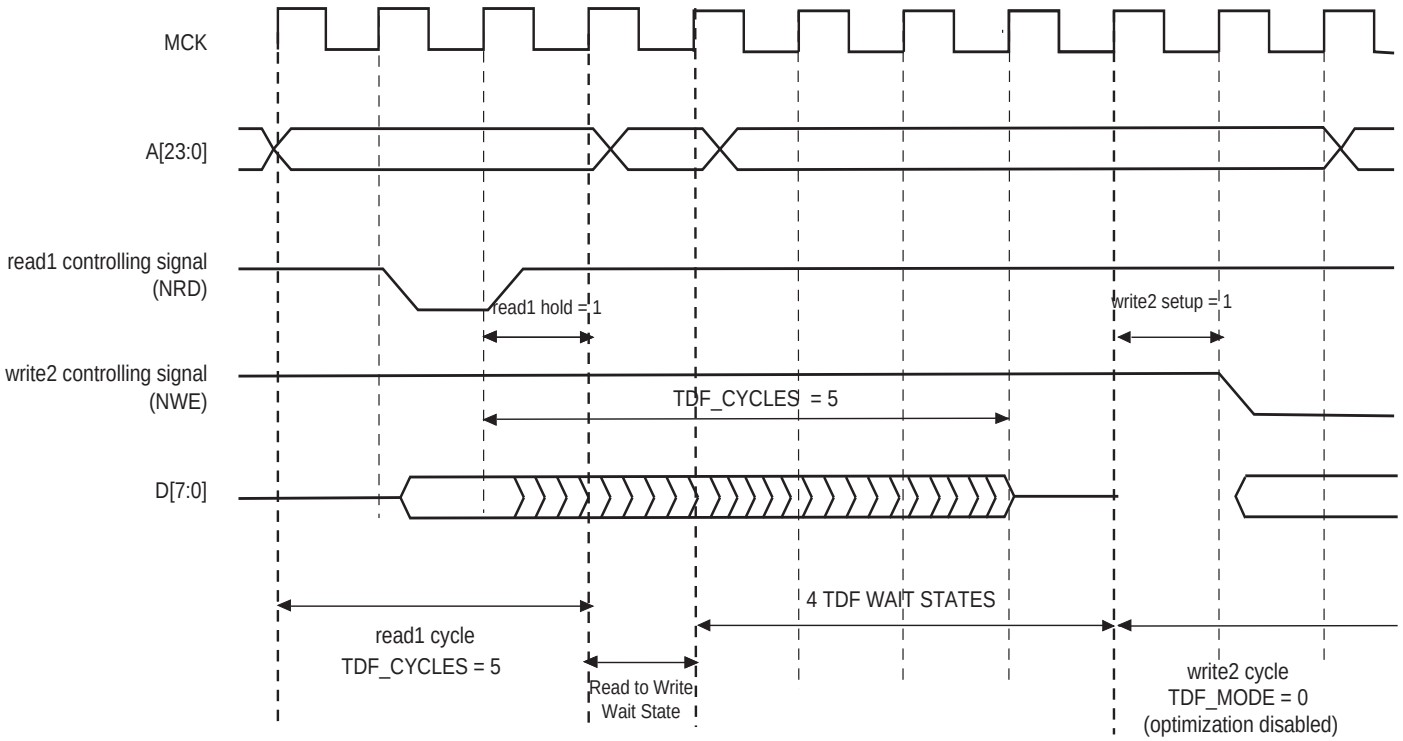


Figure 27-22. TDF Mode = 0: TDF wait states between read and write accesses on the same chip select



27.13 External Wait

Any access can be extended by an external device using the NWAIT input signal of the SMC. The SMC_MODE.EXNW_MODE field on the corresponding chip select must be set either to “10” (Frozen mode) or “11” (Ready mode). When SMC_MODE.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.

27.13.1 Restriction

When SMC_MODE.EXNW_MODE is enabled, **it is mandatory to program at least one hold cycle for the read/write controlling signal. For that reason, the NWAIT signal cannot be used in Page mode (Section 27.15 ”Asynchronous Page Mode”), or in Slow clock mode (Section 27.14 ”Slow Clock Mode”).**

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.

27.13.2 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 27-23. 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 27-24.

Figure 27-23. Write Access with NWAIT Assertion in Frozen Mode (SMC_MODE.EXNW_MODE = 10)

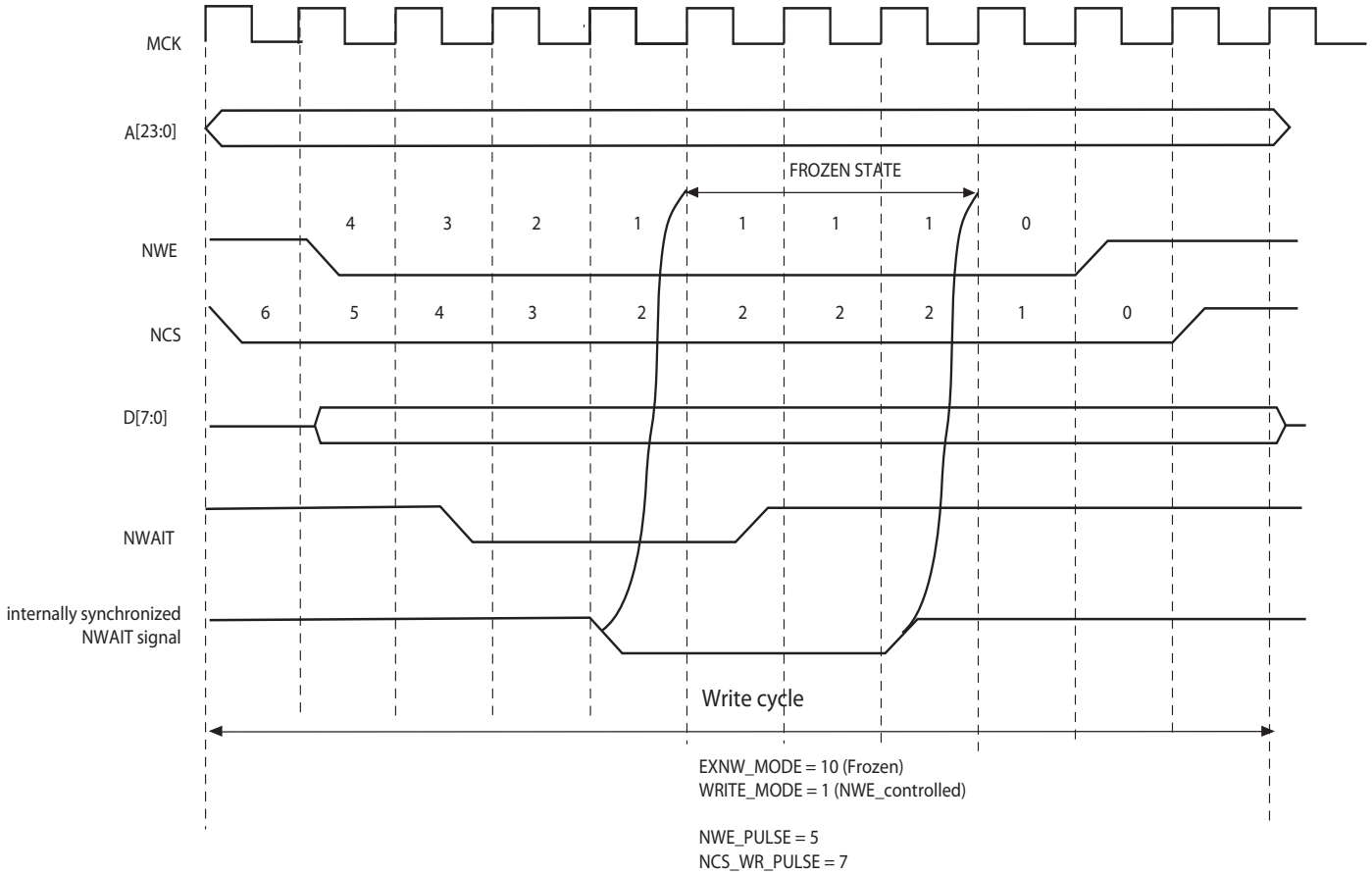
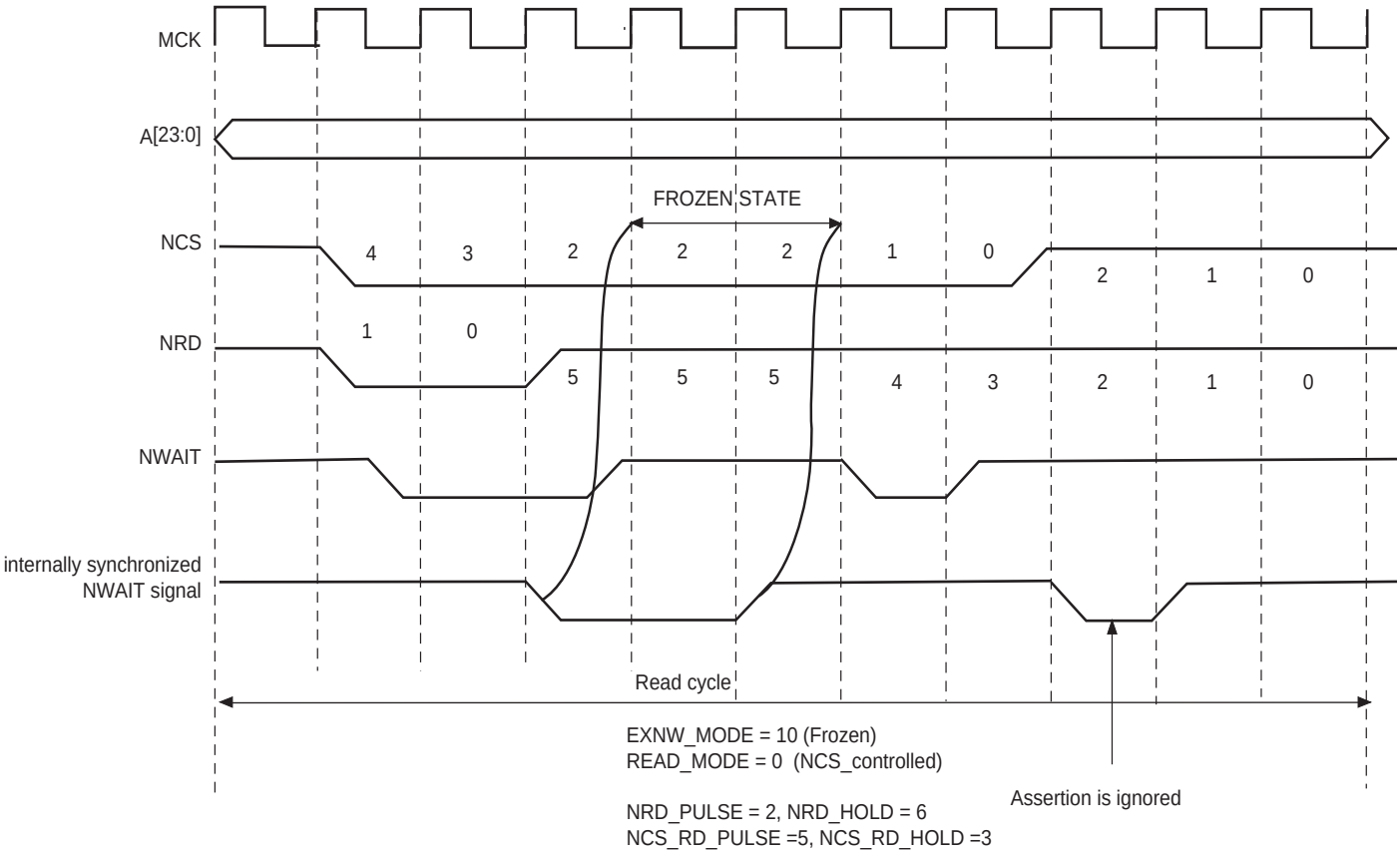


Figure 27-24. Read Access with NWAIT Assertion in Frozen Mode (SMC_MODE.EXNW_MODE = 10)



27.13.3 Ready Mode

In Ready mode (SMC_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 27-25 and Figure 27-26. 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 27-26.

Figure 27-25. NWAIT Assertion in Write Access: Ready Mode (SMC_MODE.EXNW_MODE = 11)

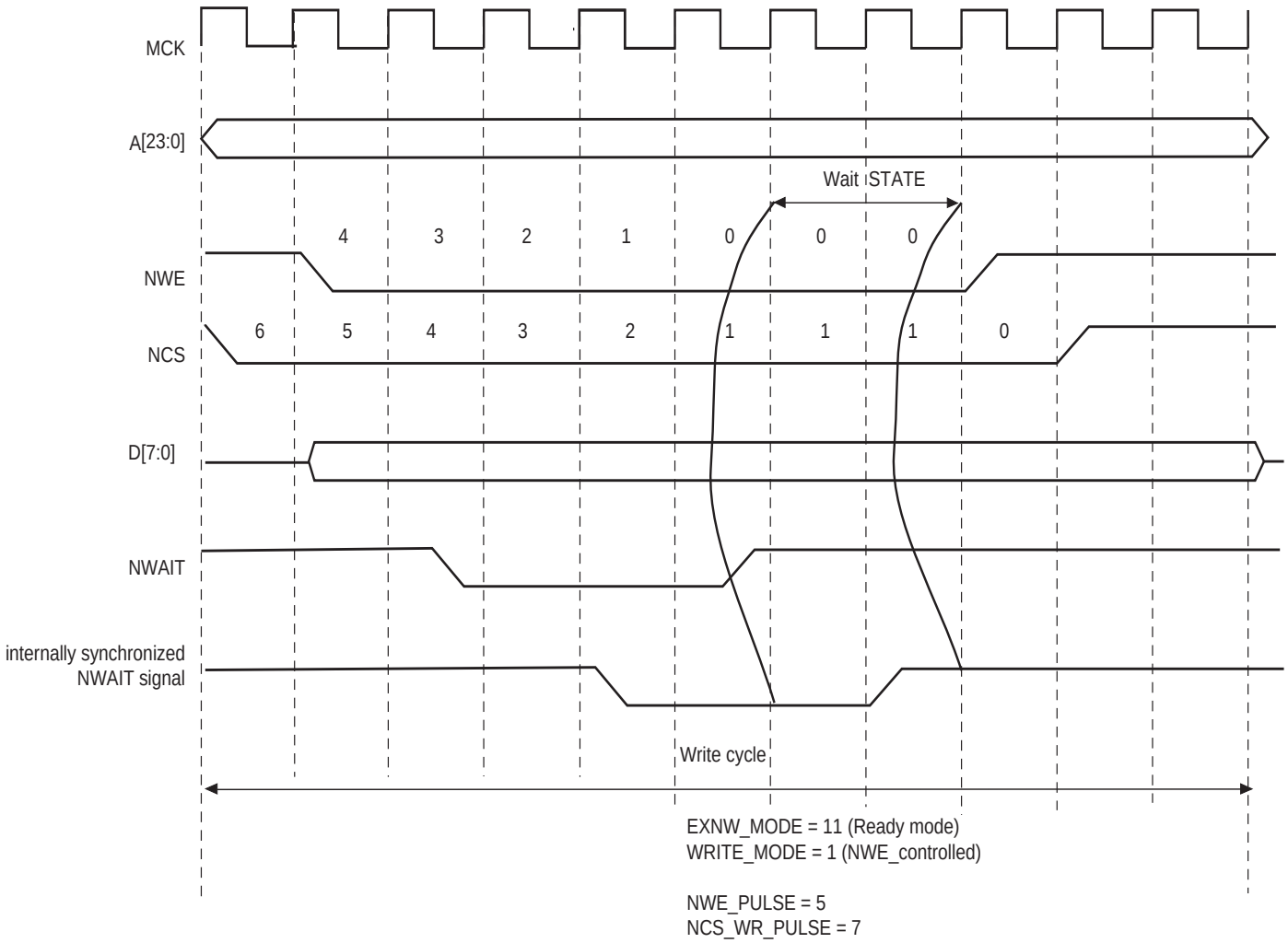
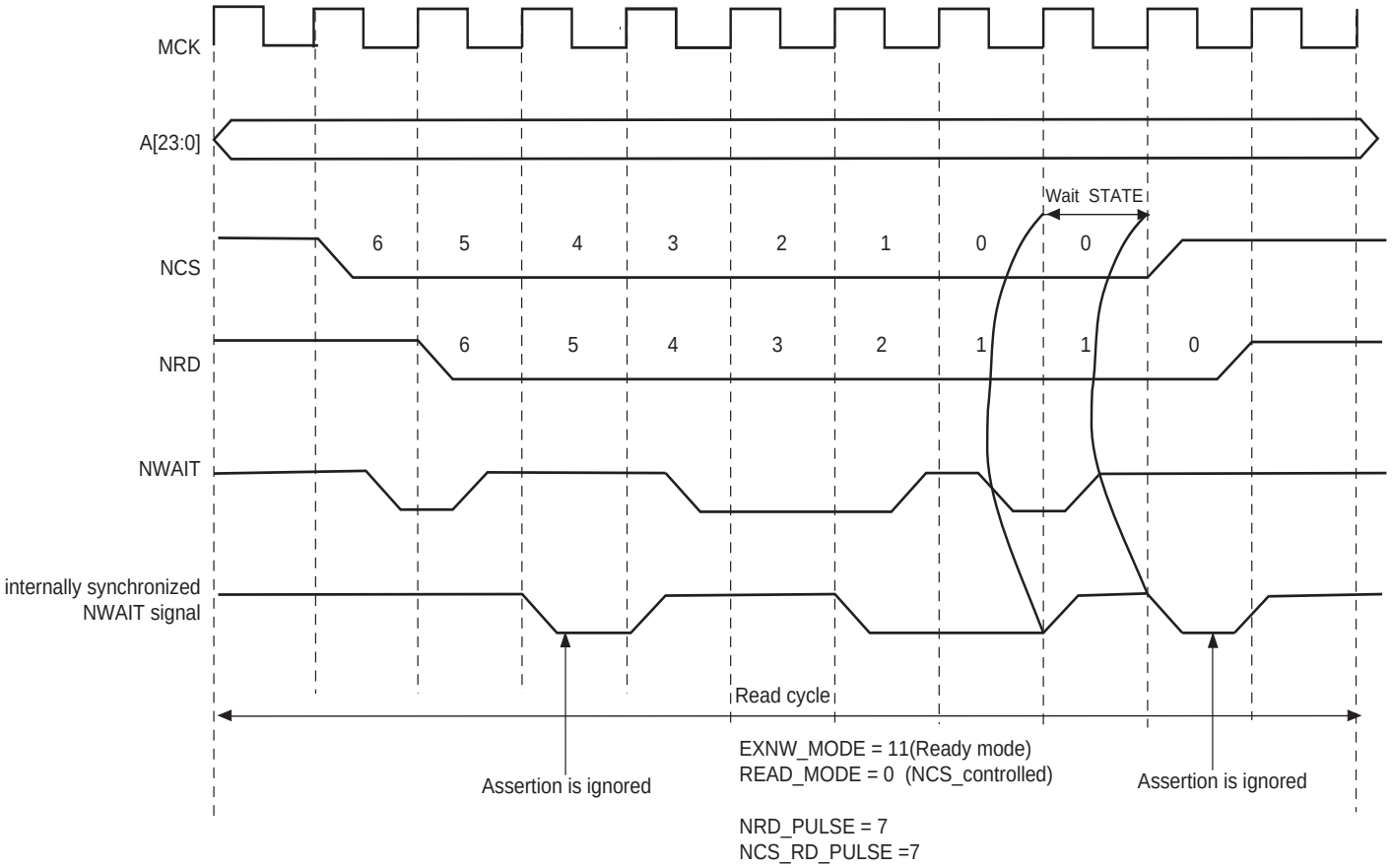


Figure 27-26. NWAIT Assertion in Read Access: Ready Mode (SMC_MODE.EXNW_MODE = 11)



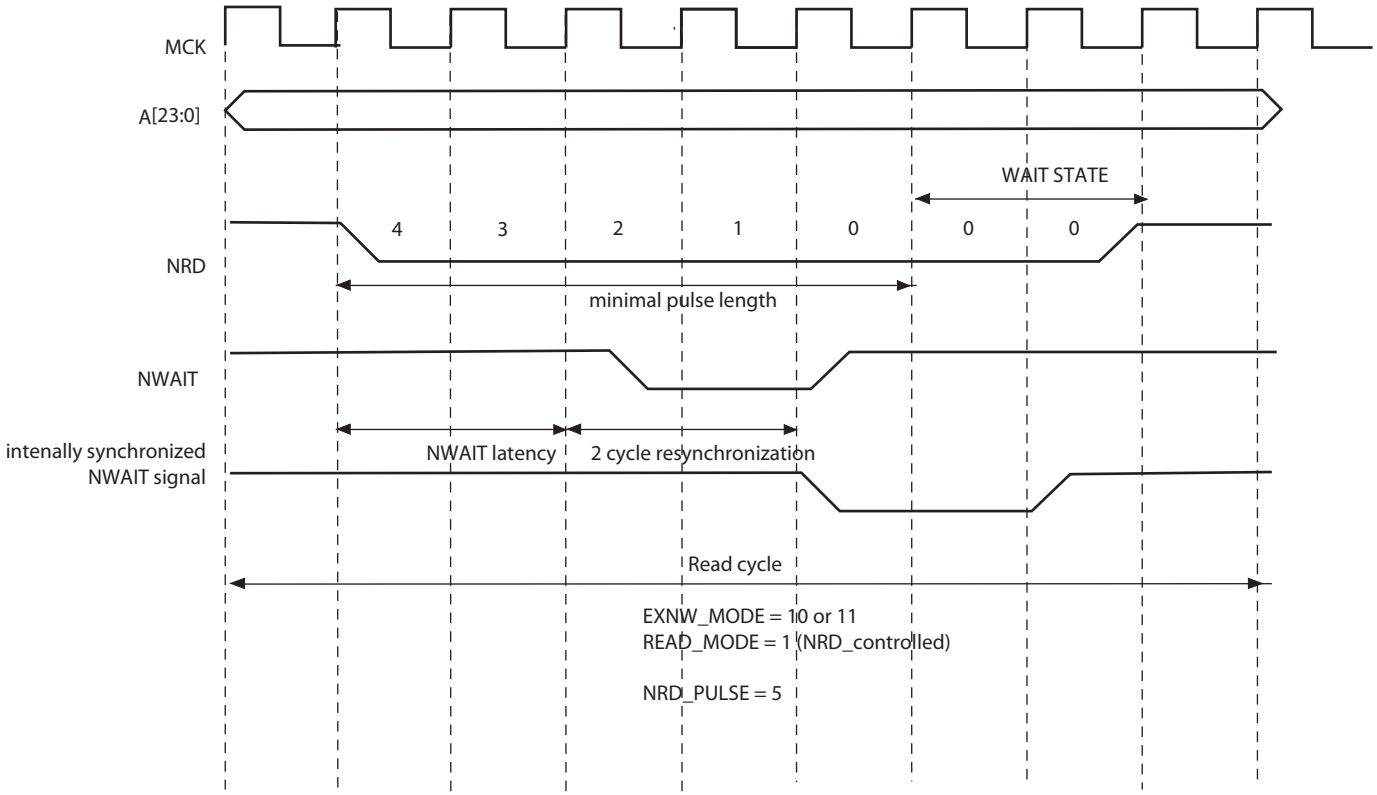
27.13.4 NWAIT Latency and Read/Write Timings

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 + one 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 27-27.

When SMC_MODE.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 27-27. NWAIT Latency



27.14 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 32kHz 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 a very slow clock rate. When activated, the Slow clock mode is active on all chip selects.

27.14.1 Slow Clock Mode Waveforms

Figure 27-28 illustrates the read and write operations in Slow clock mode. They are valid on all chip selects. Table 27-6 indicates the value of read and write parameters in Slow clock mode.

Figure 27-28. Read/Write Cycles in Slow Clock Mode

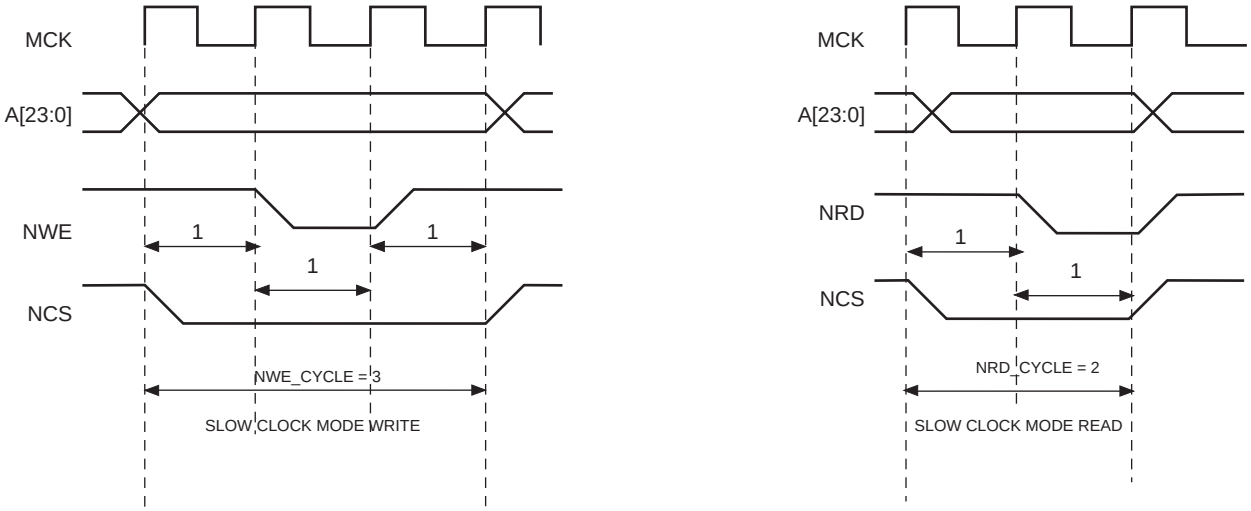


Table 27-6. Read and Write Timing Parameters in Slow Clock Mode

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

27.14.2 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 a high clock rate, with the set of Slow clock mode parameters.See Figure 27-29. The external device may not be fast enough to support such timings.

Figure 27-30 illustrates the recommended procedure to switch from one mode to the other.

Figure 27-29. Clock Rate Transition Occurs while the SMC is Performing a Write Operation

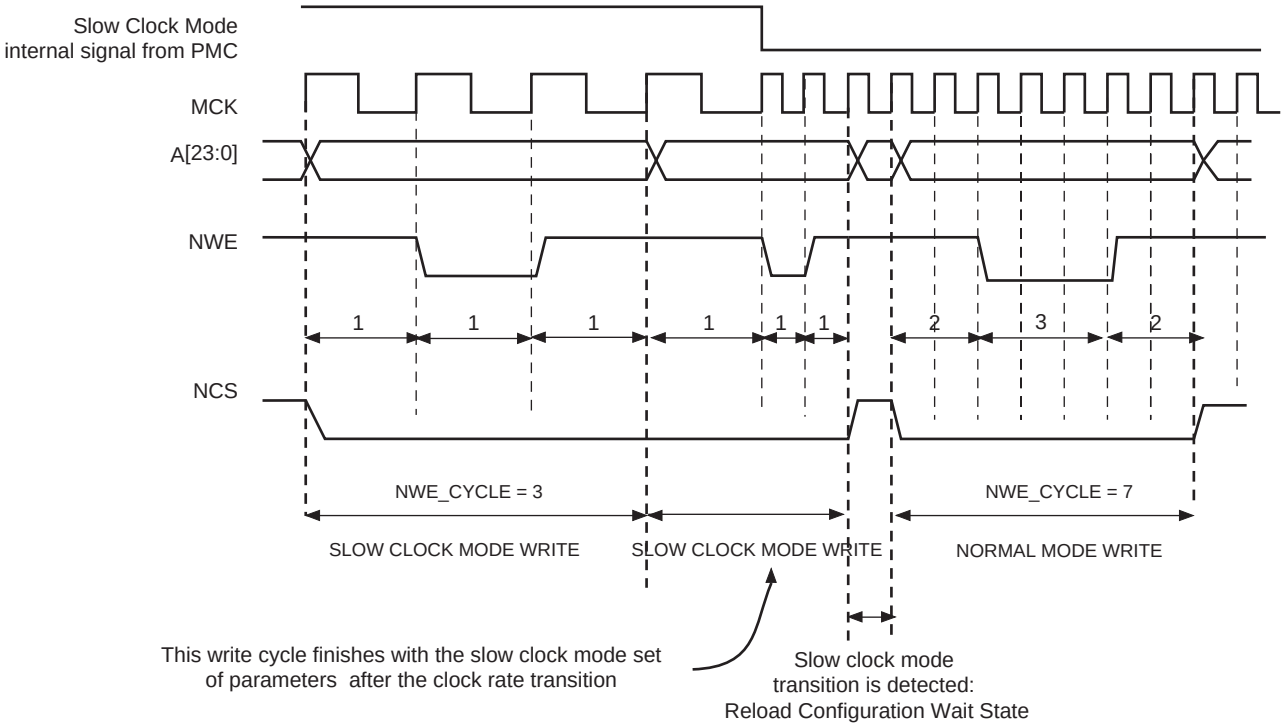
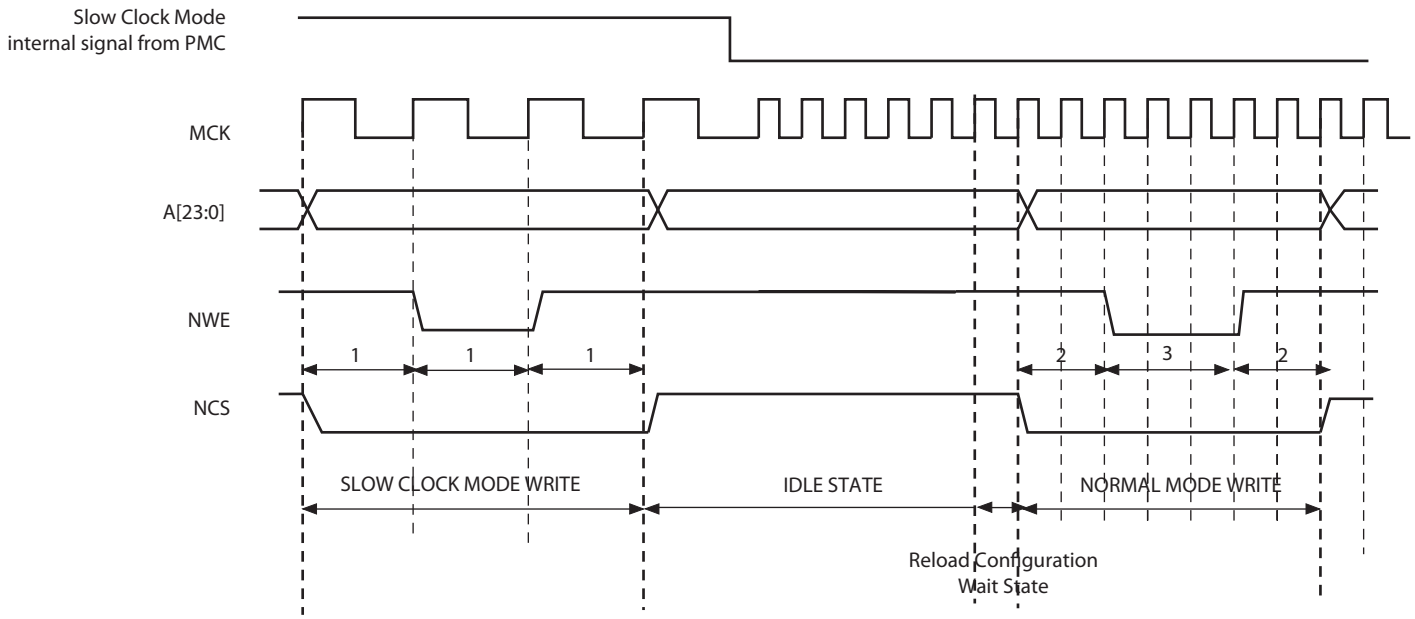


Figure 27-30. Recommended Procedure to Switch from Slow Clock Mode to Normal Mode or from Normal Mode to Slow Clock Mode



27.15 Asynchronous Page Mode

The SMC supports asynchronous burst reads in Page mode, provided that the Page mode is enabled (SMC_MODE.PMEN =1). The page size must be configured in the SMC_MODE register (PS field) to 4, 8, 16 or 32 bytes.

The page defines a set of consecutive bytes into memory. A 4-byte page (resp. 8-, 16-, 32-byte page) is always aligned to 4-byte boundaries (resp. 8-, 16-, 32-byte boundaries) of memory. The MSB of data address defines the address of the page in memory, the LSB of address define the address of the data in the page as detailed in Table 27-7.

With Page mode memory devices, the first access to one page (t_{pa}) takes longer than the subsequent accesses to the page (t_{sa}) as shown in Figure 27-31. When in Page mode, the SMC enables the user to define different read timings for the first access within one page, and next accesses within the page.

Table 27-7. Page Address and Data Address within a Page

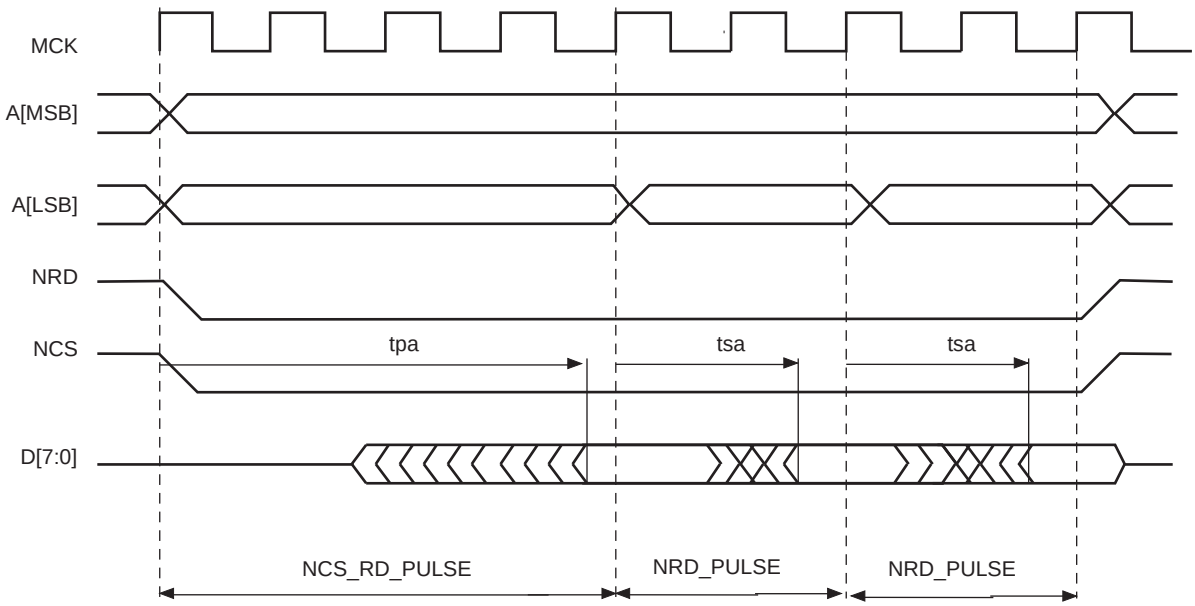
Page Size	Page Address(1)	Data Address in the Page
4 bytes	A[23:2]	A[1:0]
8 bytes	A[23:3]	A[2:0]
16 bytes	A[23:4]	A[3:0]
32 bytes	A[23:5]	A[4:0]

Note: 1. “A” denotes the address bus of the memory device.

27.15.1 Protocol and Timings in Page Mode

Figure 27-31 shows the NRD and NCS timings in Page mode access.

Figure 27-31. Page Mode Read Protocol (Address MSB and LSB are defined in Table 27-7)



The NRD and NCS signals are held low during all read transfers, whatever the programmed values of the setup and hold timings in the User Interface may be. Moreover, the NRD and NCS timings are identical. The pulse length of the first access to the page is defined with the NCS_RD_PULSE field of the SMC_PULSE register. The pulse length of subsequent accesses within the page are defined using the NRD_PULSE parameter.

In Page mode, the programming of the read timings is described in Table 27-8:

Table 27-8. Programming of Read Timings in Page Mode

Parameter	Value	Definition
READ_MODE	'x'	No impact
NCS_RD_SETUP	'x'	No impact
NCS_RD_PULSE	t_{pa}	Access time of first access to the page
NRD_SETUP	'x'	No impact
NRD_PULSE	t_{sa}	Access time of subsequent accesses in the page
NRD_CYCLE	'x'	No impact

The SMC does not check the coherency of timings. It will always apply the NCS_RD_PULSE timings as page access timing (t_{pa}) and the NRD_PULSE for accesses to the page (t_{sa}), even if the programmed value for t_{pa} is shorter than the programmed value for t_{sa}.

27.15.2 Page Mode Restriction

The Page mode is not compatible with the use of the NWAIT signal. Using the Page mode and the NWAIT signal may lead to unpredictable behavior.

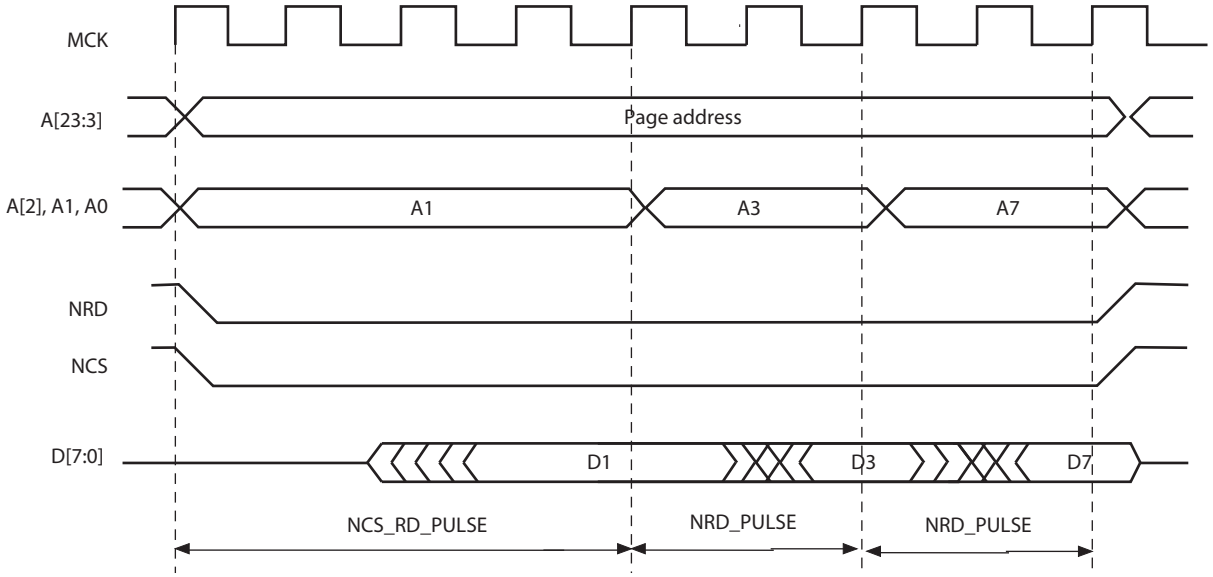
27.15.3 Sequential and Non-sequential Accesses

If the chip select and the MSB of addresses as defined in Table 27-7 are identical, then the current access lies in the same page as the previous one, and no page break occurs.

Using this information, all data within the same page, sequential or not sequential, are accessed with a minimum access time (t_{sa}). Figure 27-32 illustrates access to an 8-bit memory device in Page mode, with 8-byte pages. Access to D1 causes a page access with a long access time (t_{pa}). Accesses to D3 and D7, though they are not sequential accesses, only require a short access time (t_{sa}).

If the MSB of addresses are different, the SMC performs the access of a new page. In the same way, if the chip select is different from the previous access, a page break occurs. If two sequential accesses are made to the Page mode memory, but separated by an other internal or external peripheral access, a page break occurs on the second access because the chip select of the device was deasserted between both accesses.

Figure 27-32. Access to Non-Sequential Data within the Same Page



27.16 Static Memory Controller (SMC) User Interface

The SMC is programmed using the registers listed in Table 27-9. For each chip select, a set of four registers is used to program the parameters of the external device connected on it. In Table 27-9, “CS_number” denotes the chip select number. 16 bytes (0x10) are required per chip select.

Note: The user must confirm the SMC configuration by writing any one of the SMC_MODE registers.

Table 27-9. Register Mapping

Offset	Register	Name	Access	Reset
0x10 x CS_number + 0x00	SMC Setup Register	SMC_SETUP	Read/Write	0x01010101
0x10 x CS_number + 0x04	SMC Pulse Register	SMC_PULSE	Read/write	0x01010101
0x10 x CS_number + 0x08	SMC Cycle Register	SMC_CYCLE	Read/Write	0x00030003
0x10 x CS_number + 0x0C	SMC Mode Register	SMC_MODE	Read/Write	0x10000003
0x80	SMC Off-Chip Memory Scrambling Register	SMC_OCMS	Read/Write	0x00000000
0x84	SMC Off-Chip Memory Scrambling KEY1 Register	SMC_KEY1	Write-once	0x00000000
0x88	SMC Off-Chip Memory Scrambling KEY2 Register	SMC_KEY2	Write-once	0x00000000
0xE4	SMC Write Protection Mode Register	SMC_WPMR	Read/Write	0x00000000
0xE8	SMC Write Protection Status Register	SMC_WPSR	Read-only	0x00000000
0xEC-0xFC	Reserved	–	–	–

Notes: 1. All unlisted offset values are considered as ‘reserved’.

27.16.1 SMC Setup Register

Name: SMC_SETUP[0..3]

Address: 0x40060000 [0], 0x40060010 [1], 0x40060020 [2], 0x40060030 [3]

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	NCS_RD_SETUP					

23	22	21	20	19	18	17	16
–	–	NRD_SETUP					

15	14	13	12	11	10	9	8
–	–	NCS_WR_SETUP					

7	6	5	4	3	2	1	0
–	–	NWE_SETUP					

This register can only be written if the WPEN bit is cleared in the “SMC Write Protection Mode Register” .

- **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 in clock cycles 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

27.16.2 SMC Pulse Register

Name: SMC_PULSE[0..3]

Address: 0x40060004 [0], 0x40060014 [1], 0x40060024 [2], 0x40060034 [3]

Access: Read/Write

31	30	29	28	27	26	25	24
–	NCS_RD_PULSE						

23	22	21	20	19	18	17	16
–	NRD_PULSE						

15	14	13	12	11	10	9	8
–	NCS_WR_PULSE						

7	6	5	4	3	2	1	0
–	NWE_PULSE						

This register can only be written if the WPEN bit is cleared in the “SMC Write Protection Mode Register” .

- **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 length must be at least 1 clock cycle.

- **NCS_WR_PULSE: NCS Pulse Length in WRITE Access**

In write access, the NCS signal pulse length is defined as:

NCS pulse length = (256* NCS_WR_PULSE[6] + NCS_WR_PULSE[5:0]) clock cycles

The NCS pulse length must be at least 1 clock cycle.

- **NRD_PULSE: NRD Pulse Length**

In standard read access, the NRD signal pulse length is defined in clock cycles as:

NRD pulse length = (256* NRD_PULSE[6] + NRD_PULSE[5:0]) clock cycles

The NRD pulse length must be at least 1 clock cycle.

In Page mode read access, the NRD_PULSE parameter defines the duration of the subsequent accesses in the page.

- **NCS_RD_PULSE: NCS Pulse Length in READ Access**

In standard read access, the NCS signal pulse length is defined as:

NCS pulse length = (256* NCS_RD_PULSE[6] + NCS_RD_PULSE[5:0]) clock cycles

The NCS pulse length must be at least 1 clock cycle.

In Page mode read access, the NCS_RD_PULSE parameter defines the duration of the first access to one page.

27.16.3 SMC Cycle Register

Name: SMC_CYCLE[0..3]

Address: 0x40060008 [0], 0x40060018 [1], 0x40060028 [2], 0x40060038 [3]

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	NRD_CYCLE

23	22	21	20	19	18	17	16
NRD_CYCLE							

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	NWE_CYCLE

7	6	5	4	3	2	1	0
NWE_CYCLE							

This register can only be written if the WPEN bit is cleared in the “SMC Write Protection Mode Register” .

- **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

27.16.4 SMC Mode Register

Name: SMC_MODE[0..3]

Address: 0x4006000C [0], 0x4006001C [1], 0x4006002C [2], 0x4006003C [3]

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	PS		–	–	–	PMEN
23	22	21	20	19	18	17	16
–	–	–	TDF_MODE	TDF_CYCLES			
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	EXNW_MODE		–	–	WRITE_MODE	READ_MODE

This register can only be written if the WPEN bit is cleared in the “SMC Write Protection Mode Register” .

The user must confirm the SMC configuration by writing any one of the SMC_MODE registers.

- **READ_MODE: Read Mode**

0: The read operation is controlled by the NCS signal.

– If TDF cycles are programmed, the external bus is marked busy after the rising edge of NCS.

– If TDF optimization is enabled (TDF_MODE =1), TDF wait states are inserted after the setup of NCS.

1: The read operation is controlled by the NRD signal.

– If TDF cycles are programmed, the external bus is marked busy after the rising edge of NRD.

– If TDF optimization is enabled (TDF_MODE =1), TDF wait states are inserted after the setup of NRD.

- **WRITE_MODE: Write Mode**

0: The write operation is controlled by the NCS signal.

– If TDF optimization is enabled (TDF_MODE =1), TDF wait states will be inserted after the setup of NCS.

1: The write operation is controlled by the NWE signal.

– If TDF optimization is enabled (TDF_MODE =1), TDF wait states will be inserted after the setup of NWE.

- **EXNW_MODE: NWAIT Mode**

The NWAIT signal is used to extend the current read or write signal. It is only taken into account during the pulse phase of the read and write controlling signal. When the use of NWAIT is enabled, at least one cycle hold duration must be programmed for the read and write controlling signal.

Value	Name	Description
0	DISABLED	Disabled–The NWAIT input signal is ignored on the corresponding chip select.
1	–	Reserved
2	FROZEN	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.
3	READY	Ready Mode–The NWAIT signal indicates the availability of the external device at the end of the pulse of the controlling read or write signal, to complete the access. If high, the access normally completes. If low, the access is extended until NWAIT returns high.

- **TDF_CYCLES: Data Float Time**

This field gives the integer number of clock cycles required by the external device to release the data after the rising edge of the read controlling signal. The SMC always provide one full cycle of bus turnaround after the TDF_CYCLES period. The external bus cannot be used by another chip select during TDF_CYCLES + 1 cycles. From 0 up to 15 TDF_CYCLES can be set.

- **TDF_MODE: TDF Optimization**

0: TDF optimization disabled–the number of TDF wait states is inserted before the next access begins.

1: TDF optimization enabled–the number of TDF wait states is optimized using the setup period of the next read/write access.

- **PMEN: Page Mode Enabled**

0: Standard read is applied.

1: Asynchronous burst read in page mode is applied on the corresponding chip select.

- **PS: Page Size**

If page mode is enabled, this field indicates the size of the page in bytes.

Value	Name	Description
0	4_BYTE	4-byte page
1	8_BYTE	8-byte page
2	16_BYTE	16-byte page
3	32_BYTE	32-byte page

27.16.5 SMC Off-Chip Memory Scrambling Register

Name: SMC_OCMS

Address: 0x40060080

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
–	–	–	–	CS3SE	CS2SE	CS1SE	CS0SE
7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	SMSE

- **CSxSE: Chip Select (x = 0 to 3) Scrambling Enable**

0: Disable scrambling for CSx.

1: Enable scrambling for CSx.

- **SMSE: Static Memory Controller Scrambling Enable**

0: Disable scrambling for SMC access.

1: Enable scrambling for SMC access.

27.16.6 SMC Off-Chip Memory Scrambling Key1 Register

Name: SMC_KEY1

Address: 0x40060084

Access: Write-once[1]

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							

Note: 1. 'Write-once' access indicates that the first write access after a system reset prevents any further modification of the value of this register.

2.

- **KEY1: Off-Chip Memory Scrambling (OCMS) Key Part 1**

When off-chip memory scrambling is enabled, KEY1 and KEY2 values determine data scrambling.

27.16.7 SMC Off-Chip Memory Scrambling Key2 Register

Name: SMC_KEY2

Address: 0x40060088

Access: Write-once[1]

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							

Notes:
1. 'Write-once' access indicates that the first write access after a system reset prevents any further modification of the value of this register.
2.

- **KEY2: Off-Chip Memory Scrambling (OCMS) Key Part 2**

When off-chip memory scrambling is enabled, KEY1 and KEY2 values determine data scrambling.

27.16.8 SMC Write Protection Mode Register

Name: SMC_WPMR

Address: 0x400600E4

Access: Read/Write

31	30	29	28	27	26	25	24
WPKEY							

23	22	21	20	19	18	17	16
WPKEY							

15	14	13	12	11	10	9	8
WPKEY							

7	6	5	4	3	2	1	0
—	—	—	—	—	—	—	WPEN

- **WPEN: Write Protect Enable**

0: Disables the write protection if WPKEY corresponds to 0x534D43 (“SMC” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x534D43 (“SMC” in ASCII).

See Section 27.9.5 ”Register Write Protection” for the list of registers that can be write-protected.

- **WPKEY: Write Protection Key**

Value	Name	Description
0x534D43	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

27.16.9 SMC Write Protection Status Register

Name: SMC_WPSR

Address: 0x400600E8

Type: Read-only

31	30	29	28	27	26	25	24
—	—	—	—	—	—	—	—

23	22	21	20	19	18	17	16
WPVSRC							

15	14	13	12	11	10	9	8
WPVSRC							

7	6	5	4	3	2	1	0
—	—	—	—	—	—	—	WPVS

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of the SMC_WPSR register.

1: A write protection 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 WPVSRC.

- **WPVSRC: Write Protection Violation Source**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

28. Clock Generator

28.1 Description

The Clock Generator user interface is embedded within the Power Management Controller and is described in Section 29.18 ”Power Management Controller (PMC) User Interface”. However, the Clock Generator registers are named CKGR_.

28.2 Embedded Characteristics

The Clock Generator is made up of:

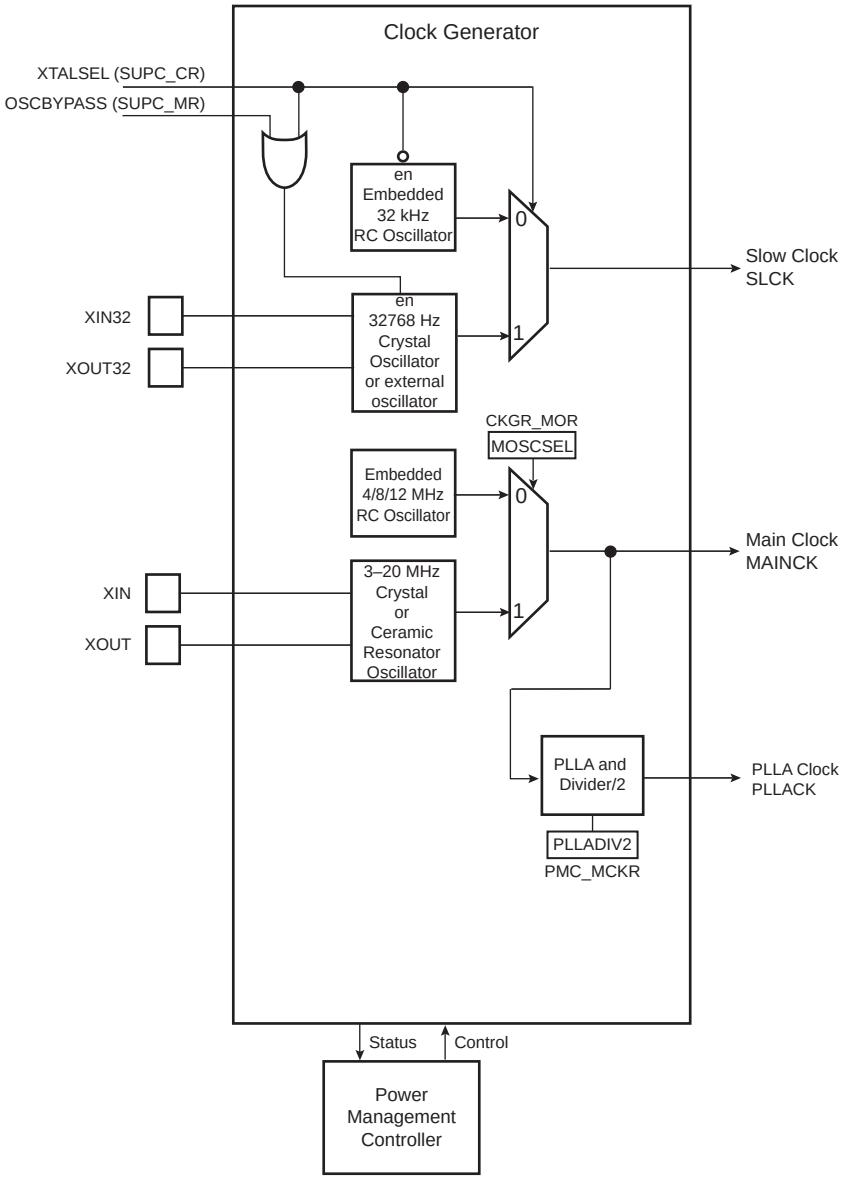
- A low-power 32768 Hz crystal oscillator with Bypass mode
- A low-power embedded 32 kHz (typical) RC oscillator
- A 3 to 20 MHz crystal or ceramic resonator-based oscillator, which can be bypassed.
- A factory-trimmed embedded RC oscillator. Three output frequencies can be selected: 4/8/12 MHz. By default 4 MHz is selected.
- A 80 to 240 MHz programmable PLL (input from 3 to 32 MHz), capable of providing the clock MCK to the processor and to the peripherals.

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 RC oscillator.
- PLLACK is the output of the divider and 80 to 240 MHz programmable PLL (PLLA).

28.3 Block Diagram

Figure 28-1. Clock Generator Block Diagram



28.4 Slow Clock

The Supply Controller embeds a slow clock generator that is supplied with the VDDIO power supply. As soon as VDDIO is supplied, both the 32768 Hz crystal oscillator and the embedded 32 kHz (typical) RC oscillator are powered up, but only the 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 32768 Hz crystal oscillator or by the embedded 32 kHz (typical) RC oscillator.

The selection of the slow clock source is made via the XTALSEL bit in the Supply Controller Control Register (SUPC_CR).

The OSCSEL bit of the Supply Controller Status Register (SUPC_SR) and the OSCSELS bit of the PMC Status Register (PMC_SR) report which oscillator is selected as the slow clock source. PMC_SR.OSCSELS informs when the switch sequence initiated by a new value written in SUPC_CR.XTALSEL is done.

28.4.1 Embedded 32 kHz (typical) RC Oscillator

By default, the embedded 32 kHz (typical) RC oscillator is enabled and selected. The user has to take into account the possible drifts of this oscillator. More details are given in the section “DC Characteristics”.

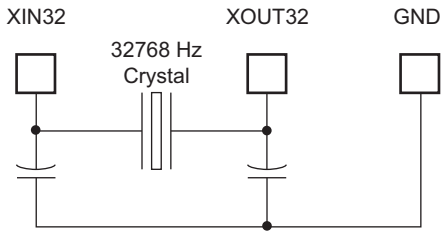
This oscillator is disabled by clearing the SUPC_CR.XTALSEL.

28.4.2 32768 Hz Crystal Oscillator

The Clock Generator integrates a low-power 32768 Hz crystal oscillator. To use this oscillator, the XIN32 and XOUT32 pins must be connected to a 32768 Hz crystal. Two external capacitors must be wired as shown in Figure 28-2. More details are given in the section “DC Characteristics”.

Note that the user is not obliged to use the 32768 Hz crystal oscillator and can use the 32 kHz (typical) RC oscillator instead.

Figure 28-2. Typical 32768 Hz Crystal Oscillator Connection



The 32768 Hz crystal oscillator provides a more accurate frequency than the 32 kHz (typical) RC oscillator.

To select the 32768 Hz crystal oscillator as the source of the slow clock, the bit SUPC_CR.XTALSEL must be set. This results in a sequence which first configures the PIO lines multiplexed with XIN32 and XOUT32 to be driven by the slow clock oscillator, then enables the 32768 Hz crystal oscillator and then disables the 32 kHz (typical) RC oscillator to save power. The switch of the slow clock source is glitch-free.

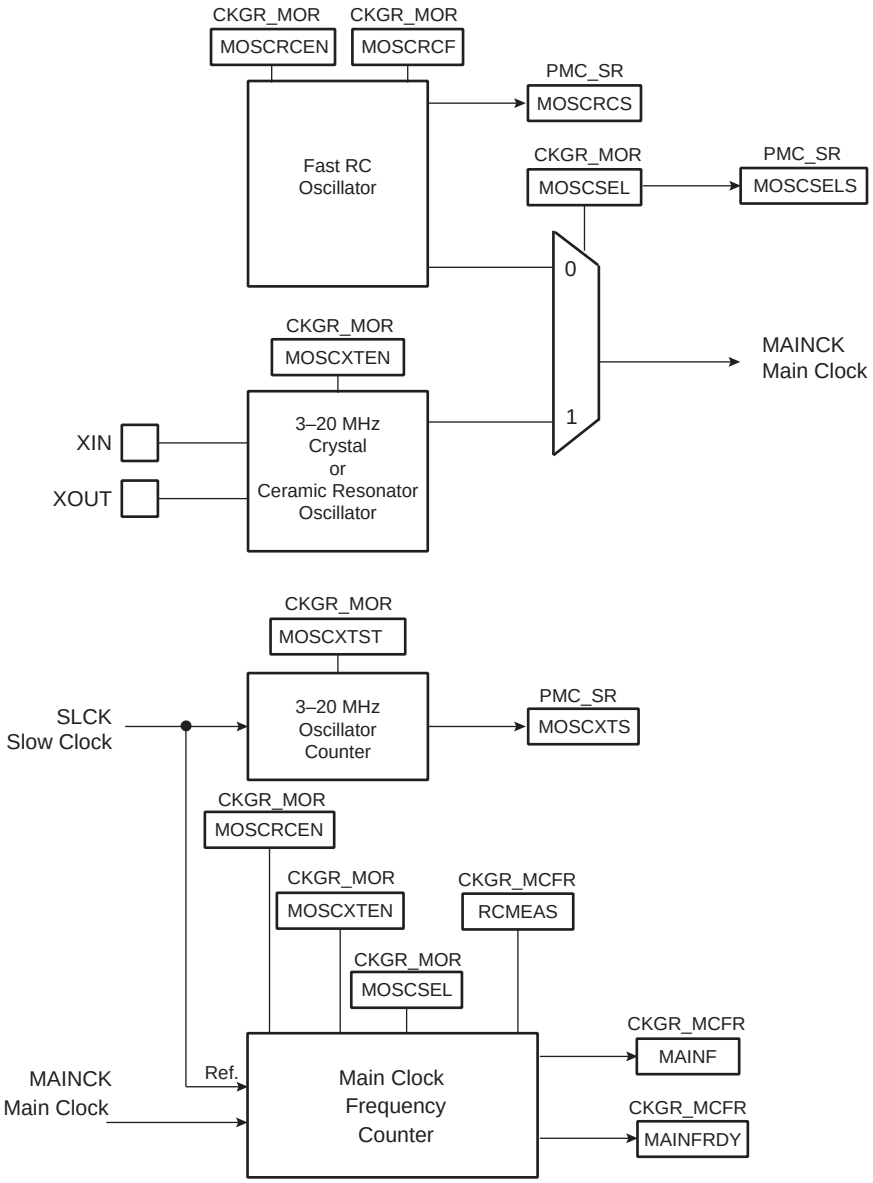
Reverting to the 32 kHz (typical) RC oscillator is only possible by shutting down the VDDIO power supply. If the user does not need the 32768 Hz crystal oscillator, the XIN32 and XOUT32 pins can be left unconnected since by default the XIN32 and XOUT32 system I/O pins are in PIO input mode with pull-up after reset.

The user can also set the 32768 Hz crystal oscillator in Bypass mode instead of connecting a crystal. In this case, the user must provide the external clock signal on XIN32. The input characteristics of the XIN32 pin are given in the section “Electrical Characteristics”. To enter Bypass mode, the OSCBYPASS bit of the Supply Controller Mode Register (SUPC_MR) must be set prior to setting SUPC_CR.XTALSEL.

28.5 Main Clock

Figure 28-3 shows the main clock block diagram.

Figure 28-3. Main Clock Block Diagram



The main clock has two sources:

- A 4/8/12 MHz RC oscillator with a fast start-up time and that is selected by default to start the system
- A 3 to 20 MHz crystal or ceramic resonator-based oscillator which can be bypassed

28.5.1 Embedded 4/8/12 MHz RC Oscillator

After reset, the 4/8/12 MHz RC oscillator is enabled with the 4 MHz frequency selected. This oscillator is selected as the source of MAINCK. MAINCK is the default clock selected to start the system.

The 4/8/12 MHz RC oscillator frequencies are calibrated in production except for the lowest frequency which is not calibrated.

Refer to the section “DC Characteristics”.

The software can disable or enable the 4/8/12 MHz RC oscillator with the MOSCRCEN bit in the Clock Generator Main Oscillator Register (CKGR_MOR).

The output frequency of the RC oscillator can be selected among 4/8/12 MHz. The selection is done via the CKGR_MOR.MOSCRCF field. When changing the 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 PMC_SR.MOSCRCS is set.

When disabling the main clock by clearing the CKGR_MOR.MOSCRCEN bit, the PMC_SR.MOSCRCS bit 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.

When main clock (MAINCK) is not used to drive the processor and frequency monitor (SLCKis used instead), it is recommended to disable the 4/8/12 MHz RC oscillator and 3 to 20 MHz crystal oscillator.

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 memory plane for code. 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 application in PMC_OCR are reset after each power up or peripheral reset.

28.5.2 4/8/12 MHz RC Oscillator Clock Frequency Adjustment

It is possible for the user to adjust the 4/8/12 MHz RC oscillator frequency through PMC_OCR. By default, SEL4/8/12 bits are cleared, so the RC oscillator will be driven with Flash calibration bits which are programmed during chip production.

The user can adjust the trimming of the 4/8/12 MHz RC oscillator through this register. This can be used to compensate derating factors such as temperature and voltage, thus providing greater accuracy.

In order to calibrate the RC oscillator lower frequency, SEL bit must be set to 1 and a frequency value must be configured in the field CAL4. Likewise, SEL8/12 bit must be set to 1 and a trim value must be configured in the field CAL8/12 in order to adjust the other frequencies of the RC oscillator.

However, the adjustment can not be done to the frequency from which the RC oscillator is operating. For example, while running from the lower possible frequency, the user can adjust the other frequencies but not the lowest one.

At any time, it is possible to restart a measurement of the frequency of the selected clock via the RCMEAS bit in Main Clock Frequency Register (CKGR_MCFR). Thus, when CKGR_MCFR.MAINFRDY reads 1, another read access on CKGR_MCFR provides an image of the frequency on CKGR_MCFR.MAINF field. The software can calculate the error with an expected frequency and correct the CAL (or CAL8/CAL12) field accordingly. This may be used to compensate frequency drift due to derating factors such as temperature and/or voltage.

28.5.3 3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator

After reset, the 3 to 20 MHz crystal or ceramic resonator-based oscillator is disabled and is not selected as the source of MAINCK.

As the source of MAINCK, the 3 to 20 MHz crystal or ceramic resonator-based oscillator provides a very precise frequency. The software enables or disables this oscillator in order to reduce power consumption via CKGR_MOR.MOSCXTEN.

When disabling this oscillator by clearing the CKGR_MOR.MOSCXTEN, PMC_SR.MOSCXTS is automatically cleared, indicating the 3 to 20 MHz crystal oscillator is off.

When enabling this oscillator, the user must initiate the start-up time counter. The start-up time depends on the characteristics of the external device connected to this oscillator.

When CKGR_MOR.MOSCXTEN and CKGR_MOR.MOSCXTST are written to enable this oscillator, the XIN and XOUT pins are automatically switched into Oscillator mode. PMC_SR.MOSCXTS is cleared and the counter starts counting down on the slow clock divided by 8 from the CKGR_MOR.MOSCXTST value. Since the CKGR_MOR.MOSCXTST value is coded with 8 bits, the maximum start-up time is about 62 ms.

When the start-up time counter reaches 0, PMC_SR.MOSCXTS is set, indicating that the 3 to 20 MHz crystal oscillator is stabilized. Setting the MOSCXTS bit in the Interrupt Mask Register (PMC_IMR) can trigger an interrupt to the processor.

28.5.4 Main Clock Source Selection

The user can select the source of the main clock from either the 4/8/12 MHz RC oscillator, the 3 to 20 MHz crystal oscillator or the ceramic resonator-based oscillator.

The advantage of the 4/8/12 MHz RC oscillator is its fast start-up time. By default, this oscillator is selected to start the system and when entering Wait mode.

The advantage of the 3 to 20 MHz crystal oscillator or ceramic resonator-based oscillator is its precise frequency.

The selection of the oscillator is made by writing CKGR_MOR.MOSCSEL. The switch of the main clock source is glitch-free, so there is no need to run out of SLCK, PLLACK in order to change the selection. PMC_SR.MOSCSELS indicates when the switch sequence is done.

Setting PMC_IMR.MOSCSELS triggers an interrupt to the processor.

Enabling the 4/8/12 MHz RC oscillator (MOSCRCEN = 1) and changing its frequency (MOSCCRF) at the same time is not allowed.

This oscillator must be enabled first and its frequency changed in a second step.

28.5.5 Bypassing the 3 to 20 MHz Crystal Oscillator

Prior to bypassing the 3 to 20 MHz crystal oscillator, the external clock frequency provided on the XIN pin must be stable and within the values specified in the XIN Clock characteristics in the section “Electrical Characteristics”.

The sequence is as follows:

1. Ensure that an external clock is connected on XIN.
2. Enable the bypass by writing a 1 to CKGR_MOR.MOSCXTBY.
3. Disable the 3 to 20 MHz crystal oscillator by writing a 0 to bit CKGR_MOR.MOSCXTEN.

28.5.6 Main Clock Frequency Counter

The frequency counter is managed by CKGR_MCFR.

During the measurement period, the frequency counter increments at the main clock speed.

A measurement is started in the following cases:

- When the RCMEAS bit of CKGR_MCFR is written to 1.
- When the 4/8/12 MHz RC oscillator 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 source selection is modified

The measurement period ends at the 16th falling edge of slow clock, the MAINFRDY bit in 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 clock cycles during 16 periods of slow clock, so that the frequency of the 4/8/12 MHz RC oscillator or 3 to 20 MHz crystal or ceramic resonator-based oscillator can be determined.

28.5.7 Switching Main Clock between the RC Oscillator and the Crystal Oscillator

When switching the source of the main clock between the RC oscillator and the crystal oscillator, both oscillators must be enabled. After completion of the switch, the unused oscillator can be disabled.

If switching to the crystal oscillator, a check must be carried out to ensure that the oscillator is present and that its frequency is valid. Follow the sequence below:

1. Select the slow clock as MCK by configuring bit CSS = 0 in the Master Clock Register (PMC_MCKR)).
2. Wait for PMC_SR.MCKRDY flag in PMC_SR to rise.
3. Enable the crystal oscillator by setting CKGR_MOR.MOSCXTEN. Configure the CKGR_MOR. MOSCXTST field with the crystal oscillator start-up time as defined in the section “Electrical Characteristics”.
4. Wait for PMC_SR.MOSCXTS flag to rise, indicating the end of a start-up period of the crystal oscillator.
5. Select the crystal oscillator as the source of the main clock by setting CKGR_MOR.MOSCSEL.
6. Read CKGR_MOR.MOSCSEL until its value equals 1.
7. Check the status of PMC_SR.MOSCSELS flag:
   - If MOSCSELS = 1: There is a crystal oscillator connected.

   a. Initiate a new frequency measurement by setting CKGR_MCFR.RCMEAS.
   b. Read CKGR_MCFR.MAINFRDY until its value equals 1.
   c. Read CKGR_MCFR.MAINF and compute the value of the crystal frequency.
   d. If the MAINF value is valid, the main clock can be switched to the crystal oscillator.
   - If MOSCSELS = 0:

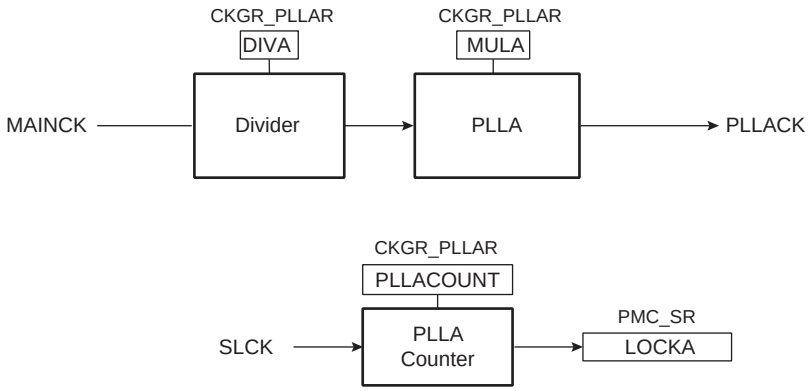
   a. There is no crystal oscillator connected or the crystal oscillator is out of specification.
   b. Select the RC oscillator as the source of the main clock by clearing CKGR_MOR.MOSCSEL.

28.6 Divider and PLL Block

The device features one divider block and one PLL block that permit a wide range of frequencies to be selected on either the master clock, the processor clock or the programmable clock outputs. A 48 MHz clock signal is provided to the embedded USB device port regardless of the frequency of the main clock.

Figure 28-4 shows the block diagram of the divider and PLL blocks.

Figure 28-4. Divider and PLL Block Diagram



28.6.1 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 cleared, the output of the corresponding divider and the PLL output is a continuous signal at level 0. On reset, each DIV field is cleared, thus the corresponding PLL input clock is stuck at 0.

The PLL (PLLA) allows multiplication of the divider's outputs. The PLL clock signal has a frequency that depends on the respective source signal frequency and on the parameters DIV (DIVA) and MUL (MULA). The factor applied to the source signal frequency is (MUL + 1)/DIV. When MUL is written to 0 or DIV = 0, the PLL is disabled and its power consumption is saved. Note that there is a delay of two SLCK clock cycles between the disable command and the real disable of the PLL. Re-enabling the PLL can be performed by writing a value higher than 0 in the MUL field and DIV higher than 0.

Whenever the PLL is re-enabled or one of its parameters is changed, the LOCK (LOCKA) bit in PMC_SR is automatically cleared. The values written in the PLLCOUNT field (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_MCKR.

To avoid programming the PLL with a multiplication factor that is too high, the user can saturate the multiplication factor value sent to the PLL by setting the PLLA_MMAX field in PMC_PMMR.

It is prohibited to change the frequency of the 4/8/12 MHz RC oscillator or to change the source of the main clock in CKGR_MOR while the master clock source is the PLL and the PLL reference clock is the 4/8/12 MHz RC oscillator.

The user must:

1. Switch on the 4/8/12 MHz RC oscillator by writing a 1 to the CSS field of PMC_MCKR.
2. Change the frequency (MOSCRCF) or oscillator selection (MOSCSEL) in CKGR_MOR.
3. Wait for MOSCRCS (if frequency changes) or MOSCSELS (if oscillator selection changes) in PMC_SR.
4. Disable and then enable the PLL.
5. Wait for the LOCK flag in PMC_SR.

6. Switch back to the PLL by writing the appropriate value to the CSS field of PMC_MCKR.

29. Power Management Controller (PMC)

29.1 Description

The Power Management Controller (PMC) optimizes power consumption by controlling all system and user peripheral clocks. The PMC enables/disables the clock inputs to many of the peripherals and the Cortex-M4 processor.

The Supply Controller selects either the embedded 32 kHz RC oscillator or the 32768 Hz 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 4/8/12 MHz RC oscillator running at 4 MHz.

The user can trim the 8 and 12 MHz RC oscillator frequencies by software.

29.2 Embedded Characteristics

The PMC provides the following clocks:

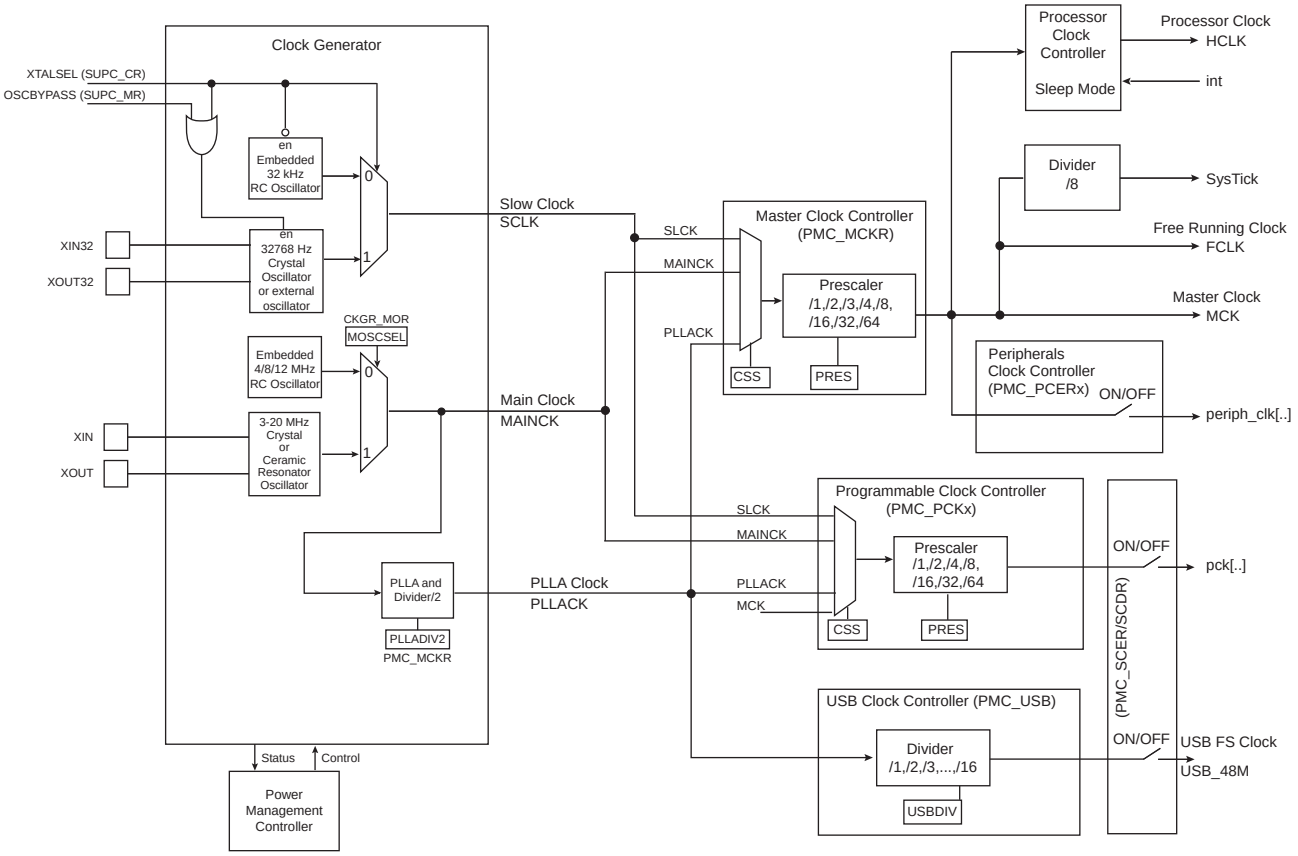
- MCK, the Master Clock, programmable from a few hundred Hz to the maximum operating frequency of the device. It is available to the modules running permanently, such as the Enhanced Embedded Flash Controller.
- Processor Clock (HCLK) , automatically switched off when entering the processor in Sleep Mode
- Free-running processor Clock (FCLK)
- The Cortex-M4 SysTick external clock
- UDP Clock (UDPCK), required by USB Device Port operations
- Peripheral Clocks, provided to the embedded peripherals (USART, SPI, TWI, TC, etc.) and independently controllable.
- Programmable Clock Outputs (PCKx), selected from the clock generator outputs to drive the device PCK pins

The PMC also provides the following features on clocks:

- A 3 to 20 MHz crystal oscillator clock failure detector
- A 32768 Hz crystal oscillator frequency monitor
- A frequency counter on main clock
- An adjustable 4/8/12 MHz RC oscillator frequency

29.3 Block Diagram

Figure 29-1. General Clock Block Diagram



29.4 Master Clock Controller

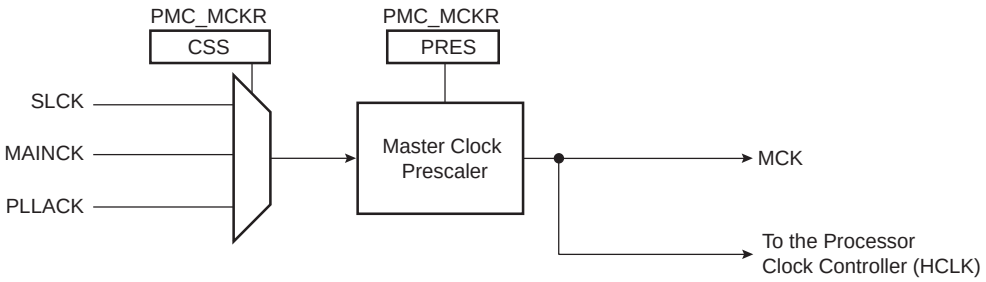
The Master Clock Controller provides selection and division of the master clock (MCK). MCK is the source clock of the peripheral clocks. The master clock is selected from one of the clocks provided by the Clock Generator.

Selecting the slow clock provides a slow clock signal to the whole device. Selecting the main clock saves power consumption of the PLL. The Master Clock Controller is made up of a clock selector and a prescaler.

The master clock selection is made by writing the CSS field (Clock Source Selection) in PMC_MCKR. 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 29-2. Master Clock Controller



29.5 Processor Clock Controller

The PMC features a Processor Clock Controller (HCLK) that implements the processor Sleep mode. These 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 Controller HCLK is enabled after a reset and is automatically re-enabled by any enabled interrupt. The processor Sleep mode is entered 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.

29.6 SysTick Clock

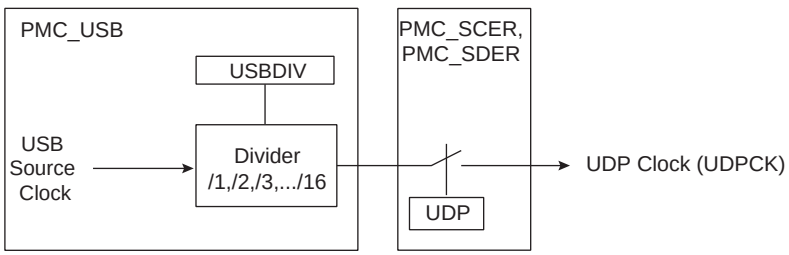
The SysTick calibration value is fixed to 15000 which allows the generation of a time base of 1 ms with SysTick clock to the maximum frequency on MCK divided by 8.

29.7 USB Clock Controller

The user can select the PLLA output as the USB source clock by writing the USBS bit in PMC_USB. If using the USB, the user must program the PLL to generate an appropriate frequency depending on the USBDIV bit in the USB Clock Register (PMC_USB).

When the PLL output is stable, i.e., the LOCK bit is set, the USB device FS clock can be enabled by setting the UDP, bit in the System Clock Enable Register (PMC_SCER). To save power on this peripheral when it is not used, the user can set the UDP, bit in the System Clock Disable Register (PMC_SCDR). The UDP, bit in the System Clock Status Register (PMC_SCSR) gives the activity of this clock. The USB device port requires both the 48 MHz signal and the peripheral clock. The USB peripheral clock may be controlled by means of the Master Clock Controller.

Figure 29-3. USB Clock Controller



29.8 Peripheral Clock Controller

The PMC 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.

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.

29.9 Free-Running Processor Clock

The free-running processor clock (FCLK)used for sampling interrupts and clocking debug blocks ensures that interrupts can be sampled, and sleep events can be traced, while the processor is sleeping. It is connected to master clock (MCK).

29.10 Programmable Clock Output Controller

The PMC controls three signals to be output on external pins, PCKx. Each signal can be independently programmed via the Programmable Clock Registers (PMC_PCKx).

PCKx can be independently selected between the slow clock (SLCK), the main clock (MAINCK), the PLLA clock (PLLACK),and the master clock (MCK) by writing the CSS field in PMC_PCKx. Each output signal can also be divided by a power of 2 between 1 and 64 by writing the PRES (Prescaler) field in PMC_PCKx.

Each output signal can be enabled and disabled by writing a 1 to the corresponding PCKx bit of PMC_SCER and PMC_SCDR, respectively. Status of the active programmable output clocks are given in the PCKx bits of PMC_SCSR.

The PCKRDYx status flag 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.

29.11 Fast Startup

At exit from Wait mode, the device allows the processor to restart in less than 10 microseconds only if the C-code function that manages the Wait mode entry and exit is linked to and executed from on-chip SRAM.

The fast startup time cannot be achieved if the first instruction after an exit is located in the embedded Flash.

If fast startup is not required, or if the first instruction after a Wait mode exit is located in embedded Flash, see Section 29.12 ”Startup from Embedded Flash”.

Prior to instructing the device to enter Wait mode:

1. Select the 4/8/12 MHz RC oscillator as the master clock source (the CSS field in PMC_MCKR must be written to 1).
2. Disable the PLL if enabled.
3. Clear the internal wake-up sources.

The system enters Wait mode either by setting the WAITMODE bit in CKGR_MOR, or by executing the WaitForEvent (WFE) instruction of the processor while the LPM bit is at 1 in PMC_FSMR. Immediately after setting the WAITMODE bit or using the WFE instruction, wait for the MCKRDY bit to be set in PMC_SR.

A fast startup is enabled upon the detection of a programmed level on one the wake-up input pins WKUPx (up to 16 inputs, see Peripheral Signal Multiplexing on I/O Lines section for exact number) or upon an active alarm from the RTC, RTT and USB Controller. The polarity of each wake-up input is programmable by writing the PMC Fast Startup Polarity Register (PMC_FSPR).

WARNING: The duration of the WKUPx pins active level must be greater than four main clock cycles.

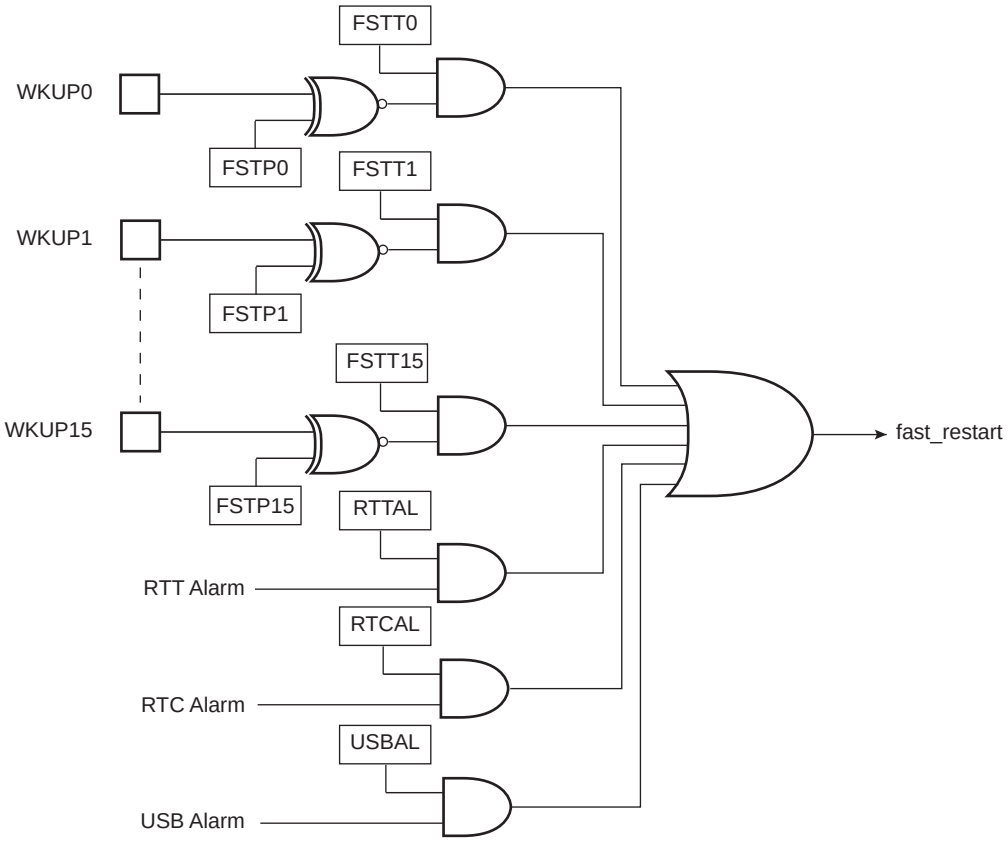
The fast startup circuitry, as shown in Figure 29-4, is fully asynchronous and provides a fast startup signal to the PMC. As soon as the fast startup signal is asserted, the embedded 4/8/12 MHz RC oscillator restarts automatically.

When entering Wait mode, the embedded Flash can be placed in one of the Low-power modes (Deep-power-down or Standby modes) depending on the configuration of the FLPM field in the PMC_FSMR. The FLPM field can be programmed at anytime and its value will be applied to the next Wait mode period.

The power consumption reduction is optimal when configuring 1 (Deep-power-down mode) in field FLPM. If 0 is programmed (Standby mode), the power consumption is slightly higher than in Deep-power-down mode.

When programming 2 in field FLPM, the Wait mode Flash power consumption is equivalent to that of the Active mode when there is no read access on the Flash.

Figure 29-4. Fast Startup Circuitry



Each wake-up input pin and alarm can be enabled to generate a fast startup event by setting the corresponding bit in 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.

29.12 Startup from Embedded Flash

The inherent start-up time of the embedded Flash cannot provide a fast startup of the system.

If system fast start-up time is not required, the first instruction after a Wait mode exit can be located in the embedded Flash. Under these conditions, prior to entering Wait mode, the Flash controller must be programmed to perform access in 0 wait-state. Refer to the section “Enhanced Embedded Flash Controller (EEFC)”.

The procedure and conditions to enter Wait mode and the circuitry to exit Wait mode are strictly the same as fast startup (see Section 29.11 ”Fast Startup”).

29.13 Main Clock Failure Detector

The clock failure detector monitors the 3 to 20 MHz crystal oscillator or ceramic resonator-based oscillator to identify a failure of this oscillator when selected as main clock.

The clock failure detector can be enabled or disabled by bit CFDEN in CKGR_MOR. After a VDDCORE reset, the detector is disabled. However, if the oscillator is disabled (MOSCXTEN = 0), the detector is also disabled.

To initialize the clock failure detector, follow the sequence below:

1. The 4/8/12 MHz RC oscillator must be selected as the source of MAINCK.
2. MCK must select MAINCK.

3. Enable the clock failure detector by setting the bit CFDEN.
4. PMC_SR must be read two slow clock cycles after enabling the clock failure detector. The value read is meaningless.

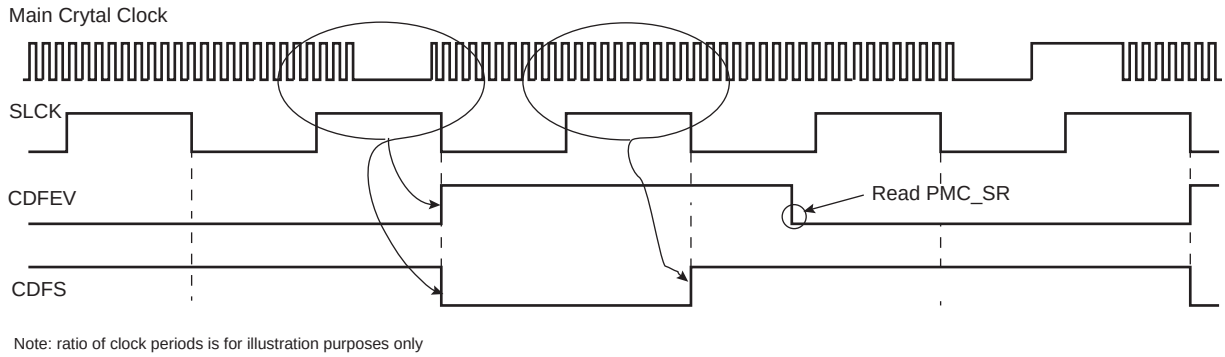
The clock failure detector is now initialized and MCK can select another clock source by programming the CSS field in PMC_MCKR.

A failure is detected by means of a counter incrementing on the main clock and detection logic is triggered by the 32 kHz (typical) RC oscillator which is automatically enabled when CFDEN=1.

The counter is cleared when the 32 kHz (typical) RC oscillator clock signal is low and enabled when the signal is high. Thus, the failure detection time is one RC oscillator period. If, during the high level period of the 32 kHz (typical) RC oscillator clock signal, less than eight 3 to 20 MHz crystal oscillator clock periods have been counted, then a failure is reported.

If a failure of the main clock is detected, bit CFDEV in PMC_SR indicates a failure event and generates an interrupt if the corresponding interrupt source is enabled. The interrupt remains active until a read occurs in PMC_SR. The user can know the status of the clock failure detection at any time by reading the CFDS bit in PMC_SR.

Figure 29-5. Clock Failure Detection (Example)



Note: ratio of clock periods is for illustration purposes only

If the 3 to 20 MHz crystal oscillator or ceramic resonator-based oscillator is selected as the source clock of MAINCK (MOSCSEL in CKGR_MOR = 1), and if MCK source is PLLACK (CSS = 2), a clock failure detection automatically forces MAINCK to be the source clock for MCK. Then, regardless of the PMC configuration, a clock failure detection automatically forces the 4/8/12 MHz RC oscillator to be the source clock for MAINCK. If this oscillator is disabled when a clock failure detection occurs, it is automatically re-enabled by the clock failure detection mechanism.

It takes two 32 kHz (typical) RC oscillator clock cycles to detect and switch from the 3 to 20 MHz crystal oscillator, to the 4/8/12 MHz RC oscillator if the source master clock (MCK) is main clock (MAINCK), or three 32 kHz (typical) RC oscillator clock cycles if the source of MCK is PLLACK.

A clock failure detection activates a fault output that is connected to the Pulse Width Modulator (PWM) Controller. With this connection, the PWM controller is able to force its outputs and to protect the driven device, if a clock failure is detected.

The user can know the status of the clock failure detector at any time by reading the FOS bit in PMC_SR.

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).

29.14 32768 Hz Crystal Oscillator Frequency Monitor

The frequency of the 32768 Hz crystal oscillator can be monitored by means of logic driven by the 4/8/12 MHz RC oscillator known as a reliable clock source. This function is enabled by configuring the XT32KFME bit of CKGR_MOR. The SEL4/SEL8/SEL12 bits of PMC_OCR must be cleared.

An error flag (XT32KERR in PMC_SR) is asserted when the 32768 Hz crystal oscillator frequency is out of the ±10% nominal frequency value (i.e., 32768 Hz). The error flag can be cleared only if the slow clock frequency monitoring is disabled.
When the 4/8/12 MHz RC oscillator frequency is 4 MHz, the accuracy of the measurement is ±40% as this frequency is not trimmed during production. Therefore, ±10% accuracy is obtained only if the RC oscillator frequency is configured for 8 or 12 MHz.

The monitored clock frequency is declared invalid if at least four consecutive clock period measurement results are over the nominal period ±10%.

Due to the possible frequency variation of the embedded 4/8/12 MHz RC oscillator acting as reference clock for the monitor logic, any 32768 Hz crystal oscillator frequency deviation over ±10% of the nominal frequency is systematically reported as an error by the XT32KERR bit in PMC_SR. Between -1% and -10% and +1% and +10%, the error is not systematically reported.

Thus only a crystal running at 32768 Hz frequency ensures that the error flag will not be asserted. The permitted drift of the crystal is 10000 ppm (1%), which allows any standard crystal to be used.

If the 4/8/12 MHz RC frequency needs to be changed while the slow clock frequency monitor is operating, the monitoring must be stopped prior to changing the 4/8/12 MHz RC frequency. It can then be re-enabled as soon as MOSCRCS is set in PMC_SR.

The error flag can be defined as an interrupt source of the PMC by setting the XT32KERR bit of PMC_IER.

29.15 Programming Sequence

1. If the 3 to 20 MHz crystal oscillator is not required, the PLL and divider can be directly configured (Step 6.) else this oscillator must be started (Step 2.).
2. Enable the 3 to 20 MHz crystal oscillator by setting the MOSCXTEN field in CKGR_MOR:

   The user can define a start-up time. This is done 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 PMC_SR to be set. This is done either by polling MOSCXTS in PMC_SR, or by waiting for the interrupt line to be raised if the associated interrupt source (MOSCXTS) has been enabled in PMC_IER.
3. Switch the MAINCK to the 3 to 20 MHz crystal oscillator by setting MOSCSEL in CKGR_MOR.
4. Wait for the MOSCSELS to be set in PMC_SR to ensure the switch is complete.
5. Check the main clock frequency:

   This main clock frequency can be measured via CKGR_MCFR.

   Read CKGR_MCFR until the MAINFRDY field is set, after which the user can read the MAINF field in CKGR_MCFR by performing an additional read. This provides the number of main clock cycles that have been counted during a period of 16 slow clock cycles.

   If MAINF = 0, switch the MAINCK to the 4/8/12 MHz RC Oscillator by clearing MOSCSEL in CKGR_MOR. If MAINF ≠ 0, proceed to Step 6.
6. Set PLLx and Divider (if not required, proceed to Step 7.):

   In the names PLLx, DIVx, MULx, LOCKx, PLLxCOUNT, and CKGR_PLLxR, ‘x’ represents A.

   All parameters needed to configure PLLx and the divider are located in CKGR_PLLxR.

   The DIVx field is used to control the divider itself. This parameter can be programmed between 0 and 127. Divider output is divider input divided by DIVx parameter. By default, DIVx field is cleared which means that the divider and PLLx are turned off.

   The MULx field is the PLLx multiplier factor. This parameter can be programmed between 0 and 80. If MULx is cleared, PLLx will be turned off, otherwise the PLLx output frequency is PLLx input frequency multiplied by (MULx + 1).

The PLLxCOUNT field specifies the number of slow clock cycles before the LOCKx bit is set in the PMC_SR after CKGR_PLLxR has been written.

Once CKGR_PLLxR has been written, the user must wait for the LOCKx bit to be set in the PMC_SR. This can be done either by polling LOCKx in PMC_SR or by waiting for the interrupt line to be raised if the associated interrupt source (LOCKx) has been enabled in PMC_IER. All fields in CKGR_PLLxR can be programmed in a single write operation. If at some stage one of the following parameters, MULx or DIVx is modified, the LOCKx bit goes low to indicate that PLLx is not yet ready. When PLLx is locked, LOCKx is set again. The user must wait for the LOCKx bit to be set before using the PLLx output clock.

7. Select the master clock and processor clock

   The master clock and the processor clock are configurable via PMC_MCKR.

   The CSS field is used to select the clock source of the master clock and processor clock dividers. By default, the selected clock source is the main clock.

   The PRES field is used to define the processor clock and master clock prescaler. The user can choose between different values (1, 2, 3, 4, 8, 16, 32, 64). Prescaler output is the selected clock source frequency divided by the PRES value.

   Once the PMC_MCKR has been written, the user must wait for the MCKRDY bit to be set in the PMC_SR. This can be done either by polling MCKRDY in PMC_SR or by waiting for the interrupt line to be raised if the associated interrupt source (MCKRDY) has been enabled in PMC_IER. PMC_MCKR must not be programmed in a single write operation. The 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, parameters CSS or PRES are modified, the MCKRDY bit goes low to indicate that the master clock and the processor clock are not yet ready. 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_PLLxR, the MCKRDY flag will go low while PLLx is unlocked. Once PLLx is locked again, LOCKx goes high and MCKRDY is set. While PLLx is unlocked, the master clock selection is automatically changed to slow clock for PLLA. For further information, see Section 29.16.2 ”Clock Switching Waveforms”.

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.

8. Select the 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. Three programmable clocks can be used. PMC_SCSR indicates which programmable clock is enabled. By default all programmable clocks are disabled.

PMC_PCKx registers are used to configure programmable clocks.

The CSS field is used to select the programmable clock divider source. Several clock options are available: main clock, slow clock, master clock, PLLACK. The slow clock is the default clock source.

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 value is cleared which means that PCKx is equal to slow clock.

Once PMC_PCKx register has been configured, the corresponding programmable clock must be enabled and the user is constrained to wait for the PCKRDYx bit to be set in the PMC_SR. This can be done either by polling PCKRDYx in PMC_SR or by waiting for the interrupt line to be raised if the associated interrupt source (PCKRDYx) has been enabled in PMC_IER. 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.

9. Enable the 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.

29.16 Clock Switching Details

29.16.1 Master Clock Switching Timings

Table 29-1 gives 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 29-1. Clock Switching Timings (Worst Case)

From To	Main Clock	SLCK	PLL Clock
Main Clock	–	4 x SLCK + 2.5 x Main Clock	3 x PLL Clock + 4 x SLCK + 1 x Main Clock
SLCK	0.5 x Main Clock + 4.5 x SLCK	–	3 x PLL Clock + 5 x SLCK
PLL Clock	0.5 x Main Clock + 4 x SLCK + PLLCOUNT x SLCK + 2.5 x PLLx Clock	2.5 x PLL Clock + 5 x SLCK + PLLCOUNT x SLCK	2.5 x PLL Clock + 4 x SLCK + PLLCOUNT x SLCK

Notes: 1. PLL designates the PLLA .
2. PLLCOUNT designates PLLACOUNT .

29.16.2 Clock Switching Waveforms

Figure 29-6. Switch Master Clock from Slow Clock to PLLx Clock

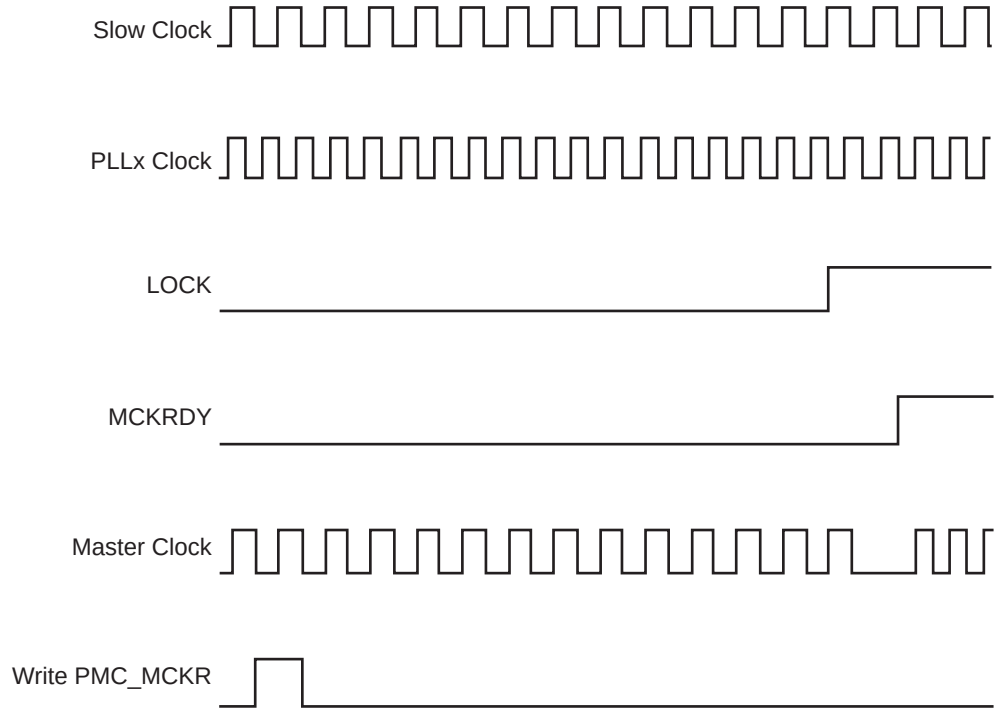


Figure 29-7. Switch Master Clock from Main Clock to Slow Clock

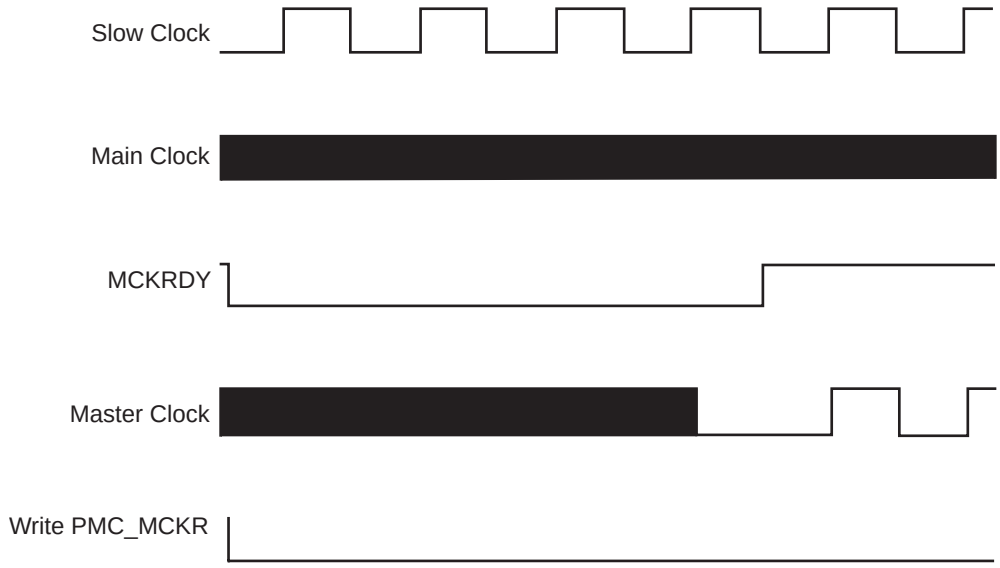


Figure 29-8. Change PLLx Programming

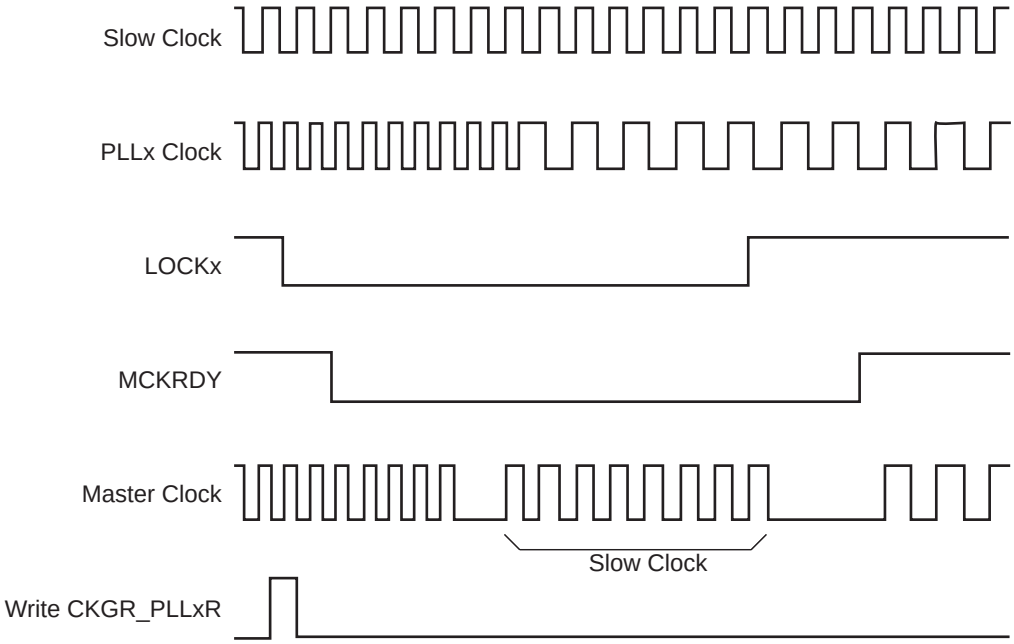
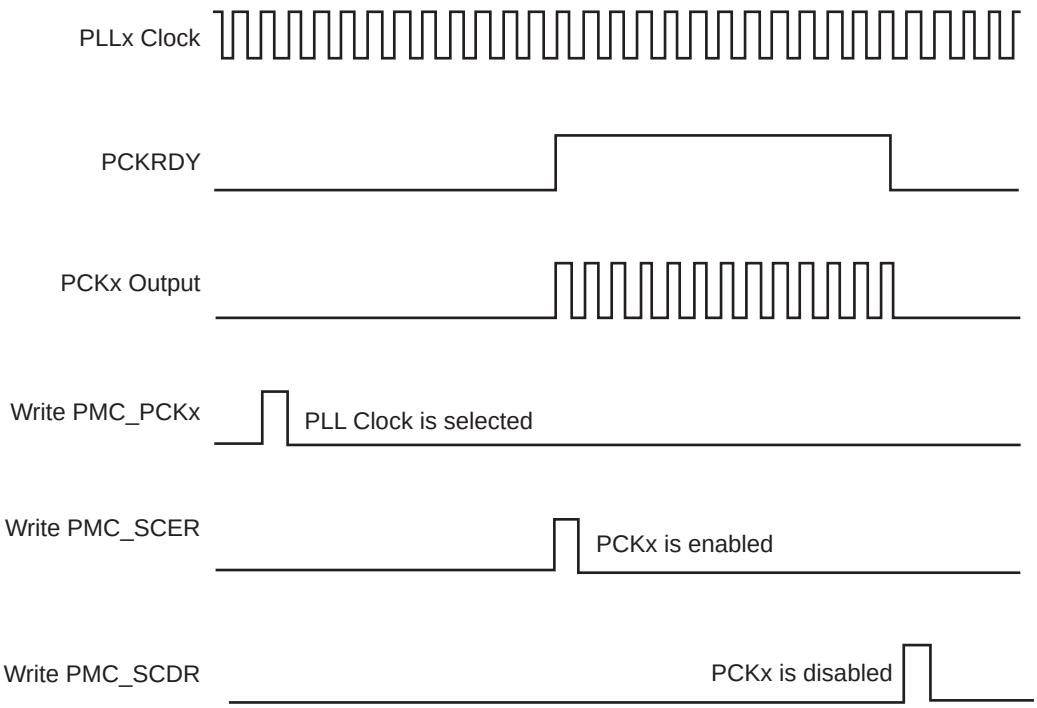


Figure 29-9. Programmable Clock Output Programming



29.17 Register Write Protection

To prevent any single software error from corrupting PMC behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the "PMC Write Protection Mode Register" (PMC_WPMR).

If a write access to a write-protected register is detected, the WPVS flag in the "PMC Write Protection Status Register" (PMC_WPSR) is set and the field WPVSRC indicates the register in which the write access has been attempted.

The WPVS bit is automatically cleared after reading the PMC_WPSR.

The following registers can be write-protected:

- "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 Main Clock Frequency 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"
- "PMC Oscillator Calibration Register"
- "PLL Maximum Multiplier Value Register"

29.18 Power Management Controller (PMC) User Interface

Table 29-2. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	System Clock Enable Register	PMC_SCER	Write-only	–
0x0004	System Clock Disable Register	PMC_SCDR	Write-only	–
0x0008	System Clock Status Register	PMC_SCSR	Read-only	0x0000_0001
0x000C	Reserved	–	–	–
0x0010	Peripheral Clock Enable Register 0	PMC_PCER0	Write-only	–
0x0014	Peripheral Clock Disable Register 0	PMC_PCDR0	Write-only	–
0x0018	Peripheral Clock Status Register 0	PMC_PCSR0	Read-only	0x0000_0000
0x001C	Reserved	–	–	–
0x0020	Main Oscillator Register	CKGR_MOR	Read/Write	0x0000_0008
0x0024	Main Clock Frequency Register	CKGR_MCFR	Read/Write	0x0000_0000
0x0028	PLLA Register	CKGR_PLLAR	Read/Write	0x0000_3F00
0x002C	Reserved	–	–	–
0x0030	Master Clock Register	PMC_MCKR	Read/Write	0x0000_0001
0x0034	Reserved	–	–	–
0x0038	USB Clock Register	PMC_USB	Read/Write	0x0000_0000
0x003C	Reserved	–	–	–
0x0040	Programmable Clock 0 Register	PMC_PCK0	Read/Write	0x0000_0000
0x0044	Programmable Clock 1 Register	PMC_PCK1	Read/Write	0x0000_0000
0x0048	Programmable Clock 2 Register	PMC_PCK2	Read/Write	0x0000_0000
0x004C–0x005C	Reserved	–	–	–
0x0060	Interrupt Enable Register	PMC_IER	Write-only	–
0x0064	Interrupt Disable Register	PMC_IDR	Write-only	–
0x0068	Status Register	PMC_SR	Read-only	0x0003_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 Protection Mode Register	PMC_WPMR	Read/Write	0x0000_0000
0x00E8	Write Protection Status Register	PMC_WPSR	Read-only	0x0000_0000
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	Reserved	–	–	–

Table 29-2. Register Mapping (Continued)

Offset	Register	Name	Access	Reset
0x0110	Oscillator Calibration Register	PMC_OCR	Read/Write	0x0040_4040
0x0114–0x0120	Reserved	–	–	–
0x0130	PLL Maximum Multiplier Value Register	PMC_PMMR	Read/Write	0x07FF_07FF
0x0134–0x144	Reserved	–	–	–

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

29.18.1 PMC System Clock Enable Register

Name: PMC_SCER

Address: 0x400E0400

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
UDP	–	–	–	–	–	–	–

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **UDP: USB Device Port Clock Enable**

0: No effect.

1: Enables the 48 MHz clock (UDPCK) of the USB Device Port.

- **PCKx: Programmable Clock x Output Enable**

0: No effect.

1: Enables the corresponding Programmable Clock output.

29.18.2 PMC System Clock Disable Register

Name: PMC_SCDR

Address: 0x400E0404

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
UDP	–	–	–	–	–	–	–

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **UDP: USB Device Port Clock Disable**

0: No effect.

1: Disables the 48 MHz clock (UDPCK) of the USB Device Port.

- **PCKx: Programmable Clock x Output Disable**

0: No effect.

1: Disables the corresponding Programmable Clock output.

29.18.3 PMC System Clock Status Register

Name: PMC_SCSR

Address: 0x400E0408

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
UDP	–	–	–	–	–	–	–

- **UDP: USB Device Port Clock Status**

0: The 48 MHz clock (UDPCK) of the USB Device Port is disabled.

1: The 48 MHz clock (UDPCK) of the USB Device Port is enabled.

- **PCKx: Programmable Clock x Output Status**

0: The corresponding Programmable Clock output is disabled.

1: The corresponding Programmable Clock output is enabled.

29.18.4 PMC Peripheral Clock Enable Register 0

Name: PMC_PCER0

Address: 0x400E0410

Access: Write-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	6	5	4	3	2	1	0
PID7	–	–	–	–	–	–	–

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **PIDx: Peripheral Clock x Enable**

0: No effect.

1: Enables the corresponding peripheral clock.

Note: PIDx refers to identifiers defined in the section “Peripheral Identifiers”. Other peripherals can be enabled in PMC_PCER1 (Section 29.18.22 ”PMC Peripheral Clock Enable Register 1”).

Note: Programming the control bits of the Peripheral ID that are not implemented has no effect on the behavior of the PMC.

29.18.5 PMC Peripheral Clock Disable Register 0

Name: PMC_PCDR0

Address: 0x400E0414

Access: Write-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	6	5	4	3	2	1	0
PID7	–	–	–	–	–	–	–

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **PIDx: Peripheral Clock x Disable**

0: No effect.

1: Disables the corresponding peripheral clock.

Note: PIDx refers to identifiers defined in the section “Peripheral Identifiers”. Other peripherals can be disabled in PMC_PCDR1 (Section 29.18.23 ”PMC Peripheral Clock Disable Register 1”).

29.18.6 PMC Peripheral Clock Status Register 0

Name: PMC_PCSR0

Address: 0x400E0418

Access: Read-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	6	5	4	3	2	1	0
PID7	–	–	–	–	–	–	–

- **PIDx: Peripheral Clock x Status**

0: The corresponding peripheral clock is disabled.

1: The corresponding peripheral clock is enabled.

Note: PIDx refers to identifiers defined in the section “Peripheral Identifiers”. Other peripherals status can be read in PMC_PCSR1 (Section 29.18.24 ”PMC Peripheral Clock Status Register 1”).

29.18.7 PMC Clock Generator Main Oscillator Register

Name: CKGR_MOR

Address: 0x400E0420

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	XT32KFME	CFDEN	MOSCSEL

23	22	21	20	19	18	17	16
KEY							

15	14	13	12	11	10	9	8
MOSCXTST							

7	6	5	4	3	2	1	0
–	MOSCRCF			MOSCRCEN	WAITMODE	MOSCXTBY	MOSCXTEN

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **MOSCXTEN: 3 to 20 MHz Crystal Oscillator Enable**

A crystal must be connected between XIN and XOUT.

0: The 3 to 20 MHz crystal oscillator is disabled.

1: The 3 to 20 MHz crystal oscillator is enabled. MOSCXTBY must be cleared.

When MOSCXTEN is set, the MOSCXTS flag is set once the crystal oscillator start-up time is achieved.

- **MOSCXTBY: 3 to 20 MHz Crystal Oscillator Bypass**

0: No effect.

1: The 3 to 20 MHz crystal oscillator is bypassed. MOSCXTEN must be cleared. 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 resets the MOSCXTS flag.

Note: When the crystal oscillator bypass is disabled (MOSCXTBY = 0), the MOSCXTS flag must be read at 0 in PMC_SR before enabling the crystal oscillator (MOSCXTEN = 1).

- **WAITMODE: Wait Mode Command (Write-only)**

0: No effect.

1: Puts the device in Wait mode.

- **MOSCRCEN: 4/8/12 MHz RC Oscillator Enable**

0: The 4/8/12 MHz RC oscillator is disabled.

1: The 4/8/12 MHz RC oscillator is enabled.

When MOSCRCEN is set, the MOSCRCS flag is set once the RC oscillator start-up time is achieved.

- **MOSCRCF: 4/8/12 MHz RC Oscillator Frequency Selection**

At startup, the RC oscillator frequency is 4 MHz.

Value	Name	Description
0	4_MHz	The RC oscillator frequency is at 4 MHz (default)
1	8_MHz	The RC oscillator frequency is at 8 MHz
2	12_MHz	The RC oscillator frequency is at 12 MHz

Note: MOSCRCF must be changed only if MOSCRCS is set in the PMC_SR. Therefore MOSCRCF and MOSCRCEN cannot be changed at the same time.

- **MOSCXTST: 3 to 20 MHz Crystal Oscillator Start-up Time**

Specifies the number of slow clock cycles multiplied by 8 for the crystal oscillator start-up time.

- **KEY: Write Access Password**

Value	Name	Description
0x37	PASSWD	Writing any other value in this field aborts the write operation. Always reads as 0.

- **MOSCSEL: Main Clock Oscillator Selection**

0: The 4/8/12 MHz RC oscillator is selected.

1: The 3 to 20 MHz crystal oscillator is selected.

- **CFDEN: Clock Failure Detector Enable**

0: The clock failure detector is disabled.

1: The clock failure detector is enabled.

Note: 1. The 32 kHz (typical) RC oscillator is automatically enabled when CFDEN=1.
2. Refer to Section 29.13 ”Main Clock Failure Detector” for CFDEN initialization.

- **XT32KFME: 32768 Hz Crystal Oscillator Frequency Monitoring Enable**

0: The 32768 Hz crystal oscillator frequency monitoring is disabled.

1: The 32768 Hz crystal oscillator frequency monitoring is enabled.

29.18.8 PMC Clock Generator Main Clock Frequency Register

Name: CKGR_MCFR

Address: 0x400E0424

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	RCMEAS	–	–	–	MAINFRDY

15	14	13	12	11	10	9	8
MAINF							

7	6	5	4	3	2	1	0
MAINF							

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **MAINF: Main Clock Frequency**

Gives the number of main clock cycles within 16 slow clock periods. To calculate the frequency of the measured clock: $f_{MAINCK} = (MAINF \times f_{SLCK}) / 16$

where frequency is in MHz.

- **MAINFRDY: Main Clock Frequency Measure Ready**

0: MAINF value is not valid or the measured oscillator is disabled or a measure has just been started by means of RCMEAS.

1: The measured oscillator has been enabled previously and MAINF value is available.

Note: To ensure that a correct value is read on the MAINF field, the MAINFRDY flag must be read at 1 then another read access must be performed on the register to get a stable value on the MAINF field.

- **RCMEAS: Restart Main Clock Source Frequency Measure (write-only)**

0: No effect.

1: Restarts measuring of the frequency of the main clock source. MAINF will carry the new frequency as soon as a low to high transition occurs on the MAINFRDY flag.

The measure is performed on the main frequency (i.e. not limited to RC oscillator only), but if the main clock frequency source is the 3 to 20 MHz crystal oscillator, the restart of measuring is not needed because of the well known stability of crystal oscillators.

29.18.9 PMC Clock Generator PLLA Register

Name: CKGR_PLLAR

Address: 0x400E0428

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	ONE	–	–	MULA		

23	22	21	20	19	18	17	16
MULA							

15	14	13	12	11	10	9	8
–	–	PLLACOUNT					

7	6	5	4	3	2	1	0
DIVA							

Possible limitations on PLLA input frequencies and multiplier factors should be checked before using the PMC.

Warning: Bit 29 must always be set to 1 when programming the CKGR_PLLAR.

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **DIVA:** PLLA Front_End Divider

0: Divider output is stuck at 0 and PLLA is disabled.

1: Divider is bypassed (divide by 1) PLLA is enabled

2–255: Clock is divided by DIVA

- **PLLACOUNT: PLLA Counter**

Specifies the number of Slow Clock cycles before the LOCKA bit is set in PMC_SR after CKGR_PLLAR is written.

- **MULA: PLLA Multiplier**

0: The PLLA is deactivated (PLLA also disabled if DIVA = 0).

1 up to 80 = The PLLA Clock frequency is the PLLA input frequency multiplied by MULA + 1.

Unlisted values are forbidden.

- **ONE: Must Be Set to 1**

Bit 29 must always be set to 1 when programming the CKGR_PLLAR.

29.18.10 PMC Master Clock Register

Name: PMC_MCKR

Address: 0x400E0430

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	PLLADIV2	–	–	–	–
7	6	5	4	3	2	1	0
–	PRES			–	–	CSS	

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **CSS: Master Clock Source Selection**

Value	Name	Description
0	SLOW_CLK	Slow Clock is selected
1	MAIN_CLK	Main Clock is selected
2	PLLA_CLK	PLLA Clock is selected

- **PRES: Processor Clock Prescaler**

Value	Name	Description
0	CLK_1	Selected clock
1	CLK_2	Selected clock divided by 2
2	CLK_4	Selected clock divided by 4
3	CLK_8	Selected clock divided by 8
4	CLK_16	Selected clock divided by 16
5	CLK_32	Selected clock divided by 32
6	CLK_64	Selected clock divided by 64
7	CLK_3	Selected clock divided by 3

- **PLLADIV2: PLLA Divisor by 2**

PLLADIV2	PLLA Clock Division
0	PLLA clock frequency is divided by 1.
1	PLLA clock frequency is divided by 2.

29.18.11 PMC USB Clock Register

Name: PMC_USB

Address: 0x400E0438

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **USBDIV: Divider for USB Clock**

USB Clock is Input clock divided by USBDIV + 1.

29.18.12 PMC Programmable Clock Register

Name: PMC_PCKx

Address: 0x400E0440

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	PRES			–	CSS		

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **CSS: Master Clock Source Selection**

Value	Name	Description
0	SLOW_CLK	Slow Clock is selected
1	MAIN_CLK	Main Clock is selected
2	PLLA_CLK	PLLA Clock is selected
4	MCK	Master Clock is selected

- **PRES: Programmable Clock Prescaler**

Value	Name	Description
0	CLK_1	Selected clock
1	CLK_2	Selected clock divided by 2
2	CLK_4	Selected clock divided by 4
3	CLK_8	Selected clock divided by 8
4	CLK_16	Selected clock divided by 16
5	CLK_32	Selected clock divided by 32
6	CLK_64	Selected clock divided by 64

29.18.13 PMC Interrupt Enable Register

Name: PMC_IER

Address: 0x400E0460

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	XT32KERR	–	–	CFDEV	MOSCRCS	MOSCSELS
15	14	13	12	11	10	9	8
–	–	–	–	–	PCKRDY2	PCKRDY1	PCKRDY0
7	6	5	4	3	2	1	0
–	–	–	–	MCKRDY	–	LOCKA	MOSCXTS

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **MOSCXTS: 3 to 20 MHz Crystal Oscillator Status Interrupt Enable**
- **LOCKA: PLLA Lock Interrupt Enable**
- **MCKRDY: Master Clock Ready Interrupt Enable**
- **PCKRDYx: Programmable Clock Ready x Interrupt Enable**
- **MOSCSELS: Main Clock Source Oscillator Selection Status Interrupt Enable**
- **MOSCRCS: 4/8/12 MHz RC Oscillator Status Interrupt Enable**
- **CFDEV: Clock Failure Detector Event Interrupt Enable**
- **XT32KERR: 32768 Hz Crystal Oscillator Error Interrupt Enable**

29.18.14 PMC Interrupt Disable Register

Name: PMC_IDR

Address: 0x400E0464

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	XT32KERR	–	–	CFDEV	MOSCRCS	MOSCSELS
15	14	13	12	11	10	9	8
–	–	–	–	–	PCKRDY2	PCKRDY1	PCKRDY0
7	6	5	4	3	2	1	0
–	–	–	–	MCKRDY	–	LOCKA	MOSCXTS

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Disables the corresponding interrupt.

- **MOSCXTS: 3 to 20 MHz Crystal Oscillator Status Interrupt Disable**
- **LOCKA: PLLA Lock Interrupt Disable**
- **MCKRDY: Master Clock Ready Interrupt Disable**
- **PCKRDYx: Programmable Clock Ready x Interrupt Disable**
- **MOSCSELS: Main Clock Source Oscillator Selection Status Interrupt Disable**
- **MOSCRCS: 4/8/12 MHz RC Oscillator Status Interrupt Disable**
- **CFDEV: Clock Failure Detector Event Interrupt Disable**
- **XT32KERR: 32768 Hz Oscillator Error Interrupt Disable**

29.18.15 PMC Status Register

Name: PMC_SR

Address: 0x400E0468

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	XT32KERR	FOS	CFDS	CFDEV	MOSCRCS	MOSCSELS

15	14	13	12	11	10	9	8
–	–	–	–	–	PCKRDY2	PCKRDY1	PCKRDY0

7	6	5	4	3	2	1	0
OSCSELS	–	–	–	MCKRDY	–	LOCKA	MOSCXTS

- **MOSCXTS: 3 to 20 MHz Crystal Oscillator Status**

0: 3 to 20 MHz crystal oscillator is not stabilized.

1: 3 to 20 MHz crystal oscillator is stabilized.

- **LOCKA: PLLA Lock Status**

0: PLLA is not locked

1: PLLA is locked.

- **MCKRDY: Master Clock Status**

0: Master Clock is not ready.

1: Master Clock is ready.

- **OSCSELS: Slow Clock Oscillator Selection**

0: Embedded 32 kHz RC oscillator is selected.

1: 32768 Hz crystal oscillator is selected.

- **PCKRDYx: Programmable Clock Ready Status**

0: Programmable Clock x is not ready.

1: Programmable Clock x is ready.

- **MOSCSELS: Main Clock Source Oscillator Selection Status**

0: Selection is in progress.

1: Selection is done.

- **MOSCRCS: 4/8/12 MHz RC Oscillator Status**

0: 4/8/12 MHz RC oscillator is not stabilized.

1: 4/8/12 MHz RC oscillator is stabilized.

- **CFDEV: Clock Failure Detector Event**

0: No clock failure detection of the 3 to 20 MHz crystal oscillator has occurred since the last read of PMC_SR.

1: At least one clock failure detection of the 3 to 20 MHz crystal oscillator has occurred since the last read of PMC_SR.

- **CFDS: Clock Failure Detector Status**

0: A clock failure of the 3 to 20 MHz crystal oscillator is not detected.

1: A clock failure of the 3 to 20 MHz crystal oscillator 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.

- **XT32KERR: 32768 Hz Crystal Oscillator Error**

0: The frequency of the 32768 Hz crystal oscillator is correct (32768 Hz +/- 1%) or the monitoring is disabled.

1: The frequency of the 32768 Hz crystal oscillator is incorrect or has been incorrect for an elapsed period of time since the monitoring has been enabled.

29.18.16 PMC Interrupt Mask Register

Name: PMC_IMR

Address: 0x400E046C

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	XT32KERR	–	–	CFDEV	MOSCRCS	MOSCSELS

15	14	13	12	11	10	9	8
–	–	–	–	–	PCKRDY2	PCKRDY1	PCKRDY0

7	6	5	4	3	2	1	0
–	–	–	–	MCKRDY	–	LOCKA	MOSCXTS

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **MOSCXTS: 3 to 20 MHz Crystal Oscillator Status Interrupt Mask**
- **LOCKA: PLLA Lock Interrupt Mask**
- **MCKRDY: Master Clock Ready Interrupt Mask**
- **PCKRDYx: Programmable Clock Ready x Interrupt Mask**
- **MOSCSELS: Main Clock Source Oscillator Selection Status Interrupt Mask**
- **MOSCRCS: 4/8/12 MHz RC Oscillator Status Interrupt Mask**
- **CFDEV: Clock Failure Detector Event Interrupt Mask**
- **XT32KERR: 32768 Hz Oscillator Error Interrupt Mask**

29.18.17 PMC Fast Startup Mode Register

Name: PMC_FSMR

Address: 0x400E0470

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	FLPM		LPM	–	USBAL	RTCAL	RTTAL

15	14	13	12	11	10	9	8
FSTT15	FSTT14	FSTT13	FSTT12	FSTT11	FSTT10	FSTT9	FSTT8

7	6	5	4	3	2	1	0
FSTT7	FSTT6	FSTT5	FSTT4	FSTT3	FSTT2	FSTT1	FSTT0

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **FSTT0–FSTT15: Fast Startup Input Enable 0 to 15**

0: The corresponding wake-up input has no effect on the PMC.

1: The corresponding wake-up input enables a fast restart signal to the PMC.

- **RTTAL: RTT Alarm Enable**

0: The RTT alarm has no effect on the PMC.

1: The RTT alarm enables a fast restart signal to the PMC.

- **RTCAL: RTC Alarm Enable**

0: The RTC alarm has no effect on the PMC.

1: The RTC alarm enables a fast restart signal to the PMC.

- **USBAL: USB Alarm Enable**

0: The USB alarm has no effect on the PMC.

1: The USB alarm enables a fast restart signal to the PMC.

- **LPM: Low-power Mode**

0: The WaitForInterrupt (WFI) or the WaitForEvent (WFE) instruction of the processor makes the processor enter Sleep mode.

1: The WaitForEvent (WFE) instruction of the processor makes the system to enter Wait mode.

- **FLPM: Flash Low-power Mode**

Value	Name	Description
0	FLASH_STANDBY	Flash is in Standby Mode when system enters Wait Mode
1	FLASH_DEEP_POWERDOWN	Flash is in Deep-power-down mode when system enters Wait Mode
2	FLASH_IDLE	Idle mode

29.18.18 PMC Fast Startup Polarity Register

Name: PMC_FSPR

Address: 0x400E0474

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
FSTP15	FSTP14	FSTP13	FSTP12	FSTP11	FSTP10	FSTP9	FSTP8
7	6	5	4	3	2	1	0
FSTP7	FSTP6	FSTP5	FSTP4	FSTP3	FSTP2	FSTP1	FSTP0

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection 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.

29.18.19 PMC Fault Output Clear Register

Name: PMC_FOCR

Address: 0x400E0478

Access: Write-only

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

- **FOCLR: Fault Output Clear**

Clears the clock failure detector fault output.

29.18.20 PMC Write Protection Mode Register

Name: PMC_WPMR

Address: 0x400E04E4

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 0x504D43 (“PMC” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x504D43 (“PMC” in ASCII).

See Section 29.17 ”Register Write Protection” for the list of registers that can be write-protected.

- **WPKEY: Write Protection Key**

Value	Name	Description
0x504D43	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

29.18.21 PMC Write Protection Status Register

Name: PMC_WPSR

Address: 0x400E04E8

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
WPVSRC							

15	14	13	12	11	10	9	8
WPVSRC							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WPVS

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of the PMC_WPSR.

1: A write protection violation has occurred since the last read of the PMC_WPSR. 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**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

29.18.22 PMC Peripheral Clock Enable Register 1

Name: PMC_PCER1

Address: 0x400E0500

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
PID47	PID46	PID45	PID44	PID43	PID42	PID41	PID40

7	6	5	4	3	2	1	0
PID39	PID38	PID37	PID36	PID35	PID34	PID33	PID32

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **PIDx: Peripheral Clock x Enable**

0: No effect.

1: Enables the corresponding peripheral clock.

Notes:
1. The values for PIDx are defined in the section “Peripheral Identifiers”.
2. Programming the control bits of the Peripheral ID that are not implemented has no effect on the behavior of the PMC.

29.18.23 PMC Peripheral Clock Disable Register 1

Name: PMC_PCDR1

Address: 0x400E0504

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
PID47	PID46	PID45	PID44	PID43	PID42	PID41	PID40

7	6	5	4	3	2	1	0
PID39	PID38	PID37	PID36	PID35	PID34	PID33	PID32

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **PIDx: Peripheral Clock x Disable**

0: No effect.

1: Disables the corresponding peripheral clock.

Note: The values for PIDx are defined in the section “Peripheral Identifiers”.

29.18.24 PMC Peripheral Clock Status Register 1

Name: PMC_PCSR1

Address: 0x400E0508

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
PID47	PID46	PID45	PID44	PID43	PID42	PID41	PID40

7	6	5	4	3	2	1	0
PID39	PID38	PID37	PID36	PID35	PID34	PID33	PID32

- **PIDx: Peripheral Clock x Status**

0: The corresponding peripheral clock is disabled.

1: The corresponding peripheral clock is enabled.

Note: The values for PIDx are defined in the section “Peripheral Identifiers”.

29.18.25 PMC Oscillator Calibration Register

Name: PMC_OCR

Address: 0x400E0510

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
SEL12	CAL12						

15	14	13	12	11	10	9	8
SEL8	CAL8						

7	6	5	4	3	2	1	0
SEL4	CAL4						

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **CAL4: RC Oscillator Calibration bits for 4 MHz**

Calibration bits applied to the RC Oscillator when SEL4 is set.

- **SEL4: Selection of RC Oscillator Calibration bits for 4 MHz**

0: Default value stored in Flash memory.

1: Value written by user in CAL4 field of this register.

- **CAL8: RC Oscillator Calibration bits for 8 MHz**

Calibration bits applied to the RC Oscillator when SEL8 is set.

- **SEL8: Selection of RC Oscillator Calibration bits for 8 MHz**

0: Factory-determined value stored in Flash memory.

1: Value written by user in CAL8 field of this register.

- **CAL12: RC Oscillator Calibration bits for 12 MHz**

Calibration bits applied to the RC Oscillator when SEL12 is set.

- **SEL12: Selection of RC Oscillator Calibration bits for 12 MHz**

0: Factory-determined value stored in Flash memory.

1: Value written by user in CAL12 field of this register.

29.18.26 PLL Maximum Multiplier Value Register

Name: PMC_PMMR

Address: 0x400E0530

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
–	–	–	–	–	PLLA_MMAX		
7	6	5	4	3	2	1	0
PLLA_MMAX							

This register can only be written if the WPEN bit is cleared in the "PMC Write Protection Mode Register" .

- **PLLA_MMAX: PLLA Maximum Allowed Multiplier Value**

Defines the maximum value of multiplication factor that can be sent to PLLA. Any value of the MULA field (see "PMC Clock Generator PLLA Register") above PLLA_MMAX is saturated to PLLA_MMAX. PLLA_MMAX write operation is cancelled in the following cases:

- The value of MULA is currently saturated by PLLA_MMAX
- The user is trying to write a value of PLLA_MMAX that is smaller than the current value of MULA

30. Advanced Encryption Standard (AES)

30.1 Description

The Advanced Encryption Standard (AES) is compliant with the American *FIPS (Federal Information Processing Standard) Publication 197* specification.

The AES supports all five confidentiality modes of operation for symmetrical key block cipher algorithms (ECB, CBC, OFB, CFB and CTR), as specified in the *NIST Special Publication 800-38A Recommendation*. It is compatible with all these modes via DMA Controller channels, minimizing processor intervention for large buffer transfers.

The 128-bit/192-bit/256-bit key is stored in four/six/eight 32-bit write-only AES Key Word registers (AES_KEYWR0–3).

The 128-bit input data and initialization vector (for some modes) are each stored in four 32-bit write-only AES Input Data registers (AES_IDATAR0–3) and AES Initialization Vector registers (AES_IVR0–3).

As soon as the initialization vector, the input data and the key are configured, the encryption/decryption process may be started. Then the encrypted/decrypted data are ready to be read out on the four 32-bit AES Output Data registers (AES_ODATAR0–3) or through the DMA channels.

30.2 Embedded Characteristics

- Compliant with *FIPS Publication 197, Advanced Encryption Standard (AES)*
- 128-bit/192-bit/256-bit Cryptographic Key
- 12/14/16 Clock Cycles Encryption/Decryption Processing Time with a 128-bit/192-bit/256-bit Cryptographic Key
- Double Input Buffer Optimizes Runtime
- Support of the Modes of Operation Specified in the *NIST Special Publication 800-38A*:
  - Electronic Code Book (ECB)
  - Cipher Block Chaining (CBC) including CBC-MAC
  - Cipher Feedback (CFB)
  - Output Feedback (OFB)
  - Counter (CTR)
- 8, 16, 32, 64 and 128-bit Data Sizes Possible in CFB Mode
- Last Output Data Mode Allows Optimized Message Authentication Code (MAC) Generation
- Connection to DMA Optimizes Data Transfers for all Operating Modes

30.3 Product Dependencies

30.3.1 Power Management

The AES may be clocked through the Power Management Controller (PMC), so the programmer must first to configure the PMC to enable the AES clock.

30.3.2 Interrupt Sources

The AES interface has an interrupt line connected to the Interrupt Controller.

Handling the AES interrupt requires programming the Interrupt Controller before configuring the AES.

Table 30-1. Peripheral IDs

Instance	ID
AES	39

30.4 Functional Description

The Advanced Encryption Standard (AES) specifies a FIPS-approved cryptographic algorithm that can be used to protect electronic data. The AES algorithm is a symmetric block cipher that can encrypt (encipher) and decrypt (decipher) information.

Encryption converts data to an unintelligible form called ciphertext. Decrypting the ciphertext converts the data back into its original form, called plaintext. The CIPHER bit in the AES Mode register (AES_MR) allows selection between the encryption and the decryption processes.

The AES is capable of using cryptographic keys of 128/192/256 bits to encrypt and decrypt data in blocks of 128 bits. This 128-bit/192-bit/256-bit key is defined in the AES_KEYWRx.

The input to the encryption processes of the CBC, CFB, and OFB modes includes, in addition to the plaintext, a 128-bit data block called the initialization vector (IV), which must be set in the AES_IVRx. The initialization vector is used in an initial step in the encryption of a message and in the corresponding decryption of the message. The AES_IVRx are also used by the CTR mode to set the counter value.

30.4.1 AES Register Endianness

In ARM processor-based products, the system bus and processors manipulate data in little-endian form. The AES interface requires little-endian format words. However, in accordance with the protocol of the FIPS 197 specification, data is collected, processed and stored by the AES algorithm in big-endian form.

The following example illustrates how to configure the AES:

If the first 64 bits of a message (according to FIPS 197, i.e., big-endian format) to be processed is 0xcafedeca_01234567, then the AES_IDATAR0 and AES_IDATAR1 registers must be written with the following pattern:

- AES_IDATAR0 = 0xcadefeca
- AES_IDATAR1 = 0x67452301

30.4.2 Operation Modes

The AES supports the following modes of operation:

- ECB: Electronic Code Book
- CBC: Cipher Block Chaining
- OFB: Output Feedback
- CFB: Cipher Feedback
  - CFB8 (CFB where the length of the data segment is 8 bits)
  - CFB16 (CFB where the length of the data segment is 16 bits)
  - CFB32 (CFB where the length of the data segment is 32 bits)
  - CFB64 (CFB where the length of the data segment is 64 bits)
  - CFB128 (CFB where the length of the data segment is 128 bits)
- CTR: Counter

The data pre-processing, post-processing and data chaining for the concerned modes are automatically performed. Refer to the *NIST Special Publication 800-38A* for more complete information.

These modes are selected by setting the OPMOD field in the AES_MR.

In CFB mode, five data sizes are possible (8, 16, 32, 64 or 128 bits), configurable by means of the CFBS field in the AES_MR (Section 30.5.2 “AES Mode Register”).

In CTR mode, the size of the block counter embedded in the module is 16 bits. Therefore, there is a rollover after processing 1 megabyte of data. If the file to be processed is greater than 1 megabyte, this file must be split into fragments of 1 megabyte or less for the first fragment if the initial value of the counter is greater than 0. Prior to loading the first fragment into AES_IDATARx, AES_IVRx must be fully programmed with the initial counter value. For any fragment, after the transfer is completed and prior to transferring the next fragment, AES_IVRx must be programmed with the appropriate counter value.

If the initial value of the counter is greater than 0 and the data buffer size to be processed is greater than 1 Mbyte, the size of the first fragment to be processed must be 1 Mbyte minus 16 × (initial value) to prevent a rollover of the internal 1-bit counter.

To have a sequential increment, the counter value must be programmed with the value programmed for the previous fragment + 2^{16} (or less for the first fragment).

All AES_IVRx fields must be programmed to take into account the possible carry propagation.

30.4.3 Double Input Buffer

The AES_IDATARx can be double-buffered to reduce the runtime of large files.

This mode allows writing a new message block when the previous message block is being processed. This is only possible when DMA accesses are performed (SMOD = 0x2).

The DUALBUFF bit in the AES_MR must be set to ‘1’ to access the double buffer.

30.4.4 Start Modes

The SMOD field in the AES_MR allows selection of the encryption (or decryption) Start mode.

30.4.4.1 Manual Mode

The sequence order is as follows:

1. Write the AES_MR with all required fields, including but not limited to SMOD and OPMOD.
2. Write the 128-bit/192-bit/256-bit key in the AES_KEYWRx.
3. Write the initialization vector (or counter) in the AES_IVRx.

Note: The AES_IVRx concern all modes except ECB.

4. Set the bit DATRDY (Data Ready) in the AES Interrupt Enable register (AES_IER), depending on whether an interrupt is required or not at the end of processing.
5. Write the data to be encrypted/decrypted in the authorized AES_IDATARx (see Table 30-2).
6. Set the START bit in the AES Control register (AES_CR) to begin the encryption or the decryption process.
7. When processing completes, the DATRDY flag in the AES Interrupt Status register (AES_ISR) is raised. If an interrupt has been enabled by setting the DATRDY bit in the AES_IER, the interrupt line of the AES is activated.
8. When software reads one of the AES_ODATARx, the DATRDY bit is automatically cleared.

Table 30-2. Authorized Input Data Registers

Operation Mode	Input Data Registers to Write
ECB	All
CBC	All
OFB	All
128-bit CFB	All

Table 30-2. Authorized Input Data Registers

Operation Mode	Input Data Registers to Write
64-bit CFB	AES_IDATAR0 and AES_IDATAR1
32-bit CFB	AES_IDATAR0
16-bit CFB	AES_IDATAR0
8-bit CFB	AES_IDATAR0
CTR	All

Note: In 64-bit CFB mode, writing to AES_IDATAR2 and AES_IDATAR3 is not allowed and may lead to errors in processing.

Note: In 32, 16, and 8-bit CFB modes, writing to AES_IDATAR1, AES_IDATAR2 and AES_IDATAR3 is not allowed and may lead to errors in processing.

30.4.4.2 Auto Mode

The Auto Mode is similar to the manual one, except that in this mode, as soon as the correct number of AES_IDATARx is written, processing is automatically started without any action in the AES_CR.

30.4.4.3 DMA Mode

The DMA Controller can be used in association with the AES to perform an encryption/decryption of a buffer without any action by software during processing.

The SMOD field in the AES_MR must be configured to 0x2 and the DMA must be configured with non-incremental addresses.

The start address of any transfer descriptor must be configured with the address of AES_IDATAR0.

The DMA chunk size configuration depends on the AES mode of operation and is listed in Table 30-3 “DMA Data Transfer Type for the Different Operation Modes”.

When writing data to AES with a first DMA channel, data are first fetched from a memory buffer (source data). It is recommended to configure the size of source data to “words” even for CFB modes. On the contrary, the destination data size depends on the mode of operation. When reading data from the AES with the second DMA channel, the source data is the data read from AES and data destination is the memory buffer. In this case, the source data size depends on the AES mode of operation and is listed in Table 30-3.

Table 30-3. DMA Data Transfer Type for the Different Operation Modes

Operation Mode	Chunk Size	Destination/Source Data Transfer Type
ECB	4	Word
CBC	4	Word
OFB	4	Word
CFB 128-bit	4	Word
CFB 64-bit	1	Word
CFB 32-bit	1	Word
CFB 16-bit	1	Half-word
CFB 8-bit	1	Byte
CTR	4	Word

30.4.5 Last Output Data Mode

This mode is used to generate cryptographic checksums on data (MAC) by means of cipher block chaining encryption algorithm (CBC-MAC algorithm for example).

After each end of encryption/decryption, the output data are available either on the AES_ODATARx for Manual and Auto mode or at the address specified in the receive buffer pointer for DMA mode (see Table 30-4 “Last Output Data Mode Behavior versus Start Modes”).

The Last Output Data (LOD) bit in the AES_MR allows retrieval of only the last data of several encryption/decryption processes.

Therefore, there is no need to define a read buffer in DMA mode.

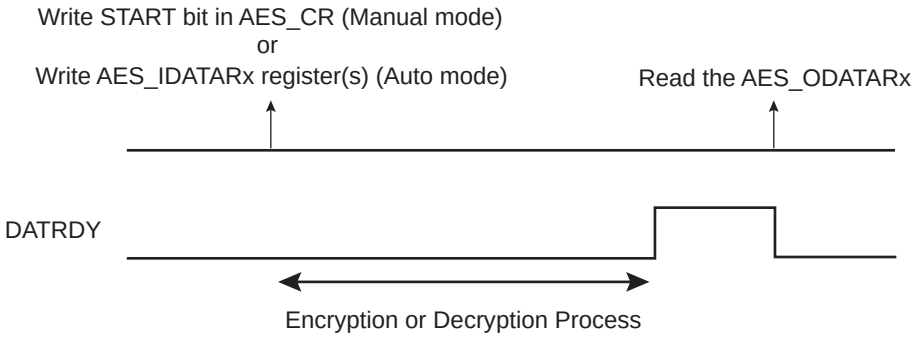
This data are only available on the AES_ODATARx.

30.4.5.1 Manual and Auto Modes

If AES_MR.LOD = 0

The DATRDY flag is cleared when at least one of the AES_ODATARx is read (see Figure 30-1).

Figure 30-1. Manual and Auto Modes with AES_MR.LOD = 0



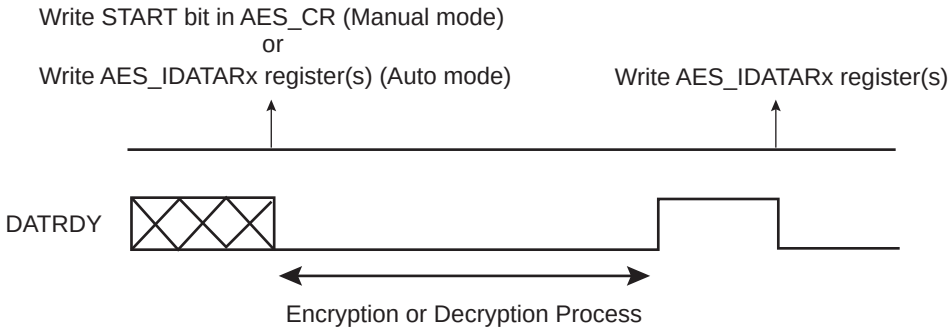
If the user does not want to read the AES_ODATARx between each encryption/decryption, the DATRDY flag will not be cleared. If the DATRDY flag is not cleared, the user cannot know the end of the following encryptions/decryptions.

If AES_MR.LOD = 1

This mode is optimized to process AES CPC-MAC operating mode.

The DATRDY flag is cleared when at least one AES_IDATAR is written (see Figure 30-2). No more AES_ODATAR reads are necessary between consecutive encryptions/decryptions.

Figure 30-2. Manual and Auto Modes with AES_MR.LOD = 1



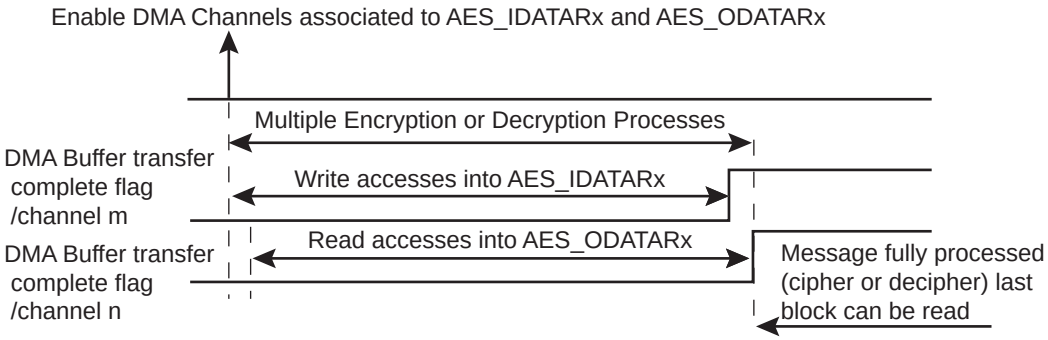
30.4.5.2 DMA Mode

If AES_MR.LOD = 0

This mode may be used for all AES operating modes except CBC-MAC where AES_MR.LOD = 1 mode is recommended.

The end of the encryption/decryption is indicated by the end of DMA transfer associated to AES_ODATARx (see Figure 30-3). Two DMA channels are required: one for writing message blocks to AES_IDATARx and one to obtain the result from AES_ODATARx.

Figure 30-3. DMA Transfer with AES_MR.LOD = 0



If AES_MR.LOD = 1

This mode is optimized to process AES CBC-MAC operating mode.

The user must first wait for the DMA buffer transfer complete flag, then for the flag DATRDY to rise to ensure that the encryption/decryption is completed (see Figure 30-4).

In this case, no receive buffers are required.

The output data are only available on the AES_ODATARx.

Figure 30-4. DMA Transfer with AES_MR.LOD = 1

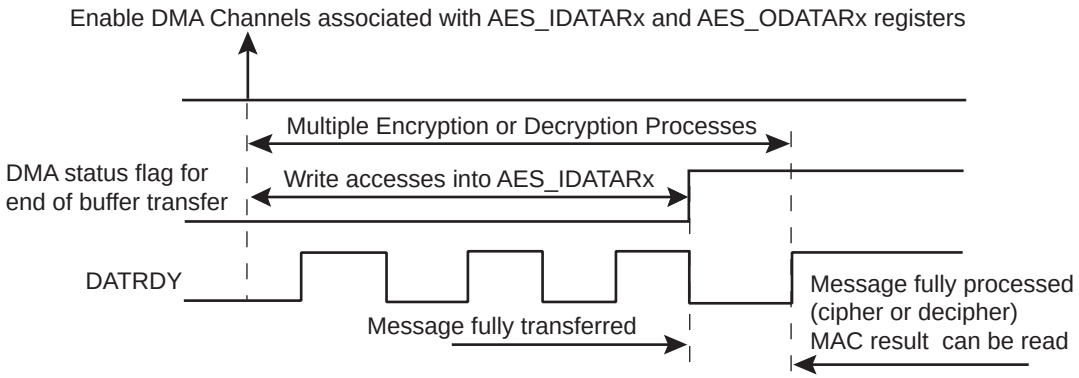


Table 30-4 summarizes the different cases.

Table 30-4. Last Output Data Mode Behavior versus Start Modes

Sequence	Manual and Auto Modes		DMA Transfer	
	AES_MR.LOD = 0	AES_MR.LOD = 1	AES_MR.LOD = 0	AES_MR.LOD = 1
DATRDY Flag Clearing Condition(1)	At least one AES_ODATAR must be read	At least one AES_IDATAR must be written	Not used	Managed by the DMA
End of Encryption/Decryption Notification	DATRDY	DATRDY	2 DMA Buffer transfer complete flags (channel m and channel n)	DMA buffer transfer complete flag, then AES DATRDY flag
Encrypted/Decrypted Data Result Location	In the AES_ODATARx	In the AES_ODATARx	At the address specified in the Channel Buffer Transfer Descriptor	In the AES_ODATARx

Note: 1. Depending on the mode, there are other ways of clearing the DATRDY flag. See Section 30.5.6 “AES Interrupt Status Register”.

Warning: In DMA mode, reading the AES_ODATARx before the last data transfer may lead to unpredictable results.

30.4.6 Security Features

30.4.6.1 Unspecified Register Access Detection

When an unspecified register access occurs, the URAD flag in the AES_ISR is raised. Its source is then reported in the Unspecified Register Access Type (URAT) field. Only the last unspecified register access is available through the URAT field.

Several kinds of unspecified register accesses can occur:

- Input Data register written during the data processing when SMOD = IDATAR0_START
- Output Data register read during data processing
- Mode register written during data processing
- Output Data register read during sub-keys generation
- Mode register written during sub-keys generation
- Write-only register read access

The URAD bit and the URAT field can only be reset by the SWRST bit in the AES_CR.

30.5 Advanced Encryption Standard (AES) User Interface

Table 30-5. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	AES_CR	Write-only	–
0x04	Mode Register	AES_MR	Read/Write	0x0
0x08–0x0C	Reserved	–	–	–
0x10	Interrupt Enable Register	AES_IER	Write-only	–
0x14	Interrupt Disable Register	AES_IDR	Write-only	–
0x18	Interrupt Mask Register	AES_IMR	Read-only	0x0
0x1C	Interrupt Status Register	AES_ISR	Read-only	0x0
0x20	Key Word Register 0	AES_KEYWR0	Write-only	–
0x24	Key Word Register 1	AES_KEYWR1	Write-only	–
0x28	Key Word Register 2	AES_KEYWR2	Write-only	–
0x2C	Key Word Register 3	AES_KEYWR3	Write-only	–
0x30	Key Word Register 4	AES_KEYWR4	Write-only	–
0x34	Key Word Register 5	AES_KEYWR5	Write-only	–
0x38	Key Word Register 6	AES_KEYWR6	Write-only	–
0x3C	Key Word Register 7	AES_KEYWR7	Write-only	–
0x40	Input Data Register 0	AES_IDATAR0	Write-only	–
0x44	Input Data Register 1	AES_IDATAR1	Write-only	–
0x48	Input Data Register 2	AES_IDATAR2	Write-only	–
0x4C	Input Data Register 3	AES_IDATAR3	Write-only	–
0x50	Output Data Register 0	AES_ODATAR0	Read-only	0x0
0x54	Output Data Register 1	AES_ODATAR1	Read-only	0x0
0x58	Output Data Register 2	AES_ODATAR2	Read-only	0x0
0x5C	Output Data Register 3	AES_ODATAR3	Read-only	0x0
0x60	Initialization Vector Register 0	AES_IVR0	Write-only	–
0x64	Initialization Vector Register 1	AES_IVR1	Write-only	–
0x68	Initialization Vector Register 2	AES_IVR2	Write-only	–
0x6C	Initialization Vector Register 3	AES_IVR3	Write-only	–
0x70–0xAC	Reserved	–	–	–
0xB0–0xFC	Reserved	–	–	–

30.5.1 AES Control Register

Name: AES_CR

Address: 0x40004000

Access: Write-only

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

- **START: Start Processing**

0: No effect.

1: Starts manual encryption/decryption process.

- **SWRST: Software Reset**

0: No effect.

1: Resets the AES. A software-triggered hardware reset of the AES interface is performed.

30.5.2 AES Mode Register

Name: AES_MR

Address: 0x40004004

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
CKEY				–	CFBS		
15	14	13	12	11	10	9	8
LOD	OPMOD			KEYSIZE		SMOD	
7	6	5	4	3	2	1	0
PROCDLY				DUALBUFF	–	–	CIPHER

- **CIPHER: Processing Mode**

0: Decrypts data.

1: Encrypts data.

- **DUALBUFF: Dual Input Buffer**

Value	Name	Description
0	INACTIVE	AES_IDATARx cannot be written during processing of previous block.
1	ACTIVE	AES_IDATARx can be written during processing of previous block when SMOD = 2. It speeds up the overall runtime of large files.

- **PROCDLY: Processing Delay**

Processing Time = N × (PROCDLY + 1)

where

N = 10 when KEYSIZE = 0

N = 12 when KEYSIZE = 1

N = 14 when KEYSIZE = 2

The processing time represents the number of clock cycles that the AES needs in order to perform one encryption/decryption.

Note: The best performance is achieved with PROCDLY equal to 0.

- **SMOD: Start Mode**

Value	Name	Description
0	MANUAL_START	Manual Mode
1	AUTO_START	Auto Mode
2	IDATAR0_START	AES_IDATAR0 access only Auto Mode (DMA)

Values which are not listed in the table must be considered as “reserved”.

If a DMA transfer is used, configure SMOD to 0x2. Refer to Section 30.4.4.3 “DMA Mode” for more details.

- **KEYSIZE: Key Size**

Value	Name	Description
0	AES128	AES Key Size is 128 bits
1	AES192	AES Key Size is 192 bits
2	AES256	AES Key Size is 256 bits

Values which are not listed in the table must be considered as “reserved”.

- **OPMOD: Operation Mode**

Value	Name	Description
0	ECB	ECB: Electronic Code Book mode
1	CBC	CBC: Cipher Block Chaining mode
2	OFB	OFB: Output Feedback mode
3	CFB	CFB: Cipher Feedback mode
4	CTR	CTR: Counter mode (16-bit internal counter)

Values which are not listed in the table must be considered as “reserved”.

For CBC-MAC operating mode, set OPMOD to CBC and LOD to 1.

- **LOD: Last Output Data Mode**

0: No effect.

After each end of encryption/decryption, the output data are available either on the output data registers (Manual and Auto modes) or at the address specified in the Channel Buffer Transfer Descriptor for DMA mode.

In Manual and Auto modes, the DATRDY flag is cleared when at least one of the Output Data registers is read.

1: The DATRDY flag is cleared when at least one of the Input Data Registers is written.

No more Output Data Register reads is necessary between consecutive encryptions/decryptions (see Section 30.4.5 “Last Output Data Mode”).

Warning: In DMA mode, reading to the Output Data registers before the last data encryption/decryption process may lead to unpredictable results.

- **CFBS: Cipher Feedback Data Size**

Value	Name	Description
0	SIZE_128BIT	128-bit
1	SIZE_64BIT	64-bit
2	SIZE_32BIT	32-bit
3	SIZE_16BIT	16-bit
4	SIZE_8BIT	8-bit

Values which are not listed in table must be considered as “reserved”.

- **CKEY: Key**

Value	Name	Description
0xE	PASSWD	This field must be written with 0xE the first time the AES_MR is programmed. For subsequent programming of the AES_MR, any value can be written, including that of 0xE. Always reads as 0.

30.5.3 AES Interrupt Enable Register

Name: AES_IER

Address: 0x40004010

Access: Write-only

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

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **DATRDY: Data Ready Interrupt Enable**
- **URAD: Unspecified Register Access Detection Interrupt Enable**

30.5.4 AES Interrupt Disable Register

Name: AES_IDR

Address: 0x40004014

Access: Write-only

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

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Disables the corresponding interrupt.

- **DATRDY: Data Ready Interrupt Disable**
- **URAD: Unspecified Register Access Detection Interrupt Disable**

30.5.5 AES Interrupt Mask Register

Name: AES_IMR

Address: 0x40004018

Access: Read-only

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

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

- **DATRDY: Data Ready Interrupt Mask**
- **URAD: Unspecified Register Access Detection Interrupt Mask**

30.5.6 AES Interrupt Status Register

Name: AES_ISR

Address: 0x4000401C

Access: Read-only

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

- **DATRDY: Data Ready (cleared by setting bit START or bit SWRST in AES_CR or by reading AES_ODATARx)**

0: Output data not valid.

1: Encryption or decryption process is completed.

Note: If AES_MR.LOD = 1: In Manual and Auto mode, the DATRDY flag can also be cleared by writing at least one AES_IDATARx.

- **URAD: Unspecified Register Access Detection Status (cleared by writing SWRST in AES_CR)**

0: No unspecified register access has been detected since the last SWRST.

1: At least one unspecified register access has been detected since the last SWRST.

- **URAT: Unspecified Register Access (cleared by writing SWRST in AES_CR)**

Value	Name	Description
0	IDR_WR_PROCESSING	Input Data register written during the data processing when SMOD = 0x2 mode.
1	ODR_RD_PROCESSING	Output Data register read during the data processing.
2	MR_WR_PROCESSING	Mode register written during the data processing.
3	ODR_RD_SUBKGEN	Output Data register read during the sub-keys generation.
4	MR_WR_SUBKGEN	Mode register written during the sub-keys generation.
5	WOR_RD_ACCESS	Write-only register read access.

Only the last Unspecified Register Access Type is available through the URAT field.

30.5.7 AES Key Word Register x

Name: AES_KEYWRx [x=0..7]

Address: 0x40004020

Access: Write-only

31	30	29	28	27	26	25	24
KEYW							

23	22	21	20	19	18	17	16
KEYW							

15	14	13	12	11	10	9	8
KEYW							

7	6	5	4	3	2	1	0
KEYW							

- **KEYW: Key Word**

The four/six/eight 32-bit Key Word registers set the 128-bit/192-bit/256-bit cryptographic key used for AES encryption/decryption.

AES_KEYWR0 corresponds to the first word of the key and respectively AES_KEYWR3/AES_KEYWR5/AES_KEYWR7 to the last one.

These registers are write-only to prevent the key from being read by another application.

30.5.8 AES Input Data Register x

Name: AES_IDATARx [x=0..3]

Address: 0x40004040

Access: Write-only

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

- **IDATA: Input Data Word**

The four 32-bit Input Data registers set the 128-bit data block used for encryption/decryption.

AES_IDATAR0 corresponds to the first word of the data to be encrypted/decrypted, and AES_IDATAR3 to the last one.

These registers are write-only to prevent the input data from being read by another application.

30.5.9 AES Output Data Register x

Name: AES_ODATARx [x=0..3]

Address: 0x40004050

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 four 32-bit Output Data registers contain the 128-bit data block that has been encrypted/decrypted.

AES_ODATAR0 corresponds to the first word, AES_ODATAR3 to the last one.

30.5.10 AES Initialization Vector Register x

Name: AES_IVRx [x=0..3]

Address: 0x40004060

Access: Write-only

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

- **IV: Initialization Vector**

The four 32-bit Initialization Vector registers set the 128-bit Initialization Vector data block that is used by some modes of operation as an additional initial input.

AES_IVR0 corresponds to the first word of the Initialization Vector, AES_IVR3 to the last one.

These registers are write-only to prevent the Initialization Vector from being read by another application.

For CBC, OFB and CFB modes, the IV input value corresponds to the initialization vector.

For CTR mode, the IV input value corresponds to the initial counter value.

Note: These registers are not used in ECB mode and must not be written.

31. Controller Area Network (CAN)

31.1 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/s.

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.

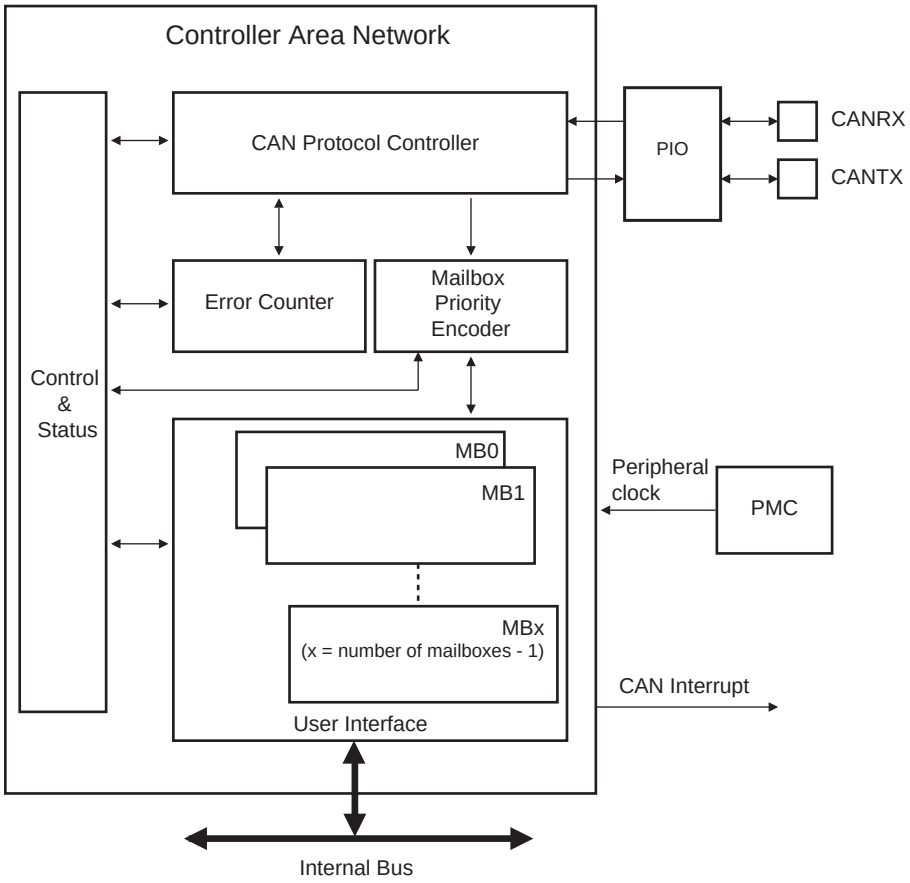
The CAN controller offers optimized features to support the Time Triggered Communication (TTC) protocol.

31.2 Embedded Characteristics

- Fully Compliant with CAN 2.0 Part A and 2.0 Part B
- Bit Rates up to 1 Mbit/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
- Register Write Protection

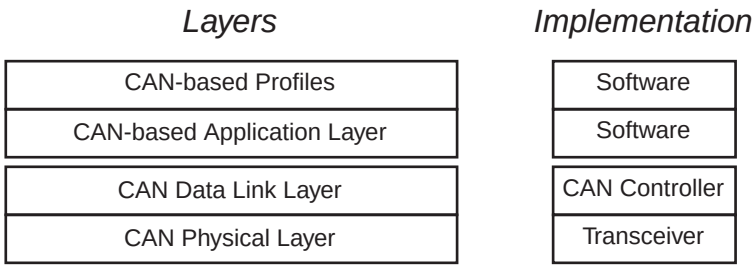
31.3 Block Diagram

Figure 31-1. CAN Block Diagram



31.4 Application Block Diagram

Figure 31-2. Application Block Diagram



31.5 I/O Lines Description

Table 31-1. I/O Lines Description

Name	Description	Type
CANRX	CAN Receive Serial Data	Input
CANTX	CAN Transmit Serial Data	Output

31.6 Product Dependencies

31.6.1 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 31-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
CAN0	CANRX0	PB3	A
CAN0	CANTX0	PB2	A
CAN1	CANRX1	PC12	C
CAN1	CANTX1	PC15	C

31.6.2 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.

31.6.3 Interrupt Sources

The CAN interrupt line is connected on one of the internal sources of the interrupt controller. Using the CAN interrupt requires the interrupt controller to be programmed first. Note that it is not recommended to use the CAN interrupt line in edge-sensitive mode.

Table 31-3. Peripheral IDs

Instance	ID
CAN0	37
CAN1	38

31.7 CAN Controller Features

31.7.1 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.

31.7.2 Mailbox Organization

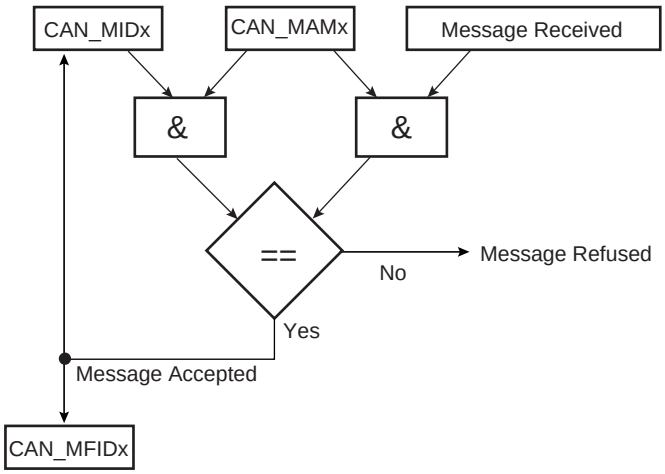
The CAN module has 8 buffers, also called channels or mailboxes. An identifier that corresponds to the CAN identifier is defined for each active mailbox. Message identifiers can match the standard frame identifier or the extended frame identifier. This identifier is defined for the first time during 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.

31.7.2.1 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 31-3. 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]();
```

31.7.2.2 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 31-4. Receive Mailbox Objects

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.

31.7.2.3 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).

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 31-5. Transmit Mailbox Objects

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 31.7.3). 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.

31.7.3 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). 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)
- 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. 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 is set. TOVF bit in the CAN_SR is cleared by reading the CAN_SR. Depending on the corresponding interrupt mask in the CAN_IMR, 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. 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 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, an interrupt is generated while TSTP is set in the CAN_SR. TSTP bit is cleared by reading the CAN_SR.

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. Time Triggered Mode is enabled by setting TTM field in the CAN_MR.

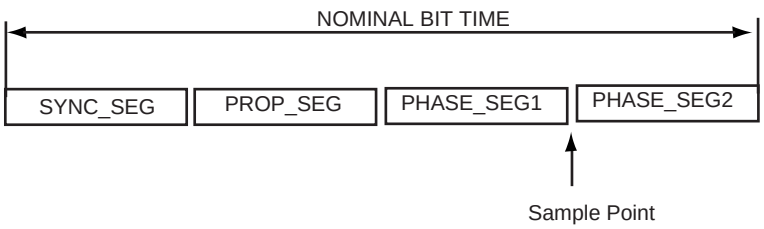
31.7.4 CAN 2.0 Standard Features

31.7.4.1 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 31-4. Partition of the CAN Bit Time



- 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 one 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".

- TIME QUANTUM

  The TIME QUANTUM (TQ) is a fixed unit of time derived from the peripheral clock period. The total number of TIME QUANTA in a bit time is programmable from 8 to 25.

- 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 two 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 four TQ.

If the SMP field in the CAN_BR 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:

$$t_{CSC} = (BRP + 1) / t_{peripheral\ clock}$$

Note: The BRP field must be within the range [1, 0x7F], i.e., BRP = 0 is not authorized.

$$t_{PRS} = t_{CSC} \times (PROPAG + 1)$$

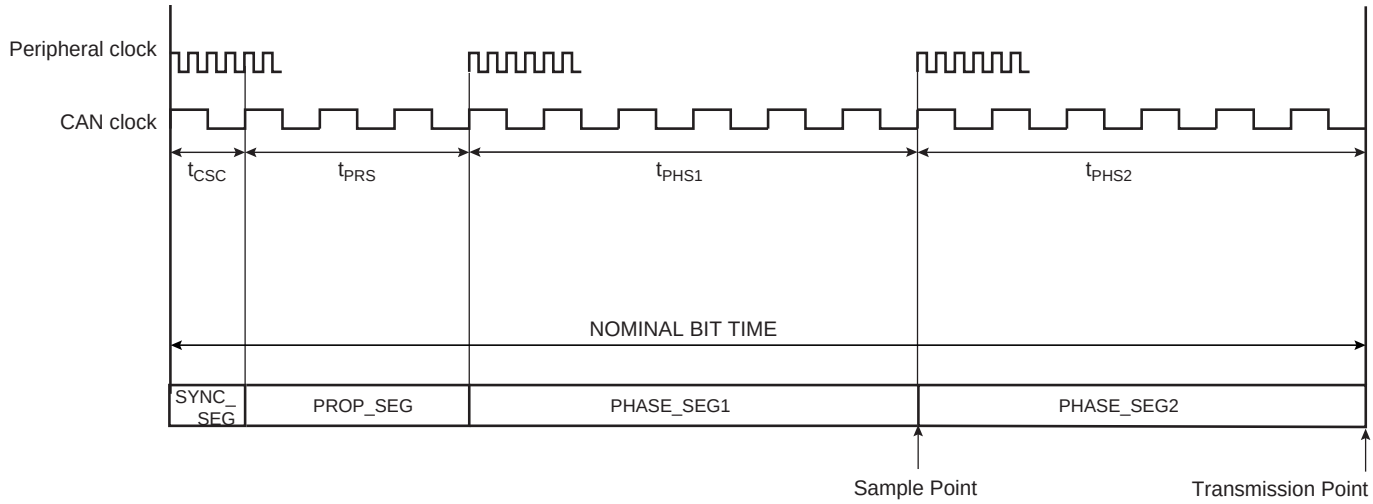
$$t_{PHS1} = t_{CSC} \times (PHASE1 + 1)$$

$$t_{PHS2} = t_{CSC} \times (PHASE2 + 1)$$

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 31-5. CAN Bit Timing



Example of bit timing determination for CAN baudrate of 500 kbit/s:

$f_{Peripheral\ clock}$ = 48 MHz

CAN baudrate = 500 kbit/s => bit time = 2 µs

Delay of the bus driver: 50 ns

Delay of the receiver: 30 ns

Delay of the bus line (20 m): 110 ns

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:

t_{CSC} = 1 time quanta = bit time / 16 = 125 ns

=> BRP = (t_{CSC} x $f_{peripheral\ clock}$) - 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:

t_{PRS} = 2 * (50+30+110) ns = 380 ns = 3 t_{CSC}

=> PROPAG = t_{PRS}/t_{CSC} - 1 = 2

The remaining time for the two phase segments is:

$t_{PHS}1 + t_{PHS}2$ = bit time - t_{CSC} - t_{PRS} = (16 - 1 - 3)t_{CSC}

$t_{PHS}1 + t_{PHS}2$ = 12 t_{CSC}

Because this number is even, we choose $t_{PHS}2 = t_{PHS}1$ (else we would choose $t_{PHS}2 = t_{PHS}1 + t_{CSC}$).

$t_{PHS}1 = t_{PHS}2$ = (12/2) t_{CSC} = 6 t_{CSC}

=> PHASE1 = PHASE2 = $t_{PHS}1/t_{CSC}$ - 1 = 5

The resynchronization jump width must comprise between one t_{CSC} and the minimum of four t_{CSC} and $t_{PHS}1$. We choose its maximum value:

t_{SJW} = Min(4 t_{CSC},$t_{PHS}1$) = 4 t_{CSC}

=> SJW = t_{SJW}/t_{CSC} - 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.

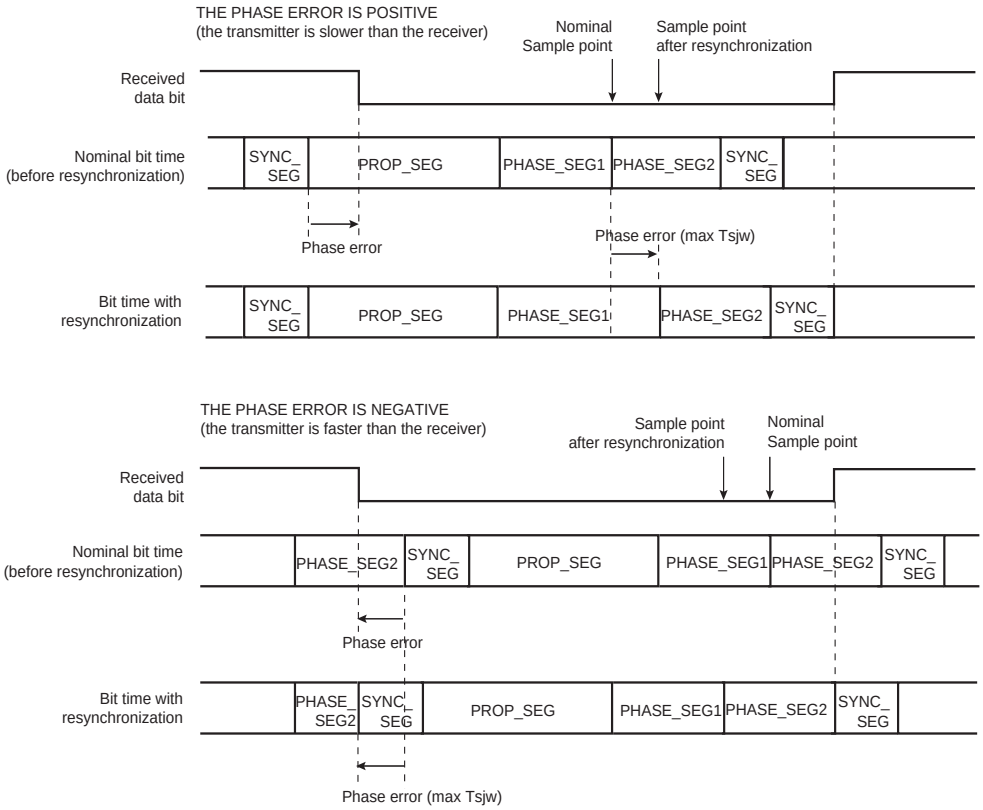
The effect of resynchronization is the same as that of hard synchronization when the magnitude of the phase error of the edge causing the resynchronization is less than or equal to the programmed value of the resynchronization jump width (t_{SJW}).

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 31-6. CAN Resynchronization



Autobaud Mode

The autobaud feature is enabled by setting the ABM field in the CAN_MR. 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.

31.7.4.2 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):

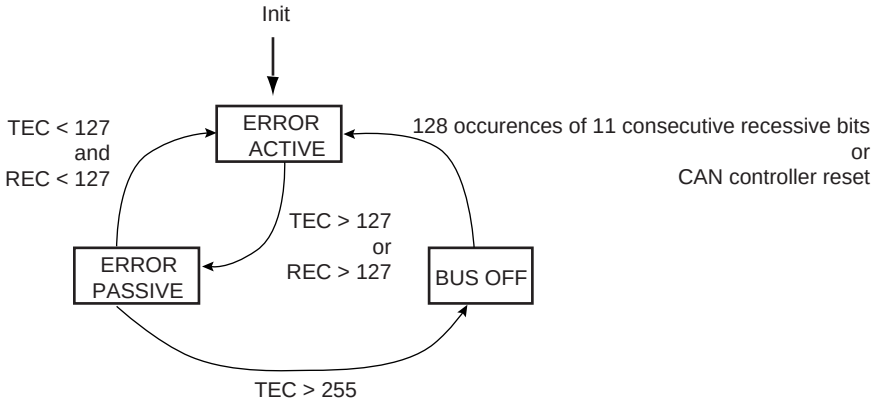
- CRC error (CERR bit in the CAN_SR): With the CRC, the transmitter calculates a checksum for the CRC bit sequence from the Start of Frame bit until the end of the Data Field. This CRC sequence is transmitted in the CRC field of the Data or Remote Frame.
- Bit-stuffing error (SERR bit in the CAN_SR): 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): 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): 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): 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 31-7. 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. The state of the CAN controller is automatically updated according to these counter values. If the CAN controller enters Error Active state, then the ERRA bit is set in the CAN_SR. The corresponding interrupt is pending while the interrupt is not masked in the CAN_IMR. If the CAN controller enters Error Passive Mode, then the ERRP bit is set in the CAN_SR and an interrupt remains pending while the ERRP bit is set in the CAN_IMR. If the CAN enters Bus Off Mode, then the BOFF bit is set in the CAN_SR. As for ERRP and ERRA, an interrupt is pending while the BOFF bit is set in the CAN_IMR.

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, but the node remains error active. The corresponding interrupt is pending while the interrupt is set in the CAN_IMR.

Refer to the Bosch CAN specification v2.0 for details on fault confinement.

Error Interrupt Handler

ERRA, WARN, ERRP and BOFF (CAN_SR) store the key transitions of the CAN bus status as defined in Figure 31-7 on page 654. The transitions depend on the TEC and REC (CAN_ECR) values as described in Section “Fault Confinement” on page 654.

These flags are latched to keep from triggering a spurious interrupt in case these bits are used as the source of an interrupt. Thus, these flags may not reflect the current status of the CAN bus.

The current CAN bus state can be determined by reading the TEC and REC fields of CAN_ECR.

31.7.4.3 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.

Reactive overload frames are automatically handled by the CAN controller even if the OVL bit in the CAN_MR is not set. An overload flag is generated in the same way as an error flag, but error counters do not increment.

31.7.5 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 is set; otherwise, the WAKEUP signal in the CAN_SR is set. These two bits 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 only after detection of 11 consecutive recessive bits on the bus.

31.7.5.1 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 is set. Depending on the corresponding mask in the CAN_IMR, an interrupt is generated while SLEEP is set.

The SLEEP signal in the CAN_SR 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. 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.

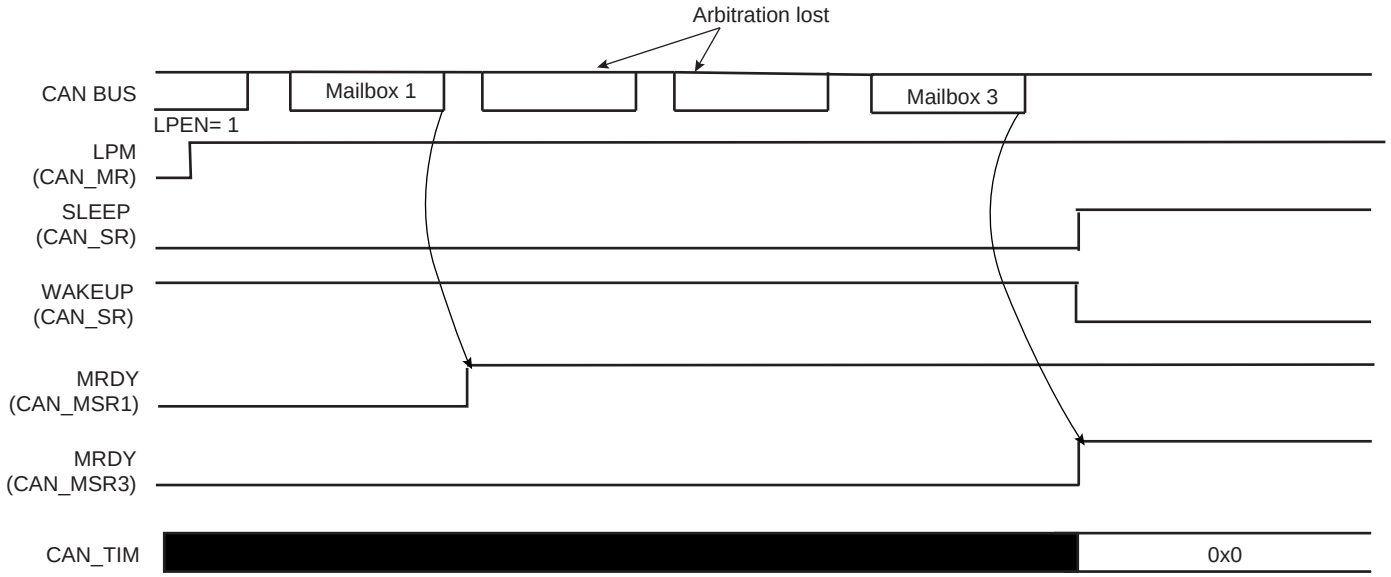
Error counters are disabled while the SLEEP signal is set to one.

Thus, to enter Low-power mode, the software application must:

- Set LPM field in the CAN_MR
- Wait for SLEEP signal rising

Now the CAN Controller clock can be disabled. This is done by programming the Power Management Controller (PMC).

Figure 31-8. Enabling Low-power Mode



31.7.5.2 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

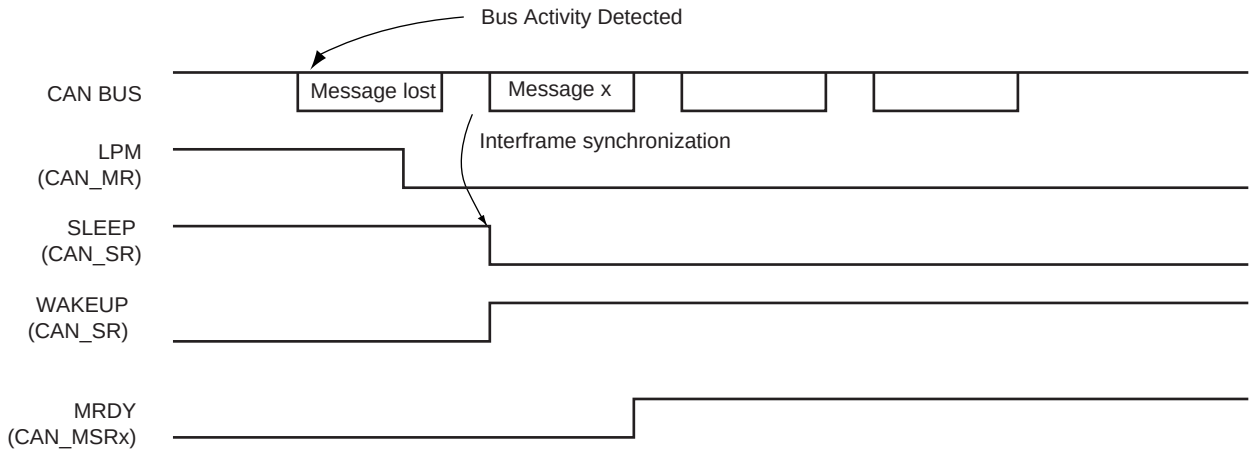
The CAN controller synchronizes itself with the bus activity by checking for eleven consecutive “recessive” bits. Once synchronized, the WAKEUP signal in the CAN_SR is set.

Depending on the corresponding mask in the CAN_IMR, an interrupt is generated while WAKEUP is set. The SLEEP signal in the CAN_SR 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 31-9).

Figure 31-9. Disabling Low-power Mode



31.8 Functional Description

31.8.1 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.

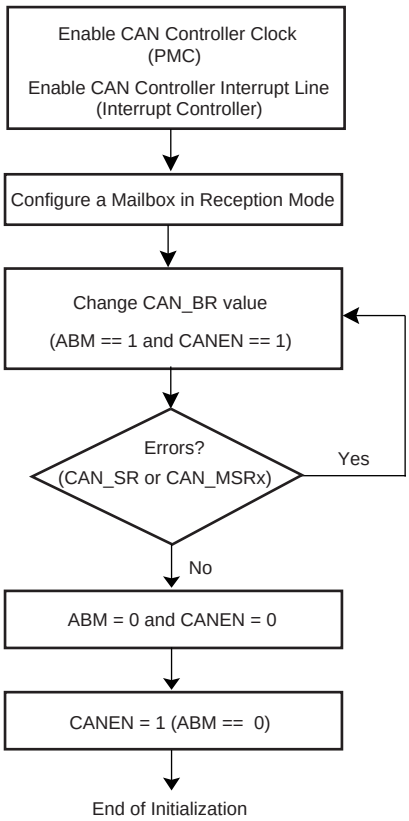
The CAN controller must be initialized with the CAN network parameters. The CAN_BR defines the sampling point in the bit time period. CAN_BR must be set before the CAN controller is enabled.

The CAN controller is enabled by setting the CANEN bit in the CAN_MR. At this stage, the internal CAN controller state machine is reset, error counters are reset to 0, and 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 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 values synchronized with the network. Once no error has been detected, the application disables the Autobaud Mode, clearing the ABM bit in the CAN_MR.

Figure 31-10. Possible Initialization Procedure



31.8.2 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. They can be unmasked by writing to the CAN_IER. After a power-up reset, all interrupt sources are disabled (masked). The current mask status can be checked by reading the CAN_IMR.

The CAN_SR 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.

31.8.3 CAN Controller Message Handling

31.8.3.1 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

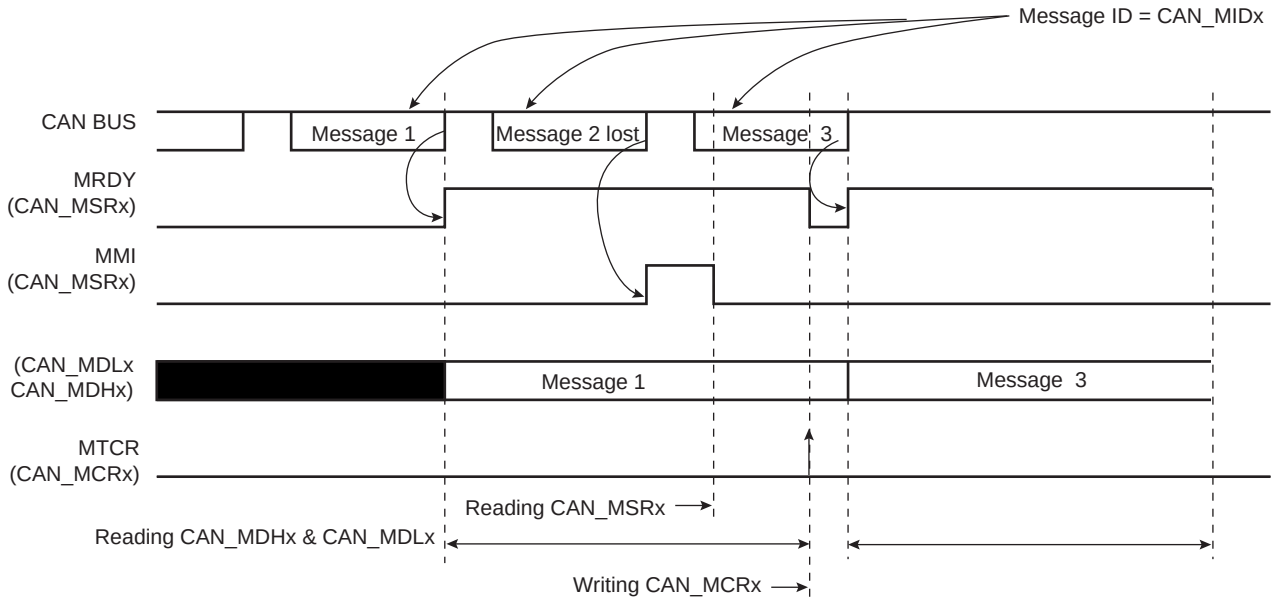
A mailbox is in Receive Mode once the MOT field in the CAN_MMRx has been configured. Message ID and Message Acceptance Mask must be set before the Receive Mode is enabled.

After Receive Mode is enabled, the MRDY flag in the CAN_MSR 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. This automatically clears the MRDY signal.

The MMI flag in the CAN_MSRx 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. 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. See Figure 31-11.

Figure 31-11. 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

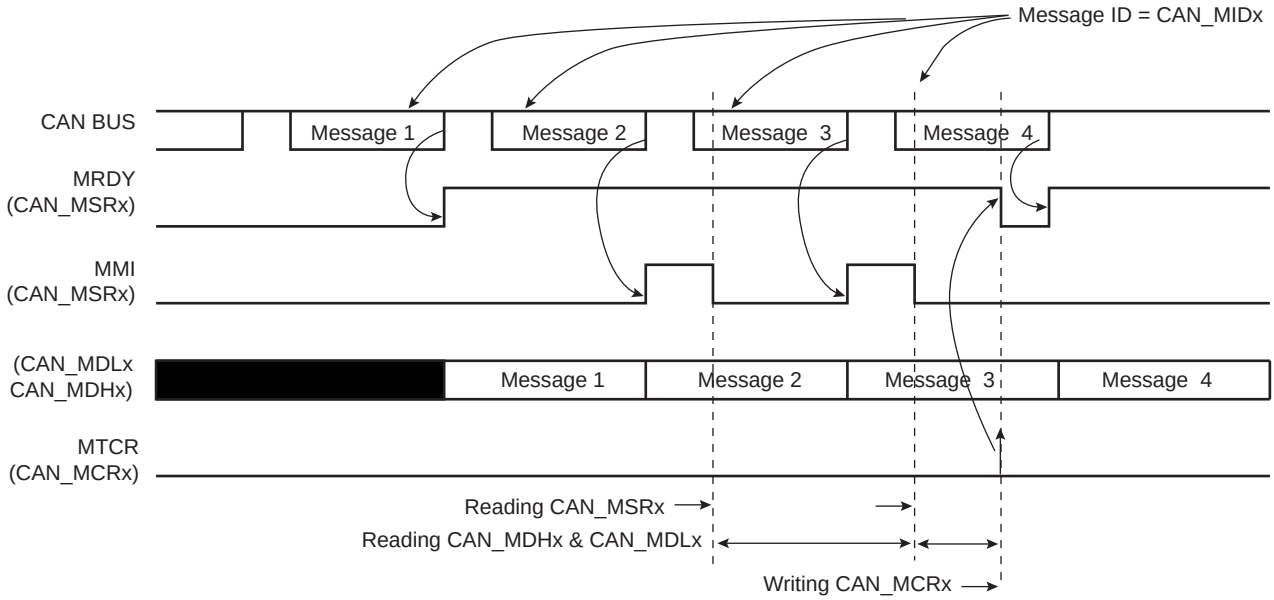
A mailbox is in Receive with Overwrite Mode once the MOT field in the CAN_MMRx has been configured. Message ID and Message Acceptance masks must be set before Receive Mode is enabled.

After Receive Mode is enabled, the MRDY flag in the CAN_MSR 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 notifies the software that a message has been dropped by the mailbox. This flag is cleared when reading the CAN_MSRx.

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 bit in the CAN_MSRx 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 31-12).

Figure 31-12. Receive with Overwrite Mailbox

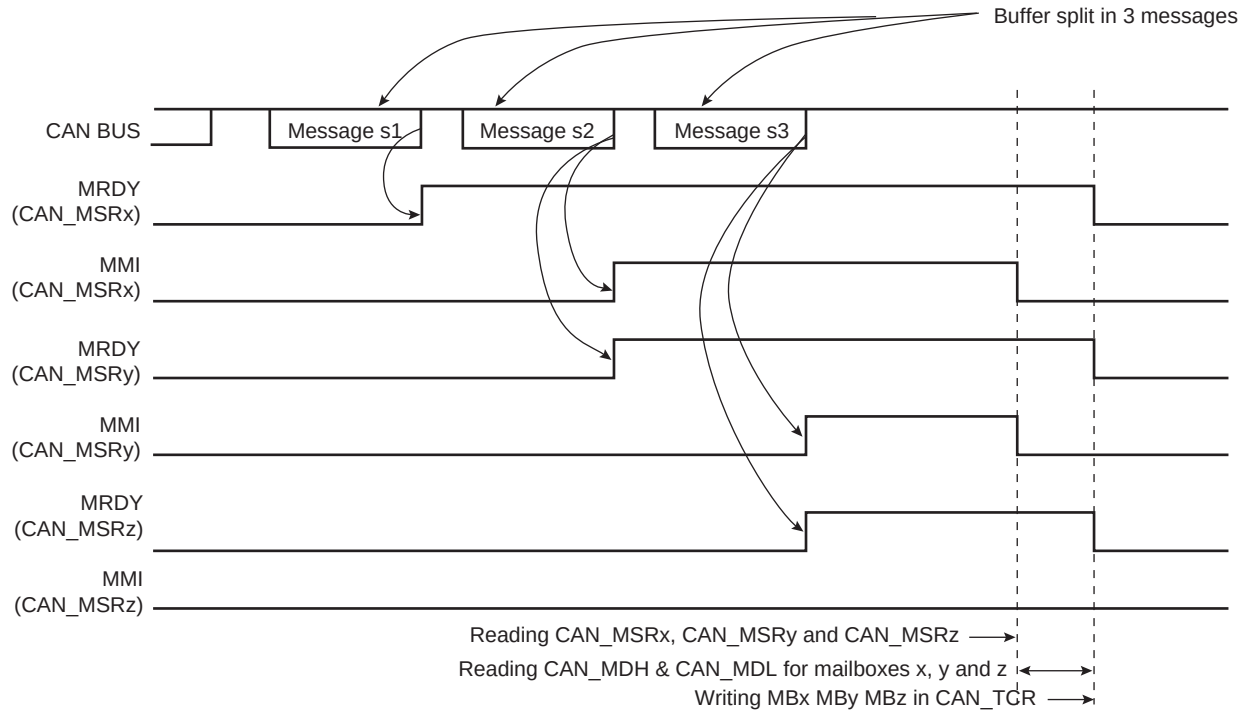


Chaining Mailboxes

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 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.

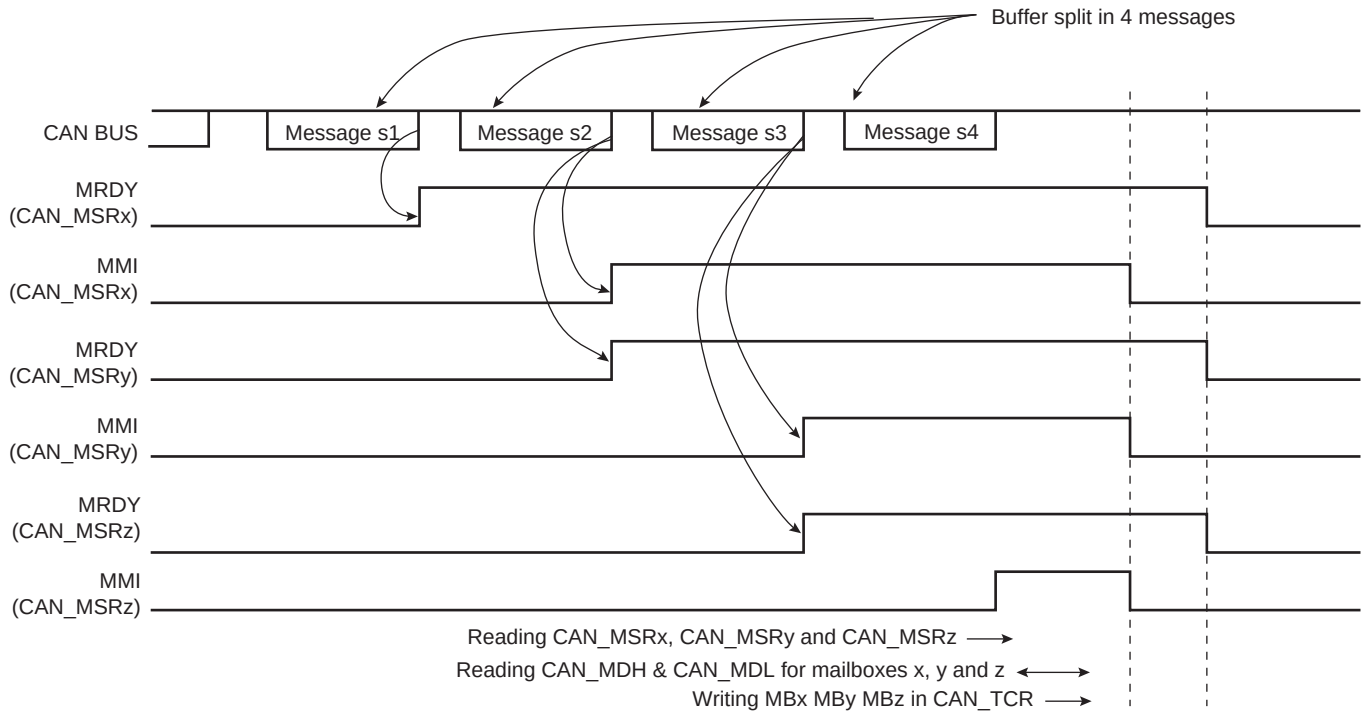
If several mailboxes are chained to receive a buffer split into several messages, all mailboxes except the last one (with the highest number) must be configured in Receive Mode. The first message received is handled by the first mailbox, the second one is refused by the first mailbox and accepted by the second mailbox, the last message is accepted by the last mailbox and refused by previous ones (see Figure 31-13).

Figure 31-13. 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 31-14).

Figure 31-14. Chaining Three Mailboxes to Receive a Buffer Split into Four Messages



31.8.3.2 Transmission Handling

A mailbox is in Transmit Mode once the MOT field in the CAN_MMRx 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 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.

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 664.

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. The priority is set in the PRIOR field of the CAN_MMRx. 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.

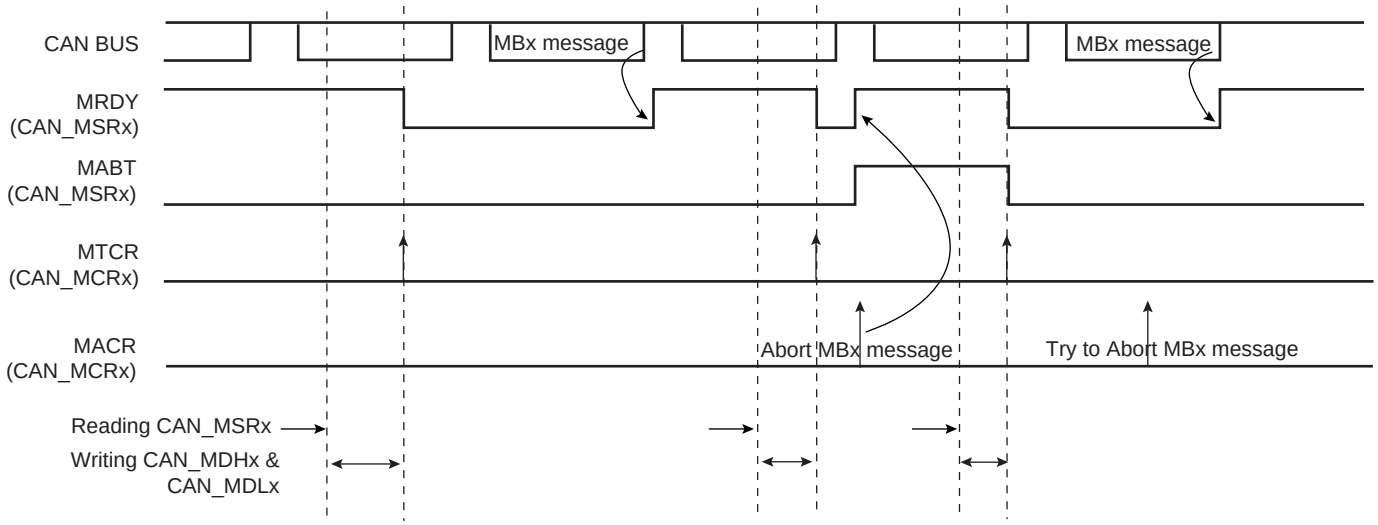
Setting the MACR bit in the CAN_MCRx aborts the transmission. Transmission for several mailboxes can be aborted by writing MBx fields in the CAN_ACR. If the message is being sent when the abort command is set, then the application is notified by the MRDY bit set and not the MABT in the CAN_MSRx. Otherwise, if the message has not been sent, then the MRDY and the MABT are set in the CAN_MSR.

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. 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 until the next transfer command.

Figure 31-15 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.

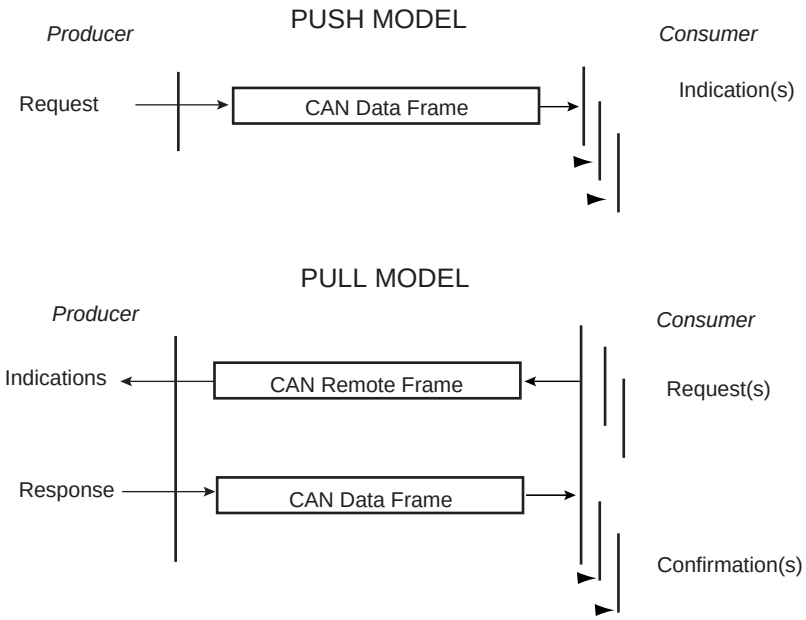
Figure 31-15. Transmitting Messages



31.8.3.3 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 31-16. 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 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 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. Data is sent after the reception of a remote frame as soon as it wins the bus arbitration.

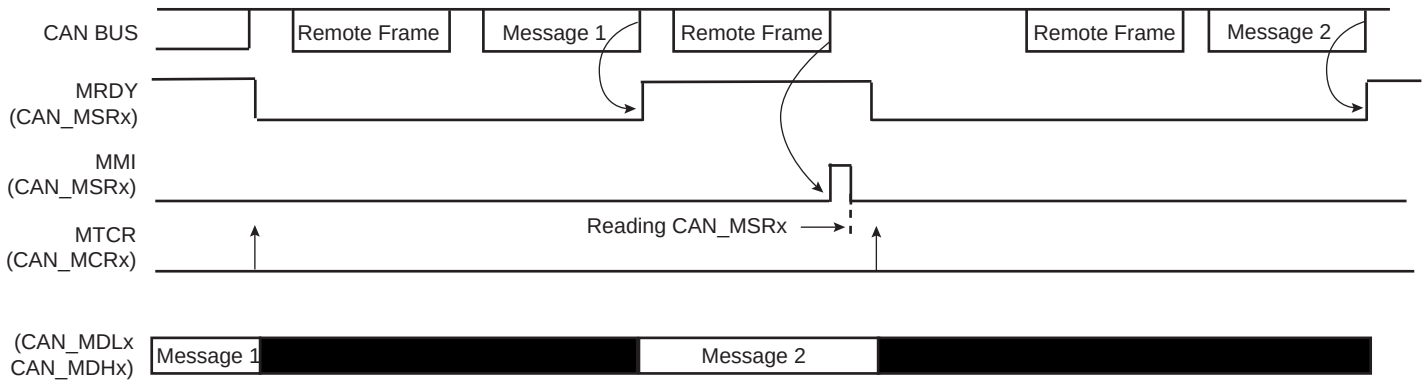
The MRDY flag remains at zero as long as the message has not been sent or aborted. No access to the mailbox data register can be done while MRDY flag is cleared. An interrupt is pending for the mailbox while the MRDY flag is set. This interrupt can be masked according to the mailbox flag in the CAN_IMR global register.

If a remote frame is received while no data are ready to be sent (signal MRDY set in the CAN_MSRx), then the MMI signal is set in the CAN_MSRx. This bit is cleared by reading the CAN_MSRx.

The MRTR field in the CAN_MSRx 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. Please refer to the section “Transmission Handling” on page 663.

Figure 31-17. Producer Handling



Consumer Configuration

A mailbox is in Consumer Mode once the MOT field in the CAN_MMRx 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 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 or the MBx bit in the global CAN_TCR. The application is notified of the answer by the MRDY flag set in the CAN_MSRx. 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.

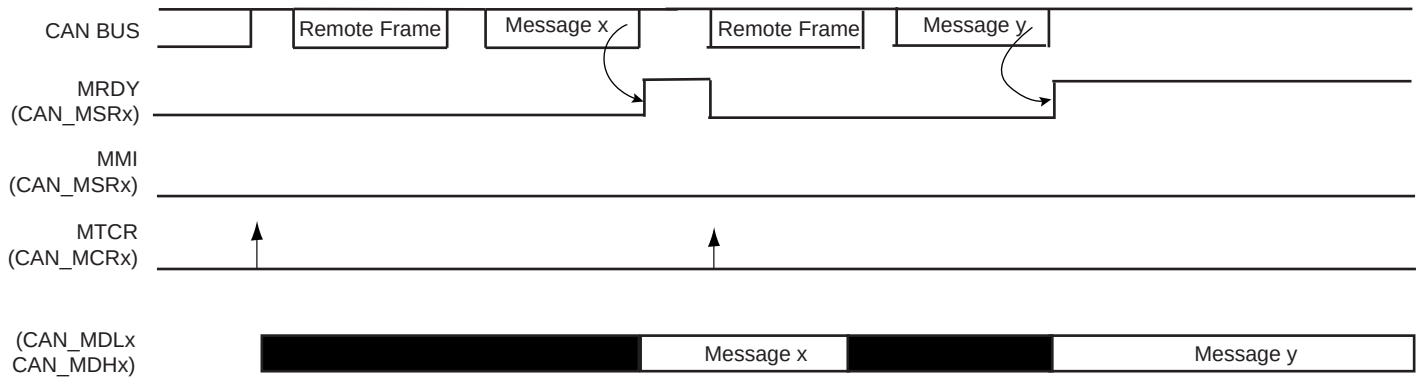
The MRTR bit in the CAN_MCRx has no effect. This field is used only when using Transmit Mode.

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, they will be lost. The application is notified by reading the MMI bit in the CAN_MSRx. 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.

Figure 31-18. Consumer Handling



31.8.4 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:

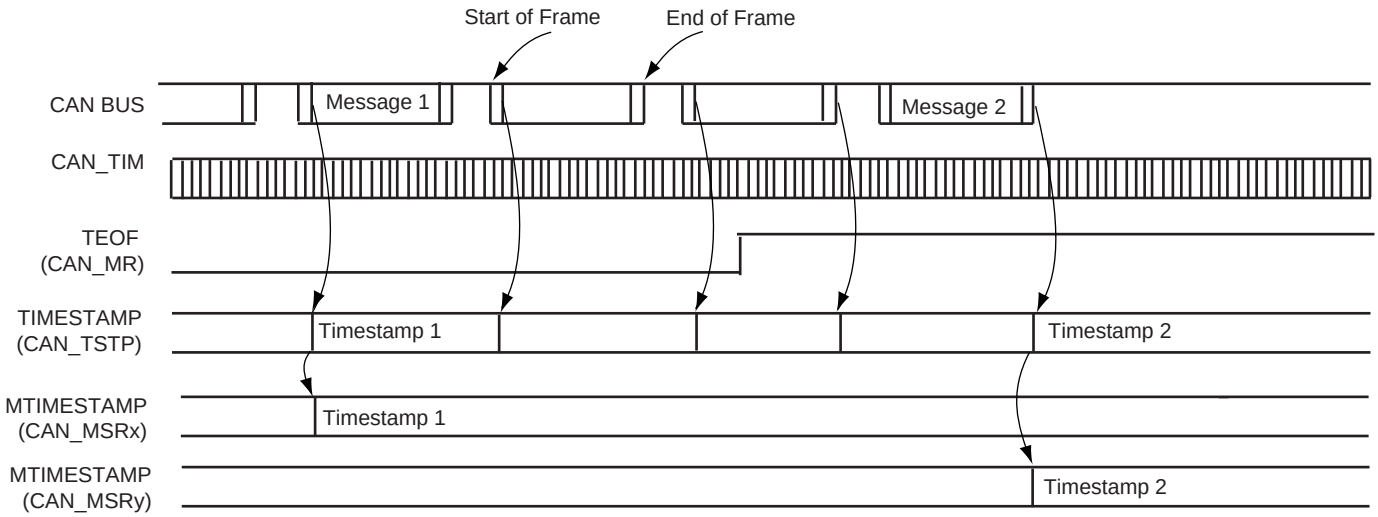
- Timestamping Mode: The value of the internal timer is captured at each Start Of Frame or each End Of Frame.
- Time Triggered Mode: The mailbox transfer operation is triggered when the internal timer reaches the mailbox trigger.

Timestamping Mode is enabled by clearing the TTM bit in the CAN_MR. Time Triggered Mode is enabled by setting the TTM bit in the CAN_MR.

31.8.4.1 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. The value read in the CAN_MSRx corresponds to the internal timer value at the Start Of Frame or the End Of Frame of the message handled by the mailbox.

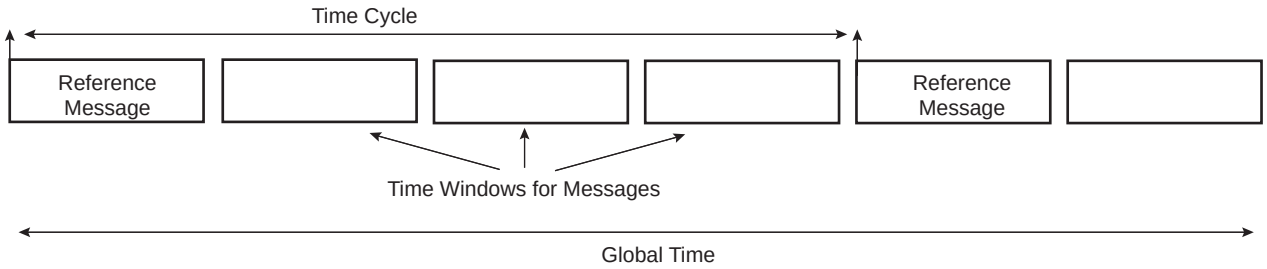
Figure 31-19. Mailbox Timestamp



31.8.4.2 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 31-20. Time Triggered Principle



Time Trigger Mode is enabled by setting the TTM field in the CAN_MR. 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. 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. 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.

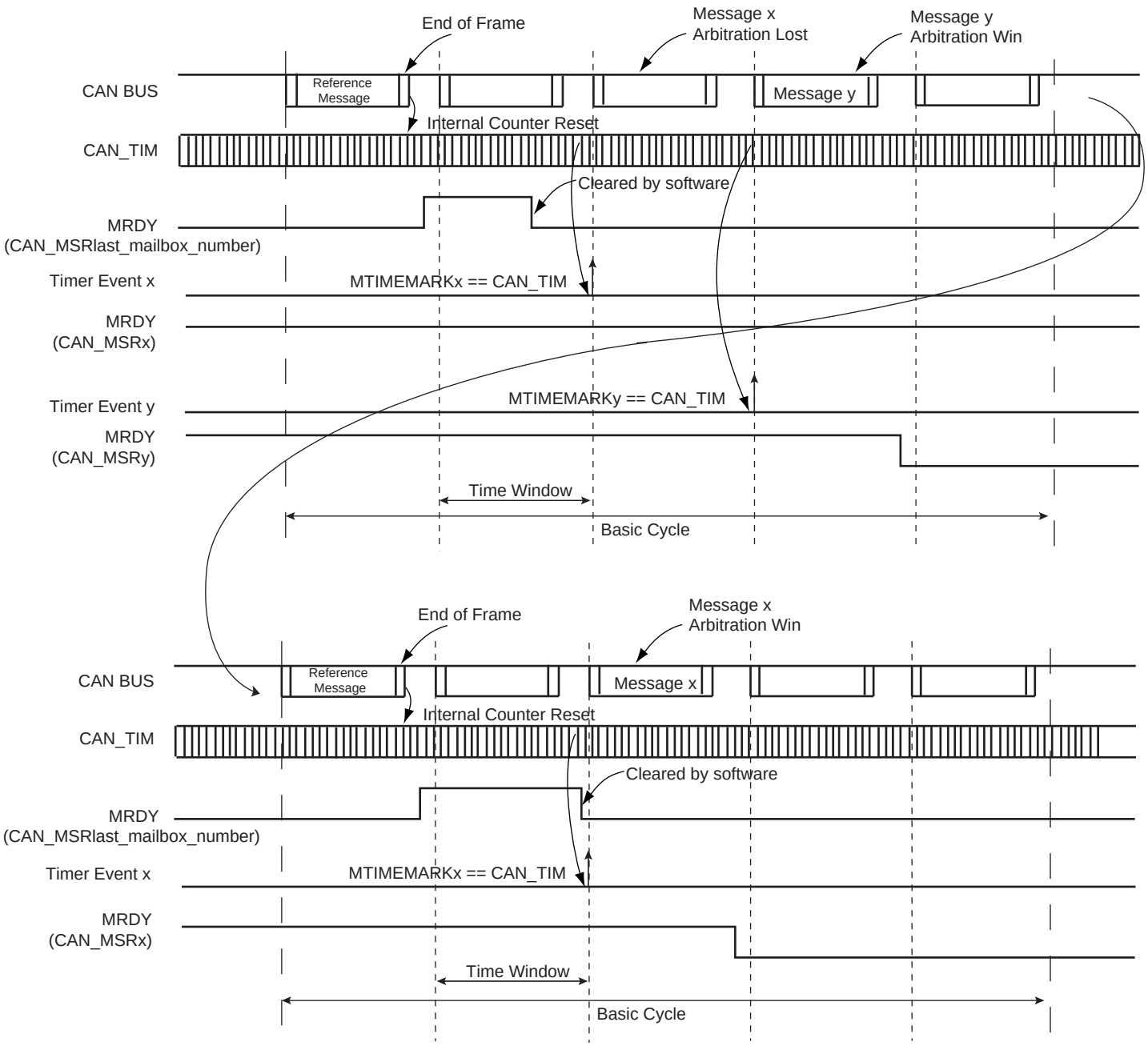
In Time Triggered Mode, transmit operations are delayed until the internal timer event for the mailbox. The application prepares a message to be sent by setting the MTCR in the CAN_MCRx. The message is not sent until the CAN_TIM value is less than the MTIMEMARK value defined in the CAN_MMRx.

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.

Freezing the Internal Timer Counter

The internal counter can be frozen by setting TIMFRZ in the CAN_MR. 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 is set when the counter is frozen. The TOVF bit in the CAN_SR is cleared by reading the CAN_SR. Depending on the corresponding interrupt mask in the CAN_IMR, an interrupt is generated when TOVF is set.

Figure 31-21. Time Triggered Operations



31.8.5 Register Write Protection

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_WPSR.

The following registers can be write-protected:

- CAN Mode Register
- CAN Baudrate Register
- CAN Message Mode Register
- CAN Message Acceptance Mask Register
- CAN Message ID Register

31.9 Controller Area Network (CAN) User Interface

Table 31-6. 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–x00E0	Reserved	–	–	–
0x00E4	Write Protection Mode Register	CAN_WPMR	Read/Write	0x0
0x00E8	Write Protection Status Register	CAN_WPSR	Read-only	0x0
0x00EC–0x01FC	Reserved	–	–	–
0x0200 + MB * 0x20 + 0x00	Mailbox Mode Register(2)	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	–

2. Mailbox number ranges from 0 to 7.

31.9.1 CAN Mode Register

Name: CAN_MR

Address: 0x40010000 (0), 0x40014000 (1)

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
DRPT	TIMFRZ	TTM	TEOF	OVL	ABM	LPM	CANEN

This register can only be written if the WPEN bit is cleared in the 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**

0: Disable Low-power mode.

1: Enable Low-power mode.

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 667.

- **DRPT: Disable Repeat**

0: When a transmit mailbox loses the bus arbitration, the transfer request remains pending.

1: When a transmit mailbox loses the bus arbitration, the transfer request is automatically aborted. It automatically raises the MABT and MRDT flags in the corresponding CAN_MSRx.

31.9.2 CAN Interrupt Enable Register

Name: CAN_IER

Address: 0x40010004 (0), 0x40014004 (1)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	BERR	FERR	AERR	SERR	CERR
23	22	21	20	19	18	17	16
TSTP	TOVF	WAKEUP	SLEEP	BOFF	ERRP	WARN	ERRA
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

- **MBx: Mailbox x Interrupt Enable**

0: No effect.

1: Enable Mailbox x interrupt.

- **ERRA: Error Active Mode Interrupt Enable**

0: No effect.

1: Enable ERRA interrupt.

- **WARN: Warning Limit Interrupt Enable**

0: No effect.

1: Enable WARN interrupt.

- **ERRP: Error Passive Mode Interrupt Enable**

0: No effect.

1: Enable ERRP interrupt.

- **BOFF: Bus Off Mode Interrupt Enable**

0: No effect.

1: Enable BOFF interrupt.

- **SLEEP: Sleep Interrupt Enable**

0: No effect.

1: Enable SLEEP interrupt.

- **WAKEUP: 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.

31.9.3 CAN Interrupt Disable Register

Name: CAN_IDR

Address: 0x40010008 (0), 0x40014008 (1)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	BERR	FERR	AERR	SERR	CERR
23	22	21	20	19	18	17	16
TSTP	TOVF	WAKEUP	SLEEP	BOFF	ERRP	WARN	ERRA
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

- **MBx: Mailbox x Interrupt Disable**

0: No effect.

1: Disable Mailbox x interrupt.

- **ERRA: Error Active Mode Interrupt Disable**

0: No effect.

1: Disable ERRA interrupt.

- **WARN: Warning Limit Interrupt Disable**

0: No effect.

1: Disable WARN interrupt.

- **ERRP: Error Passive Mode Interrupt Disable**

0: No effect.

1: Disable 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.

31.9.4 CAN Interrupt Mask Register

Name: CAN_IMR

Address: 0x4001000C (0), 0x4001400C (1)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	BERR	FERR	AERR	SERR	CERR
23	22	21	20	19	18	17	16
TSTP	TOVF	WAKEUP	SLEEP	BOFF	ERRP	WARN	ERRA
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

- **MBx: Mailbox x Interrupt Mask**

0: Mailbox x interrupt is disabled.

1: Mailbox x interrupt is enabled.

- **ERRA: Error Active Mode Interrupt Mask**

0: ERRA interrupt is disabled.

1: ERRA interrupt is enabled.

- **WARN: Warning Limit Interrupt Mask**

0: Warning Limit interrupt is disabled.

1: Warning Limit interrupt is enabled.

- **ERRP: Error Passive Mode Interrupt Mask**

0: 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.

31.9.5 CAN Status Register

Name: CAN_SR

Address: 0x40010010 (0), 0x40014010 (1)

Access: Read-only

31	30	29	28	27	26	25	24
OVLSY	TBSY	RBSY	BERR	FERR	AERR	SERR	CERR
23	22	21	20	19	18	17	16
TSTP	TOVF	WAKEUP	SLEEP	BOFF	ERRP	WARN	ERRA
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

- **MBx: Mailbox x Event**

0: No event occurred on Mailbox x.

1: An event occurred on Mailbox x.

An event corresponds to MRDY, MABT bits in the CAN_MSRx.

- **ERRA: Error Active Mode (automatically cleared by reading CAN_SR)**

0: CAN controller has not reached Error Active Mode since the last read of CAN_SR.

1: CAN controller has reached Error Active Mode since the last read of CAN_SR.

This flag is set depending on TEC and REC counter values. It is set when a node is neither in Error Passive Mode nor in Bus Off Mode.

- **WARN: Warning Limit (automatically cleared by reading CAN_SR)**

0: CAN controller Warning Limit has not been reached since the last read of CAN_SR.

1: CAN controller Warning Limit has been reached since the last read of CAN_SR.

This flag is set depending on TEC and REC counter values. It is set when at least one of the counter values has reached a value greater or equal to 96.

- **ERRP: Error Passive Mode (automatically cleared by reading CAN_SR)**

0: CAN controller has not reached Error Passive Mode since the last read of CAN_SR.

1: CAN controller has reached Error Passive Mode since the last read of CAN_SR.

This flag is set depending on TEC and REC counters values.

A node is in error passive state when TEC counter is greater or equal to 128 (decimal) or when the REC counter is greater or equal to 128 (decimal).

- **BOFF: Bus Off Mode (automatically cleared by reading CAN_SR)**

0: CAN controller has not reached Bus Off Mode.

1: CAN controller has reached Bus Off Mode since the last read of CAN_SR.

This flag is set depending on TEC counter value. A node is in bus off state when TEC counter is greater or equal to 256 (decimal).

- **SLEEP: CAN Controller in Low-power Mode**

0: CAN controller is not in Low-power mode.

1: CAN controller is in Low-power mode.

This flag is automatically reset when Low-power mode is disabled

- **WAKEUP: CAN Controller is not in Low-power Mode**

0: CAN controller is in Low-power mode.

1: CAN controller is not in Low-power mode.

When a WAKEUP event occurs, the CAN controller is synchronized with the bus activity. Messages can be transmitted or received. The CAN controller clock must be available when a WAKEUP event occurs. This flag is automatically reset when the CAN Controller enters Low-power mode.

- **TOVF: Timer Overflow (automatically cleared by reading CAN_SR)**

0: The timer has not rolled-over FFFFh to 0000h.

1: The timer rolls-over FFFFh to 0000h.

- **TSTP: Timestamp (automatically cleared by reading CAN_SR)**

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).

- **CERR: Mailbox CRC Error (automatically cleared by reading CAN_SR)**

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.

- **SERR: Mailbox Stuffing Error (automatically cleared by reading CAN_SR)**

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.

- **AERR: Acknowledgment Error (automatically cleared by reading CAN_SR)**

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.

- **FERR: Form Error (automatically cleared by reading CAN_SR)**

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

- **BERR: Bit Error (automatically cleared by reading CAN_SR)**

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.

- **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.

31.9.6 CAN Baudrate Register

Name: CAN_BR

Address: 0x40010014 (0), 0x40014014 (1)

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	SMP
23	22	21	20	19	18	17	16
–	BRP						
15	14	13	12	11	10	9	8
–	–	SJW		–	PROPAG		
7	6	5	4	3	2	1	0
–	PHASE1			–	PHASE2		

This register can only be written if the WPEN bit is cleared in the CAN Write Protection Mode Register.

Any modification on one of the fields of the CAN_BR must be done while CAN module is disabled.

To compute the different bit timings, please refer to the Section 31.7.4.1 “CAN Bit Timing Configuration” on page 649.

- **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.

$$t_{CSC} = (BRP + 1) / t_{peripheral\ clock}$$

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 peripheral clock, centered on sample point.

SMP Sampling Mode is automatically disabled if BRP = 0.

31.9.7 CAN Timer Register

Name: CAN_TIM

Address: 0x40010018 (0), 0x40014018 (1)

Access: Read-only

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

- **TIMER: Timer**

This field represents the internal CAN controller 16-bit timer value.

31.9.8 CAN Timestamp Register

Name: CAN_TIMESTP

Address: 0x4001001C (0), 0x4001401C (1)

Access: Read-only

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

- **MTIMESTAMP: Timestamp**

This field carries the value of the internal CAN controller 16-bit timer value at the start or end of frame.

If the TEOF bit is cleared in the CAN_MR, 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. If the TSTP mask in the CAN_IMR is set, an interrupt is generated while TSTP flag is set in the CAN_SR. The TSTP flag is cleared by reading the CAN_SR.

Note: The CAN_TIMESTP register is reset when the CAN is disabled then enabled via the CANEN bit in the CAN_MR.

31.9.9 CAN Error Counter Register

Name: CAN_ECR

Address: 0x40010020 (0), 0x40014020 (1)

Access: Read-only

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

- **REC: Receive Error Counter**

When a receiver detects an error, REC will be increased by one, except when the detected error is a BIT ERROR while sending an ACTIVE ERROR FLAG or an OVERLOAD FLAG.

When a receiver detects a dominant bit as the first bit after sending an ERROR FLAG, REC is increased by 8.

When a receiver detects a BIT ERROR while sending an ACTIVE ERROR FLAG, REC is increased by 8.

Any node tolerates up to 7 consecutive dominant bits after sending an ACTIVE ERROR FLAG, PASSIVE ERROR FLAG or OVERLOAD FLAG. After detecting the 14th consecutive dominant bit (in case of an ACTIVE ERROR FLAG or an 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.

31.9.10 CAN Transfer Command Register

Name: CAN_TCR

Address: 0x40010024 (0), 0x40014024 (1)

Access: Write-only

31	30	29	28	27	26	25	24
TIMRST	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

This register initializes several transfer requests at the same time.

- **MBx: Transfer Request for Mailbox x**

Mailbox Object Type	Description
Receive	It receives the next message.
Receive with overwrite	This triggers a new reception.
Transmit	Sends data prepared in the mailbox as soon as possible.
Consumer	Sends a remote frame.
Producer	Sends data prepared in the mailbox after receiving a remote frame from a consumer.

This flag clears the MRDY and MABT flags in the corresponding CAN_MSRx.

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.

31.9.11 CAN Abort Command Register

Name: CAN_ACR

Address: 0x40010028 (0), 0x40014028 (1)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

This register initializes several abort requests at the same time.

- **MBx: Abort Request for Mailbox x**

Mailbox Object Type	Description
Receive	No action
Receive with overwrite	No action
Transmit	Cancels transfer request if the message has not been transmitted to the CAN transceiver.
Consumer	Cancels the current transfer before the remote frame has been sent.
Producer	Cancels the current transfer. The next remote frame is not serviced.

It is possible to set the MACR field (in the CAN_MCRx) for each mailbox.

31.9.12 CAN Write Protection Mode Register

Name: CAN_WPMR

Address: 0x400100E4 (0), 0x400140E4 (1)

Access: Read/Write

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

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x43414E (“CAN” written in ASCII)

1: Enables the write protection if WPKEY corresponds to 0x43414E (“CAN” written in ASCII)

See Section 31.8.5 “Register Write Protection” for the list of registers that can be write-protected.

- **WPKEY: Write Protection Key Password**

Value	Name	Description
0x43414E	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0

31.9.13 CAN Write Protection Status Register

Name: CAN_WPSR

Address: 0x400100E8 (0), 0x400140E8 (1)

Access: Read-only

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

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of the CAN_WPSR.

1: A write protection violation has occurred since the last read of the CAN_WPSR. 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**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

31.9.14 CAN Message Mode Register

Name: CAN_MMRx [x=0..7]

Address: 0x40010200 (0)[0], 0x40010220 (0)[1], 0x40010240 (0)[2], 0x40010260 (0)[3], 0x40010280 (0)[4], 0x400102A0 (0)[5], 0x400102C0 (0)[6], 0x400102E0 (0)[7], 0x40014200 (1)[0], 0x40014220 (1)[1], 0x40014240 (1)[2], 0x40014260 (1)[3], 0x40014280 (1)[4], 0x400142A0 (1)[5], 0x400142C0 (1)[6], 0x400142E0 (1)[7]

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the 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 667.

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

31.9.15 CAN Message Acceptance Mask Register

Name: CAN_MAMx [x=0..7]

Address: 0x40010204 (0)[0], 0x40010224 (0)[1], 0x40010244 (0)[2], 0x40010264 (0)[3], 0x40010284 (0)[4], 0x400102A4 (0)[5], 0x400102C4 (0)[6], 0x400102E4 (0)[7], 0x40014204 (1)[0], 0x40014224 (1)[1], 0x40014244 (1)[2], 0x40014264 (1)[3], 0x40014284 (1)[4], 0x400142A4 (1)[5], 0x400142C4 (1)[6], 0x400142E4 (1)[7]

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the 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.

0: The corresponding message ID bit is not masked

1: The corresponding message ID bit is masked

- **MIDvA: Identifier for standard frame mode**

Acceptance mask for corresponding field of the message IDvA register of the mailbox.

0: The corresponding message ID bit is not masked

1: The corresponding message ID bit is masked

- **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.

31.9.16 CAN Message ID Register

Name: CAN_MIDx [x=0..7]

Address: 0x40010208 (0)[0], 0x40010228 (0)[1], 0x40010248 (0)[2], 0x40010268 (0)[3], 0x40010288 (0)[4], 0x400102A8 (0)[5], 0x400102C8 (0)[6], 0x400102E8 (0)[7], 0x40014208 (1)[0], 0x40014228 (1)[1], 0x40014248 (1)[2], 0x40014268 (1)[3], 0x40014288 (1)[4], 0x400142A8 (1)[5], 0x400142C8 (1)[6], 0x400142E8 (1)[7]

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the 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**

31.9.17 CAN Message Family ID Register

Name: CAN_MFIDx [x=0..7]

Address: 0x4001020C (0)[0], 0x4001022C (0)[1], 0x4001024C (0)[2], 0x4001026C (0)[3], 0x4001028C (0)[4], 0x400102AC (0)[5], 0x400102CC (0)[6], 0x400102EC (0)[7], 0x4001420C (1)[0], 0x4001422C (1)[1], 0x4001424C (1)[2], 0x4001426C (1)[3], 0x4001428C (1)[4], 0x400142AC (1)[5], 0x400142CC (1)[6], 0x400142EC (1)[7]

Access: Read-only

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

- **MFID: Family ID**

This field contains the concatenation of CAN_MIDx register bits masked by the CAN_MAMx register. This field is useful to speed up message ID decoding. The message acceptance procedure is described below.

As an example:

```
CAN_MIDx = 0x305A4321
CAN_MAMx = 0x3FF0F0FF
CAN_MFIDx = 0x000000A3
```

31.9.18 CAN Message Status Register

Name: CAN_MSRx [x=0..7]

Address: 0x40010210 (0)[0], 0x40010230 (0)[1], 0x40010250 (0)[2], 0x40010270 (0)[3], 0x40010290 (0)[4], 0x400102B0 (0)[5], 0x400102D0 (0)[6], 0x400102F0 (0)[7], 0x40014210 (1)[0], 0x40014230 (1)[1], 0x40014250 (1)[2], 0x40014270 (1)[3], 0x40014290 (1)[4], 0x400142B0 (1)[5], 0x400142D0 (1)[6], 0x400142F0 (1)[7]

Access: Read-only

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

These register fields are updated each time a message transfer is received or aborted.

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). If the field CAN_MR.TEOF 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 field CAN_MR.TEOF 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 (cleared by writing MTCR or MACR in the CAN_MCRx)**

An interrupt is triggered when MABT is set.

0: Previous transfer is not aborted.

1: Previous transfer has been aborted.

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 (cleared by writing MTCR or MACR in the CAN_MCRx)**

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.

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 (cleared by reading CAN_MSRx)**

0: No message has been ignored during the previous transfer

1: At least one message has been ignored during the previous transfer

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.

31.9.19 CAN Message Data Low Register

Name: CAN_MDLx [x=0..7]

Address: 0x40010214 (0)[0], 0x40010234 (0)[1], 0x40010254 (0)[2], 0x40010274 (0)[3], 0x40010294 (0)[4], 0x400102B4 (0)[5], 0x400102D4 (0)[6], 0x400102F4 (0)[7], 0x40014214 (1)[0], 0x40014234 (1)[1], 0x40014254 (1)[2], 0x40014274 (1)[3], 0x40014294 (1)[4], 0x400142B4 (1)[5], 0x400142D4 (1)[6], 0x400142F4 (1)[7]

Access: Read/Write

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

- **MDL: Message Data Low Value**

When MRDY bit is set in the CAN_MSRx, 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 bit in the CAN_MSRx. In this mode, the software application must re-read CAN_MDH and CAN_MDL, while the MMI bit in the CAN_MSRx 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]

31.9.20 CAN Message Data High Register

Name: CAN_MDHx [x=0..7]

Address: 0x40010218 (0)[0], 0x40010238 (0)[1], 0x40010258 (0)[2], 0x40010278 (0)[3], 0x40010298 (0)[4], 0x400102B8 (0)[5], 0x400102D8 (0)[6], 0x400102F8 (0)[7], 0x40014218 (1)[0], 0x40014238 (1)[1], 0x40014258 (1)[2], 0x40014278 (1)[3], 0x40014298 (1)[4], 0x400142B8 (1)[5], 0x400142D8 (1)[6], 0x400142F8 (1)[7]

Access: Read/Write

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

- **MDH: Message Data High Value**

When MRDY bit is set in the CAN_MSRx, 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 bit in the CAN_MSRx. In this mode, the software application must re-read CAN_MDH and CAN_MDL, while the MMI bit in the CAN_MSRx 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]

31.9.21 CAN Message Control Register

Name: CAN_MCRx [x=0..7]

Address: 0x4001021C (0)[0], 0x4001023C (0)[1], 0x4001025C (0)[2], 0x4001027C (0)[3], 0x4001029C (0)[4], 0x400102BC (0)[5], 0x400102DC (0)[6], 0x400102FC (0)[7], 0x4001421C (1)[0], 0x4001423C (1)[1], 0x4001425C (1)[2], 0x4001427C (1)[3], 0x4001429C (1)[4], 0x400142BC (1)[5], 0x400142DC (1)[6], 0x400142FC (1)[7]

Access: Write-only

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

- **MDLC: Mailbox Data Length Code**

Mailbox Object Type	Description
Receive	No action.
Receive with overwrite	No action.
Transmit	Length of the mailbox message.
Consumer	No action.
Producer	Length of the mailbox message to be sent after the remote frame reception.

- **MRTR: Mailbox Remote Transmission Request**

Mailbox Object Type	Description
Receive	No action
Receive with overwrite	No action
Transmit	Set the RTR bit in the sent frame
Consumer	No action, the RTR bit in the sent frame is set automatically
Producer	No action

Consumer situations can be handled automatically by setting the mailbox object type in Consumer. This requires only one mailbox.

It can also be handled using two mailboxes, one in reception, the other in transmission. The MRTR and the MTCR bits must be set in the same time.

- **MACR: Abort Request for Mailbox x**

Mailbox Object Type	Description
Receive	No action
Receive with overwrite	No action
Transmit	Cancels transfer request if the message has not been transmitted to the CAN transceiver.
Consumer	Cancels the current transfer before the remote frame has been sent.
Producer	Cancels the current transfer. The next remote frame will not be serviced.

This flag clears the MRDY and MABT flags in the CAN_MSRx.

It is possible to set the MACR field for several mailboxes in the same time, setting several bits to the CAN_ACR.

- **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.

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.

32. Chip Identifier (CHIPID)

32.1 Description

Chip Identifier (CHIPID) registers are used to recognize 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 CHIPID registers are embedded: Chip ID Register (CHIPID_CIDR) and Chip ID Extension Register (CHIPID_EXID). Both registers contain a hard-wired value that is read-only.

The CHIPID_CIDR register contains the following fields:

- VERSION: Identifies the revision of the silicon
- EPROC: Indicates the embedded ARM processor
- NVPTYP and NVPSIZ: Identify the type of embedded non-volatile memory and the size
- SRAMSIZ: Indicates the size of the embedded SRAM
- ARCH: Identifies the set of embedded peripherals
- EXT: Shows the use of the extension identifier register

The CHIPID_EXID register is device-dependent and reads 0 if CHIPID_CIDR.EXT = 0.

32.2 Embedded Characteristics

- Chip ID Registers
  - Identification of the Device Revision, Sizes of the Embedded Memories, Set of Peripherals, Embedded Processor

Table 32-1. SAM4E Chip ID Registers

Chip Name	CHIPID_CIDR	CHIPID_EXID
SAM4E16E	0xA3CC_0CE0	0x0012_0200
SAM4E8E	0xA3CC_0CE0	0x0012_0208
SAM4E16C	0xA3CC_0CE0	0x0012_0201
SAM4E8C	0xA3CC_0CE0	0x0012_0209

32.3 Chip Identifier (CHIPID) User Interface

Table 32-2. Register Mapping

Offset	Register	Name	Access	Reset
0x0	Chip ID Register	CHIPID_CIDR	Read-only	–
0x4	Chip ID Extension Register	CHIPID_EXID	Read-only	–

32.3.1 Chip ID Register

Name: CHIPID_CIDR

Address: 0x400E0740

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
0	SAM x7	Cortex-M7
1	ARM946ES	ARM946ES
2	ARM7TDMI	ARM7TDMI
3	CM3	Cortex-M3
4	ARM920T	ARM920T
5	ARM926EJS	ARM926EJS
6	CA5	Cortex-A5
7	CM4	Cortex-M4

- **NVPSIZ: Nonvolatile Program Memory Size**

Value	Name	Description
0	NONE	None
1	8K	8 Kbytes
2	16K	16 Kbytes
3	32K	32 Kbytes
4	–	Reserved
5	64K	64 Kbytes
6	–	Reserved
7	128K	128 Kbytes
8	160K	160 Kbytes
9	256K	256 Kbytes
10	512K	512 Kbytes

Value	Name	Description
11	–	Reserved
12	1024K	1024 Kbytes
13	–	Reserved
14	2048K	2048 Kbytes
15	–	Reserved

- **NVPSIZ2: Second Nonvolatile Program Memory Size**

Value	Name	Description
0	NONE	None
1	8K	8 Kbytes
2	16K	16 Kbytes
3	32K	32 Kbytes
4	–	Reserved
5	64K	64 Kbytes
6	–	Reserved
7	128K	128 Kbytes
8	–	Reserved
9	256K	256 Kbytes
10	512K	512 Kbytes
11	–	Reserved
12	1024K	1024 Kbytes
13	–	Reserved
14	2048K	2048 Kbytes
15	–	Reserved

- **SRAMSIZ: Internal SRAM Size**

Value	Name	Description
0	48K	48 Kbytes
1	192K	192 Kbytes
2	384K	384 Kbytes
3	6K	6 Kbytes
4	24K	24 Kbytes
5	4K	4 Kbytes
6	80K	80 Kbytes
7	160K	160 Kbytes
8	8K	8 Kbytes
9	16K	16 Kbytes
10	32K	32 Kbytes
11	64K	64 Kbytes

Value	Name	Description
12	128K	128 Kbytes
13	256K	256 Kbytes
14	96K	96 Kbytes
15	512K	512 Kbytes

- **ARCH: Architecture Identifier**

Value	Name	Description
0x3C	SAM4E	SAM4E Series

- **NVPTYP: Nonvolatile Program Memory Type**

Value	Name	Description
0	ROM	ROM
1	ROMLESS	ROMless or on-chip Flash
2	FLASH	Embedded Flash Memory
3	ROM_FLASH	ROM and Embedded Flash Memory ● NVPSIZ is ROM size ● NVPSIZ2 is Flash size
4	SRAM	SRAM emulating ROM

- **EXT: Extension Flag**

0: Chip ID has a single register definition without extension.

1: An extended Chip ID exists.

32.3.2 Chip ID Extension Register

Name: CHIPID_EXID

Address: 0x400E0744

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**

This field is cleared if CHIPID_CIDR.EXT = 0.CHIPID_EXID[1:0]: Package Type

Value	Name	Description
0	PACKAGE_TYPE	Package 144
1	PACKAGE_TYPE	Package 100

CHIPID_EXID[4:2]: Flash Size

Value	Name	Description
0	FLASH_SIZE	1024 Kbytes
2	FLASH_SIZE	512 Kbytes

CHIPID_EXID[31:5]: Product Number

Value	Name	Description
0x0012_020	PRODUCT_NUMBER	SAM4E Product Series

33. Parallel Input/Output Controller (PIO)

33.1 Description

The Parallel Input/Output Controller (PIO) manages up to 32 fully programmable input/output lines. Each I/O line may be dedicated as a general-purpose I/O or be assigned to a function of an embedded peripheral. This ensures effective optimization of the pins of the 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 peripheral clock cycle.
- A debouncing filter providing rejection of unwanted pulses from key or push button operations.
- Multi-drive capability similar to an open drain I/O line.
- Control of the pull-up and pull-down of the I/O line.
- Input visibility and output control.

The PIO Controller also features a synchronous output providing up to 32 bits of data output in a single write operation.

An 8-bit parallel capture mode is also available which can be used to interface a CMOS digital image sensor, an ADC, a DSP synchronous port in synchronous mode, etc.

33.2 Embedded Characteristics

- Up to 32 Programmable I/O Lines
- Fully Programmable through Set/Clear Registers
- Multiplexing of Four Peripheral Functions per I/O Line
- For each I/O Line (Whether Assigned to a Peripheral or Used as General Purpose I/O)
  - Input Change Interrupt
  - Programmable Glitch Filter
  - Programmable Debouncing Filter
  - Multi-drive Option Enables Driving in Open Drain
  - Programmable Pull-Up on Each I/O Line
  - Pin Data Status Register, Supplies Visibility of the Level on the Pin at Any Time
  - Additional Interrupt Modes on a Programmable Event: Rising Edge, Falling Edge, Low-Level or High-Level
  - Lock of the Configuration by the Connected Peripheral
- Synchronous Output, Provides Set and Clear of Several I/O Lines in a Single Write
- Register Write Protection
- Programmable Schmitt Trigger Inputs
- Programmable I/O Delay
- Parallel Capture Mode
  - Can Be Used to Interface a CMOS Digital Image Sensor, an ADC, etc.
  - One Clock, 8-bit Parallel Data and Two Data Enable on I/O Lines
  - Data Can be Sampled Every Other Time (For Chrominance Sampling Only)
  - Supports Connection of One Peripheral DMA Controller (PDC) Channel Which Offers Buffer Reception Without Processor Intervention

33.3 Block Diagram

Figure 33-1. Block Diagram

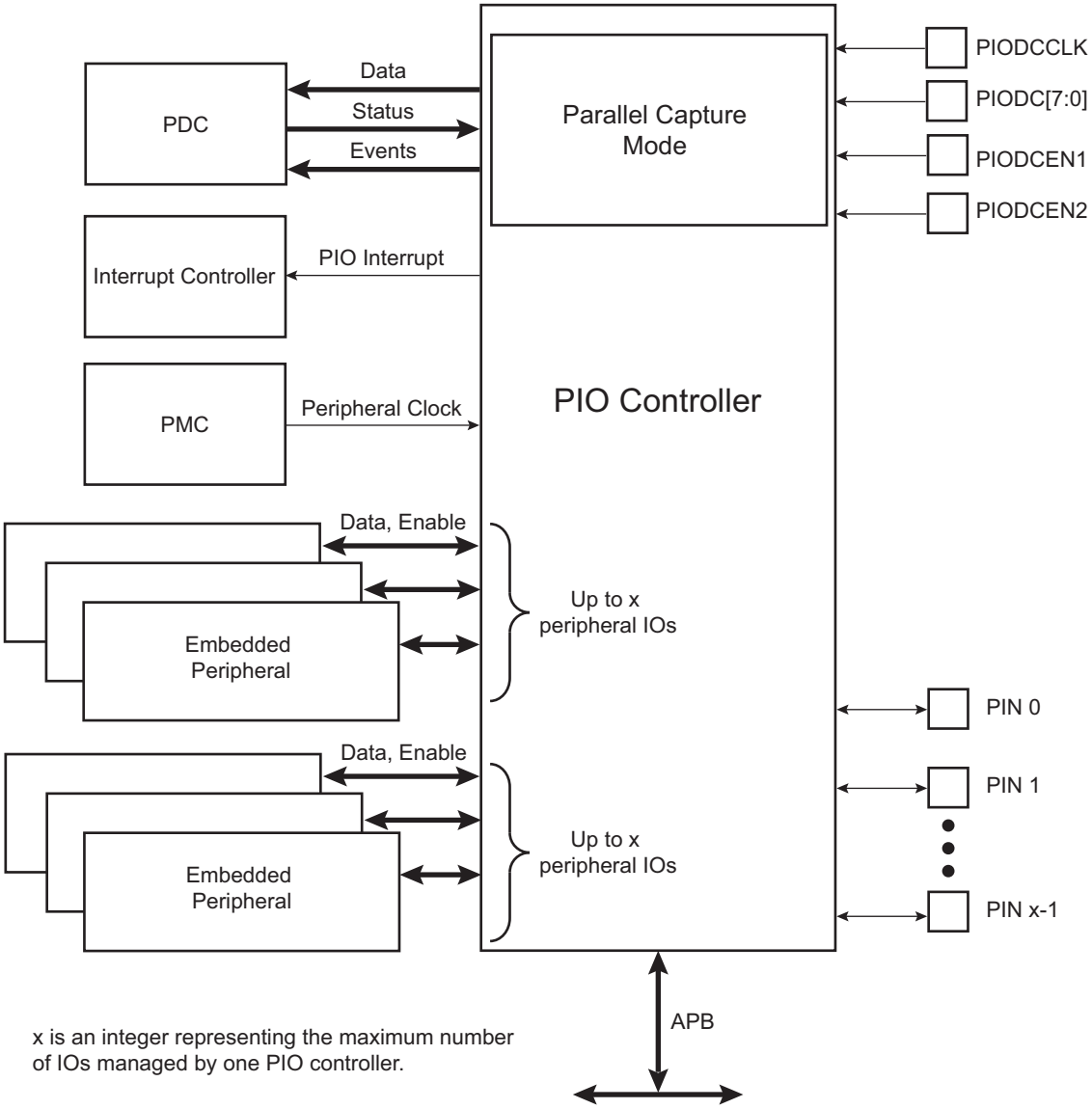


Table 33-1. Signal Description

Signal Name	Signal Description	Signal Type
PIODCCLK	Parallel Capture Mode Clock	Input
PIODC[7:0]	Parallel Capture Mode Data	Input
PIODCEN1	Parallel Capture Mode Data Enable 1	Input
PIODCEN2	Parallel Capture Mode Data Enable 2	Input

33.4 Product Dependencies

33.4.1 Pin Multiplexing

Each pin is configurable, depending on the product, as either a general-purpose I/O line only, or as an I/O line multiplexed with one or two peripheral I/Os. As the multiplexing is hardware defined and thus product-dependent, the hardware designer and programmer must carefully determine the configuration of the PIO Controllers required by their application. When an I/O line is general-purpose only, i.e., not multiplexed with any peripheral I/O, programming of the PIO Controller regarding the assignment to a peripheral has no effect and only the PIO Controller can control how the pin is driven by the product.

33.4.2 Power Management

The Power Management Controller controls the peripheral clock in order to save power. Writing any of the registers of the user interface does not require the peripheral clock to be enabled. This means that the configuration of the I/O lines does not require the peripheral 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, the interrupt modes on a programmable event and the read of the pin level require the clock to be validated.

After a hardware reset, the peripheral clock is disabled by default.

The user must configure the Power Management Controller before any access to the input line information.

33.4.3 Interrupt Sources

For interrupt handling, the PIO Controllers are considered as user peripherals. This means that the PIO Controller interrupt lines are connected among the interrupt sources. Refer to the PIO Controller peripheral identifier in the Peripheral Identifiers table to identify the interrupt sources dedicated to the PIO Controllers. Using the PIO Controller requires the Interrupt Controller to be programmed first.

The PIO Controller interrupt can be generated only if the peripheral clock is enabled.

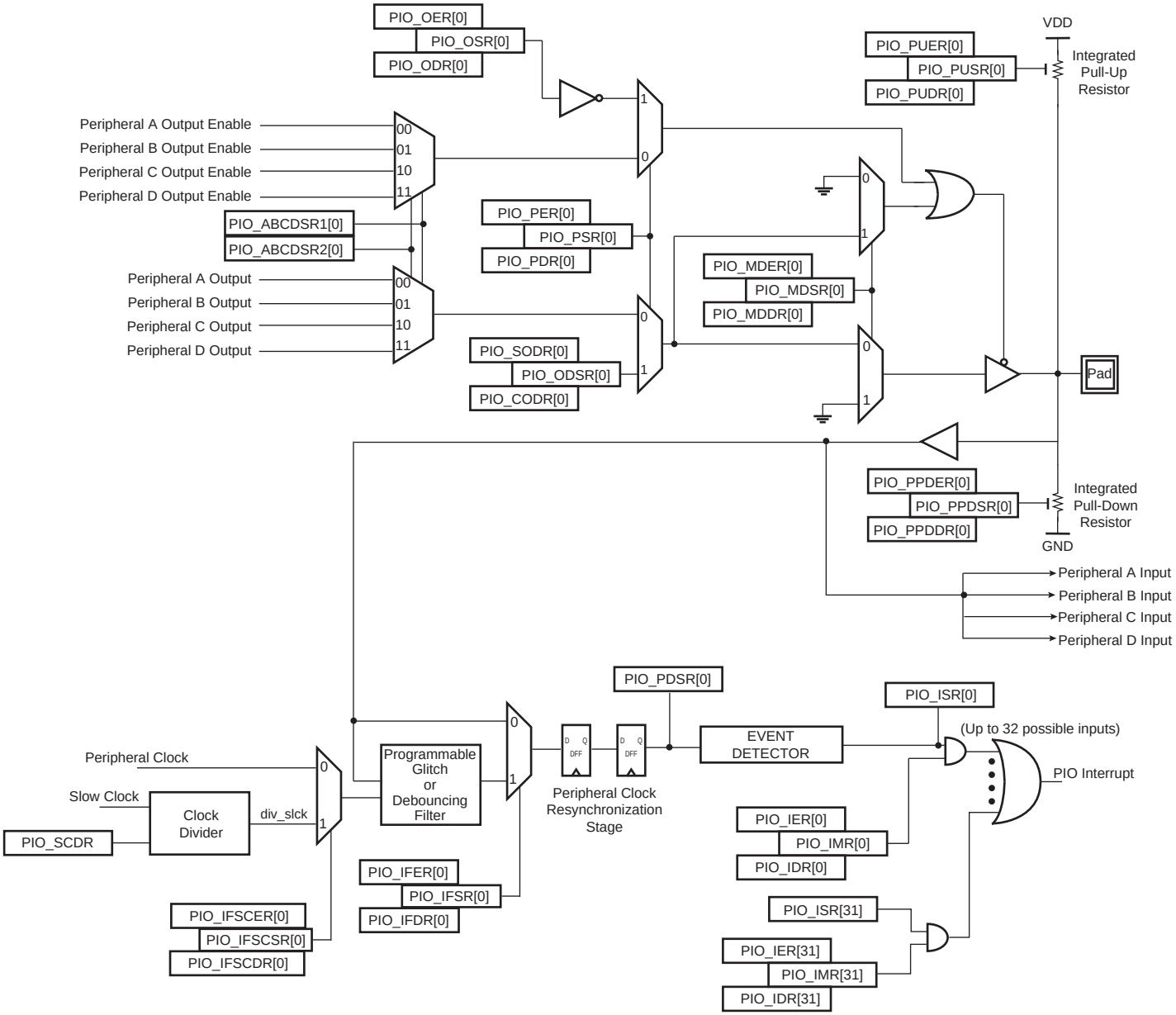
Table 33-2. Peripheral IDs

Instance	ID
PIOA	9
PIOB	10
PIOC	11
PIOD	12
PIOE	13

33.5 Functional Description

The PIO Controller features up to 32 fully-programmable I/O lines. Most of the control logic associated to each I/O is represented in Figure 33-2. In this description each signal shown represents one of up to 32 possible indexes.

Figure 33-2. I/O Line Control Logic



33.5.1 Pull-up and Pull-down Resistor Control

Each I/O line is designed with an embedded pull-up resistor and an embedded pull-down resistor. The pull-up resistor can be enabled or disabled by writing to the Pull-up Enable Register (PIO_PUER) or Pull-up Disable Register (PIO_PUDR), respectively. Writing to these registers results in setting or clearing the corresponding bit in the Pull-up Status Register (PIO_PUSR). Reading a one in PIO_PUSR means the pull-up is disabled and reading a zero means the pull-up is enabled. The pull-down resistor can be enabled or disabled by writing the Pull-down Enable Register (PIO_PPDER) or the Pull-down Disable Register (PIO_PPDDR), respectively. Writing in these

registers results in setting or clearing the corresponding bit in the Pull-down Status Register (PIO_PPDSR). Reading a one in PIO_PPDSR means the pull-up is disabled and reading a zero means the pull-down is enabled.

Enabling the pull-down resistor while the pull-up resistor is still enabled is not possible. In this case, the write of PIO_PPDER for the relevant I/O line is discarded. Likewise, enabling the pull-up resistor while the pull-down resistor is still enabled is not possible. In this case, the write of PIO_PUER for the relevant I/O line is discarded.

Control of the pull-up resistor is possible regardless of the configuration of the I/O line.

After reset, depending on the I/O, pull-up or pull-down can be set.

33.5.2 I/O Line or Peripheral Function Selection

When a pin is multiplexed with one or two peripheral functions, the selection is controlled with the Enable Register (PIO_PER) and the Disable Register (PIO_PDR). The Status Register (PIO_PSR) 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 zero indicates that the pin is controlled by the corresponding on-chip peripheral selected in the ABCD Select registers (PIO_ABCDSR1 and PIO_ABCDSR2). A value of one 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 a one for the corresponding bit.

After reset, the I/O lines are controlled by the PIO Controller, i.e., PIO_PSR resets at one. 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 and depends on the multiplexing of the device.

33.5.3 Peripheral A or B or C or D Selection

The PIO Controller provides multiplexing of up to four peripheral functions on a single pin. The selection is performed by writing PIO_ABCDSR1 and PIO_ABCDSR2.

For each pin:

- The corresponding bit at level zero in PIO_ABCDSR1 and the corresponding bit at level zero in PIO_ABCDSR2 means peripheral A is selected.
- The corresponding bit at level one in PIO_ABCDSR1 and the corresponding bit at level zero in PIO_ABCDSR2 means peripheral B is selected.
- The corresponding bit at level zero in PIO_ABCDSR1 and the corresponding bit at level one in PIO_ABCDSR2 means peripheral C is selected.
- The corresponding bit at level one in PIO_ABCDSR1 and the corresponding bit at level one in PIO_ABCDSR2 means peripheral D is selected.

Note that multiplexing of peripheral lines A, B, C and D only affects the output line. The peripheral input lines are always connected to the pin input (see Figure 33-2).

Writing in PIO_ABCDSR1 and PIO_ABCDSR2 manages the multiplexing regardless of the configuration of the pin. However, assignment of a pin to a peripheral function requires a write in PIO_ABCDSR1 and PIO_ABCDSR2 in addition to a write in PIO_PDR.

After reset, PIO_ABCDSR1 and PIO_ABCDSR2 are zero, 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.

If the software selects a peripheral A, B, C or D which does not exist for a pin, no alternate functions are enabled for this pin and the selection is taken into account. The PIO Controller does not carry out checks to prevent selection of a peripheral which does not exist.

33.5.4 Output Control

When the I/O line is assigned to a peripheral function, i.e., the corresponding bit in PIO_PSR is at zero, the drive of the I/O line is controlled by the peripheral. Peripheral A or B or C or D depending on the value in PIO_ABCDSR1 and PIO_ABCDSR2 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 the Output Enable Register (PIO_OER) and Output Disable Register (PIO_ODR). The results of these write operations are detected in the Output Status Register (PIO_OSR). When a bit in this register is at zero, the corresponding I/O line is used as an input only. When the bit is at one, 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 the Set Output Data Register (PIO_SODR) and the Clear Output Data Register (PIO_CODR). These write operations, respectively, set and clear the Output Data Status Register (PIO_ODSR), 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 affects PIO_ODSR. This is important as it defines the first level driven on the I/O line.

33.5.5 Synchronous Data Output

Clearing one or more PIO line(s) and setting another one or more PIO line(s) synchronously cannot be done by using PIO_SODR and PIO_CODR. 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. Only bits unmasked by the Output Write Status Register (PIO_OWSR) are written. The mask bits in PIO_OWSR are set by writing to the Output Write Enable Register (PIO_OWER) and cleared by writing to the Output Write Disable Register (PIO_OWDR).

After reset, the synchronous data output is disabled on all the I/O lines as PIO_OWSR resets at 0x0.

33.5.6 Multi-Drive Control (Open Drain)

Each I/O can be independently programmed in open drain by using the multi-drive feature. This feature permits several drivers to be connected on the I/O line which is driven low only by each device. An external pull-up resistor (or enabling of the internal one) is generally required to guarantee a high level on the line.

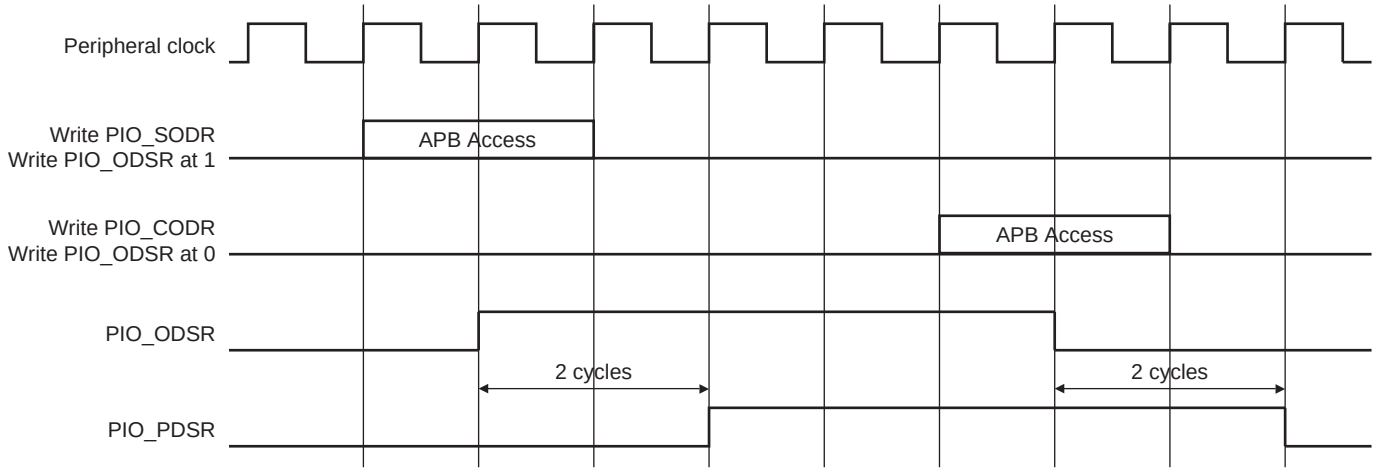
The multi-drive feature is controlled by the Multi-driver Enable Register (PIO_MDER) and the Multi-driver Disable Register (PIO_MDDR). The multi-drive can be selected whether the I/O line is controlled by the PIO Controller or assigned to a peripheral function. The Multi-driver Status Register (PIO_MDSR) indicates the pins that are configured to support external drivers.

After reset, the multi-drive feature is disabled on all pins, i.e., PIO_MDSR resets at value 0x0.

33.5.7 Output Line Timings

Figure 33-3 shows how the outputs are driven either by writing PIO_SODR or PIO_CODR, or by directly writing PIO_ODSR. This last case is valid only if the corresponding bit in PIO_OWSR is set. Figure 33-3 also shows when the feedback in the Pin Data Status Register (PIO_PDSR) is available.

Figure 33-3. Output Line Timings



33.5.8 Inputs

The level on each I/O line can be read through PIO_PDSR. This register indicates the level of the I/O lines regardless of their configuration, whether uniquely as an input, or driven by the PIO Controller, or driven by a peripheral.

Reading the I/O line levels requires the clock of the PIO Controller to be enabled, otherwise PIO_PDSR reads the levels present on the I/O line at the time the clock was disabled.

33.5.9 Input Glitch and Debouncing Filters

Optional input glitch and debouncing filters are independently programmable on each I/O line.

The glitch filter can filter a glitch with a duration of less than 1/2 peripheral clock 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 PIO Input Filter Slow Clock Disable Register (PIO_IFSCDR) and the PIO Input Filter Slow Clock Enable Register (PIO_IFSCER). Writing PIO_IFSCDR and PIO_IFSCER, respectively, sets and clears bits in the Input Filter Slow Clock Status Register (PIO_IFSCSR).

The current selection status can be checked by reading the PIO_IFSCSR.

- If PIO_IFSCSR[i] = 0: The glitch filter can filter a glitch with a duration of less than 1/2 master clock period.
- If PIO_IFSCSR[i] = 1: The debouncing filter can filter a pulse with a duration of less than 1/2 programmable divided slow clock period.

For the debouncing filter, the period of the divided slow clock is defined by writing in the DIV field of the Slow Clock Divider Debouncing Register (PIO_SCDR):

$t_{div\_slck} = ((DIV + 1) \times 2) \times t_{slck}$

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 peripheral clock or divided slow clock depending on PIO_IFSCDR and PIO_IFSCER programming) is automatically rejected, while a pulse with a duration of one selected clock (peripheral clock or divided slow clock) cycle or more is accepted. For pulse durations between 1/2 selected clock cycle and one 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 one 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, illustrated in Figure 33-4 and Figure 33-5.

The glitch filters are controlled by the Input Filter Enable Register (PIO_IFER), the Input Filter Disable Register (PIO_IFDR) and the Input Filter Status Register (PIO_IFSR). 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 peripheral clock is enabled.

Figure 33-4. Input Glitch Filter Timing

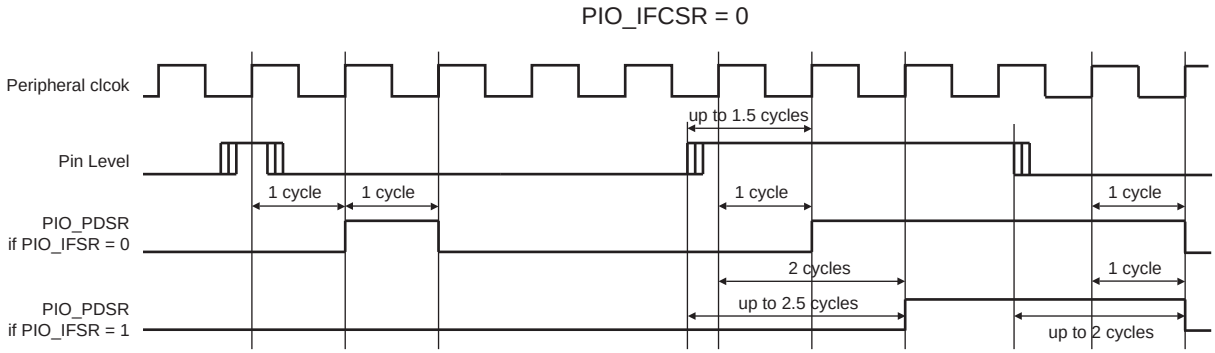
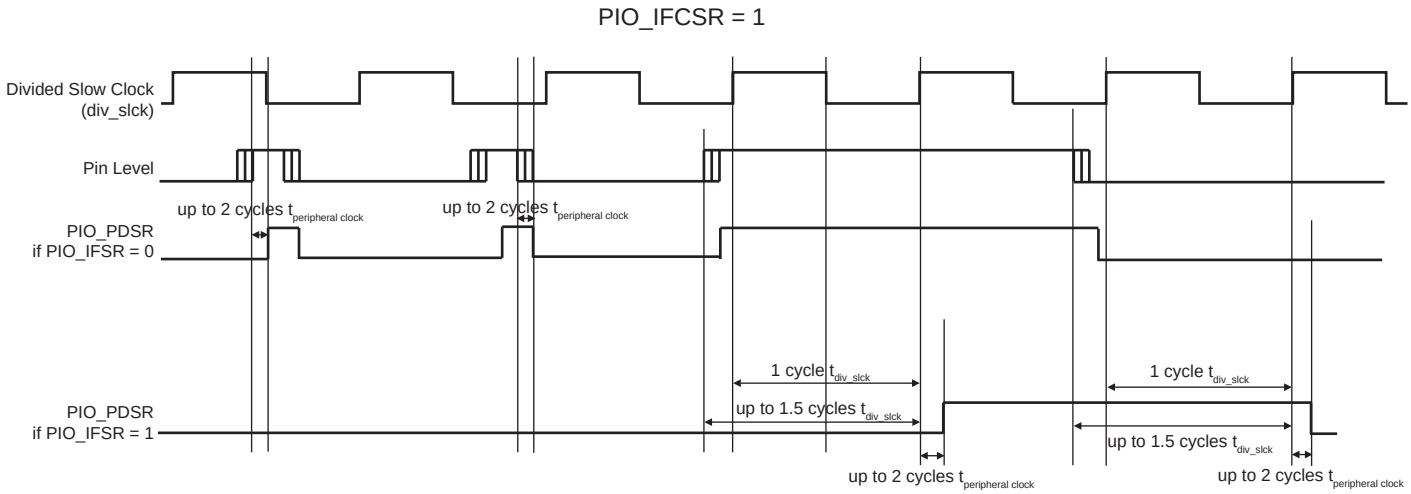


Figure 33-5. Input Debouncing Filter Timing



33.5.10 Input Edge/Level Interrupt

The PIO Controller can be programmed to generate an interrupt when it detects an edge or a level on an I/O line. The Input Edge/Level interrupt is controlled by writing the Interrupt Enable Register (PIO_IER) and the Interrupt Disable Register (PIO_IDR), which enable and disable the input change interrupt respectively by setting and clearing the corresponding bit in the Interrupt Mask Register (PIO_IMR). As input change detection is possible only by comparing two successive samplings of the input of the I/O line, the peripheral 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 Additional Interrupt Modes Enable Register (PIO_AIMER) and Additional Interrupt Modes Disable Register (PIO_AIMDR). The current state of this selection can be read through the Additional Interrupt Modes Mask Register (PIO_AIMMR).

These additional modes are:

- Rising edge detection
- Falling edge detection
- Low-level detection
- High-level detection

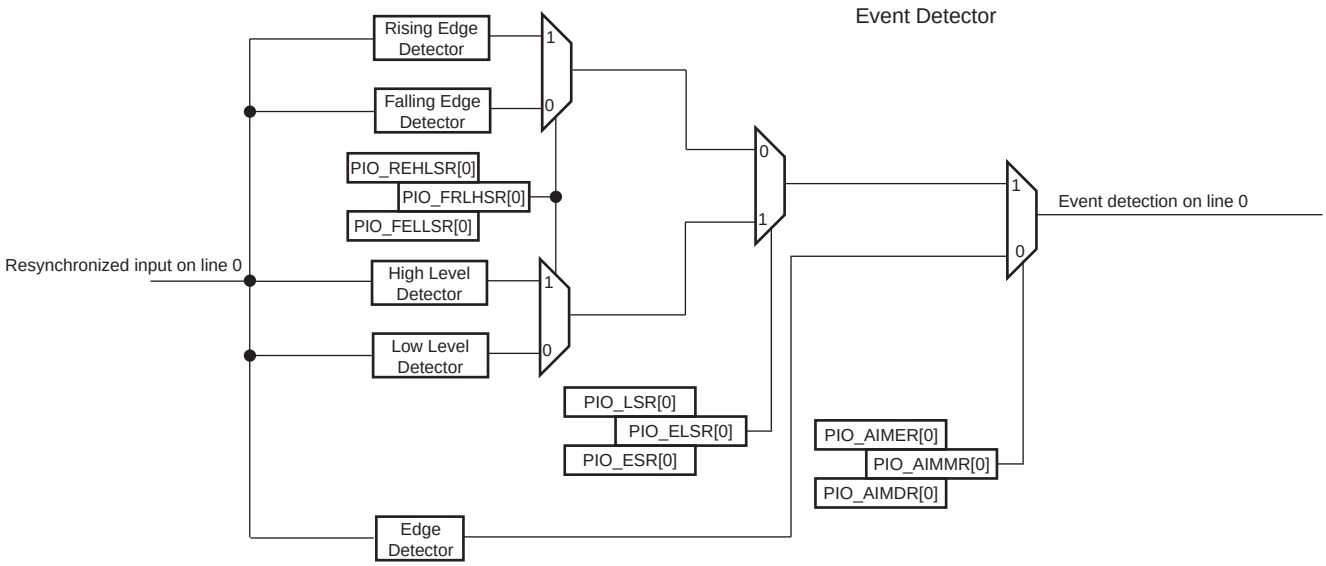
In order to select an additional interrupt mode:

- The type of event detection (edge or level) must be selected by writing in the Edge Select Register (PIO_ESR) and Level Select Register (PIO_LSR) which select, respectively, the edge and level detection. The current status of this selection is accessible through the Edge/Level Status Register (PIO_ELSR).
- The polarity of the event detection (rising/falling edge or high/low-level) must be selected by writing in the Falling Edge/Low-Level Select Register (PIO_FELLSR) and Rising Edge/High-Level Select Register (PIO_REHLSR) which allow to select falling or rising edge (if edge is selected in PIO_ELSR) edge or high- or low-level detection (if level is selected in PIO_ELSR). The current status of this selection is accessible through the Fall/Rise - Low/High Status Register (PIO_FRLHSR).

When an input edge or level is detected on an I/O line, the corresponding bit in the Interrupt Status Register (PIO_ISR) is set. If the corresponding bit in PIO_IMR is set, the PIO Controller interrupt line is asserted.The interrupt signals of the 32 channels are ORed-wired together to generate a single interrupt signal to the interrupt controller.

When the software reads PIO_ISR, all the interrupts are automatically cleared. This signifies that all the interrupts that are pending when PIO_ISR is read must be handled. When an Interrupt is enabled on a “level”, the interrupt is generated as long as the interrupt source is not cleared, even if some read accesses in PIO_ISR are performed.

Figure 33-6. Event Detector on Input Lines (Figure Represents Line 0)



Example of interrupt generation on following lines:

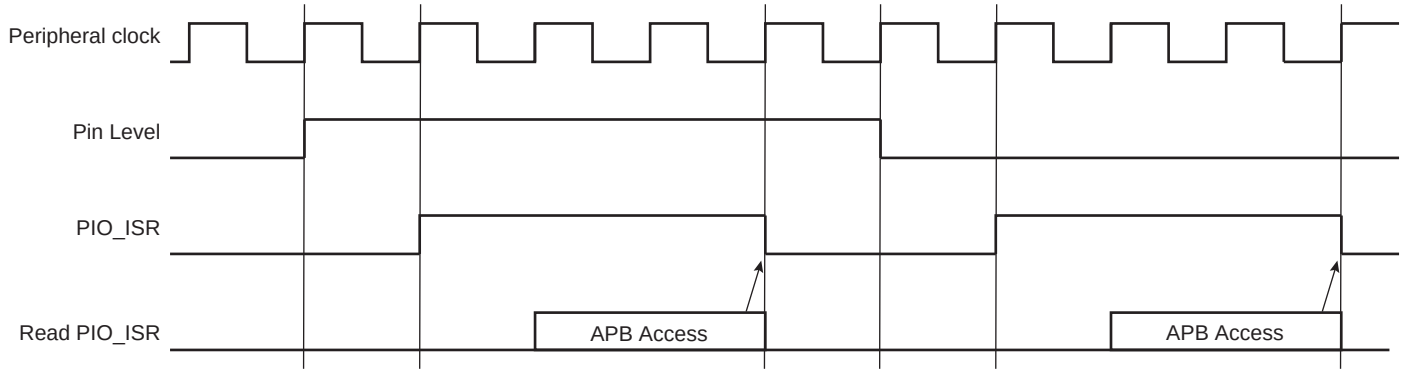
- Rising edge on PIO line 0
- Falling edge on PIO line 1
- Rising edge on PIO line 2
- Low-level on PIO line 3
- High-level on PIO line 4
- High-level on PIO line 5
- Falling edge on PIO line 6
- Rising edge on PIO line 7
- Any edge on the other lines

Table 33-3 provides the required configuration for this example.

Table 33-3. Configuration for Example Interrupt Generation

Configuration	Description
Interrupt Mode	All the interrupt sources are enabled by writing 32'hFFFF_FFFF in PIO_IER. Then the additional interrupt mode is enabled for lines 0 to 7 by writing 32'h0000_00FF in PIO_AIMER.
Edge or Level Detection	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.
Falling/Rising Edge or Low/High-Level Detection	Lines 0, 2, 4, 5 and 7 are configured in rising edge or high-level detection by writing 32'h0000_00B5 in PIO_REHLSR. The other lines are configured in falling edge or low-level detection by default if they have not been previously configured. Otherwise, lines 1, 3 and 6 must be configured in falling edge/low-level detection by writing 32'h0000_004A in PIO_FELLSR.

Figure 33-7. Input Change Interrupt Timings When No Additional Interrupt Modes



33.5.11 I/O Lines Lock

When an I/O line is controlled by a peripheral (particularly the Pulse Width Modulation Controller PWM), it can become locked by the action of this peripheral via an input of the PIO Controller. When an I/O line is locked, the write of the corresponding bit in PIO_PER, PIO_PDR, PIO_MDER, PIO_MDDR, PIO_PUDR, PIO_PUER, PIO_ABCDSR1 and PIO_ABCDSR2 is discarded in order to lock its configuration. The user can know at anytime which I/O line is locked by reading the PIO Lock Status Register (PIO_LOCKSR). Once an I/O line is locked, the only way to unlock it is to apply a hardware reset to the PIO Controller.

33.5.12 Programmable I/O Delays

The PIO interface consists of a series of signals driven by peripherals or directly by software. The simultaneous switching outputs on these busses may lead to a peak of current in the internal and external power supply lines.

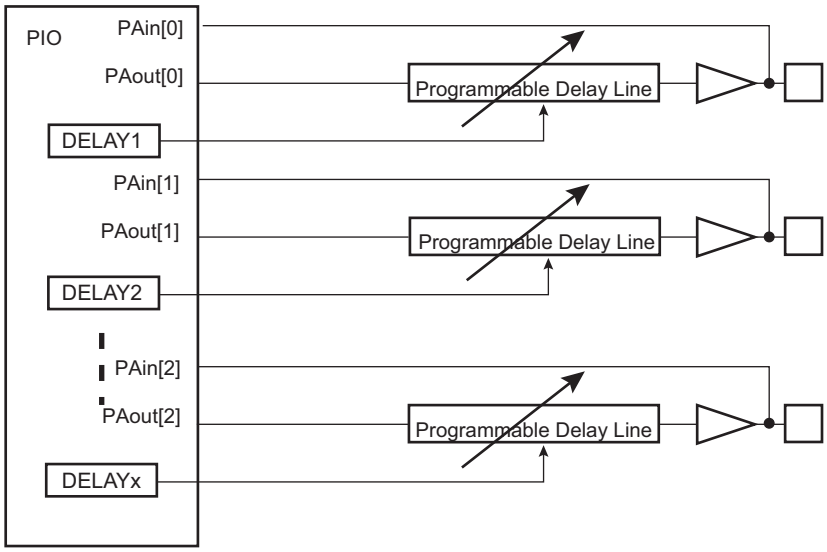
In order to reduce the current peak in such cases, additional propagation delays can be adjusted independently for pad buffers by means of configuration registers, PIO_DELAYR.

For each I/O supporting the additional programmable delay, the delay ranges from 0 to - ns (worst case process, voltage, temperature). The delay can differ between I/Os supporting this feature. Delay can be modified per programming for each I/O. The minimal additional delay that can be programmed on a PAD supporting this feature is 1/16 of the maximum programmable delay.

Only pads PA26-PA27-PA30-PA31 can be configured.

When programming 0x0 in fields, no delay is added (reset value) and the propagation delay of the pad buffers is the inherent delay of the pad buffer. When programming 0xF in fields, the propagation delay of the corresponding pad is maximal.

Figure 33-8. Programmable I/O Delays



33.5.13 Programmable Schmitt Trigger

It is possible to configure each input for the Schmitt trigger. By default the Schmitt trigger is active. Disabling the Schmitt trigger is requested when using the QTouch® Library.

33.5.14 Parallel Capture Mode

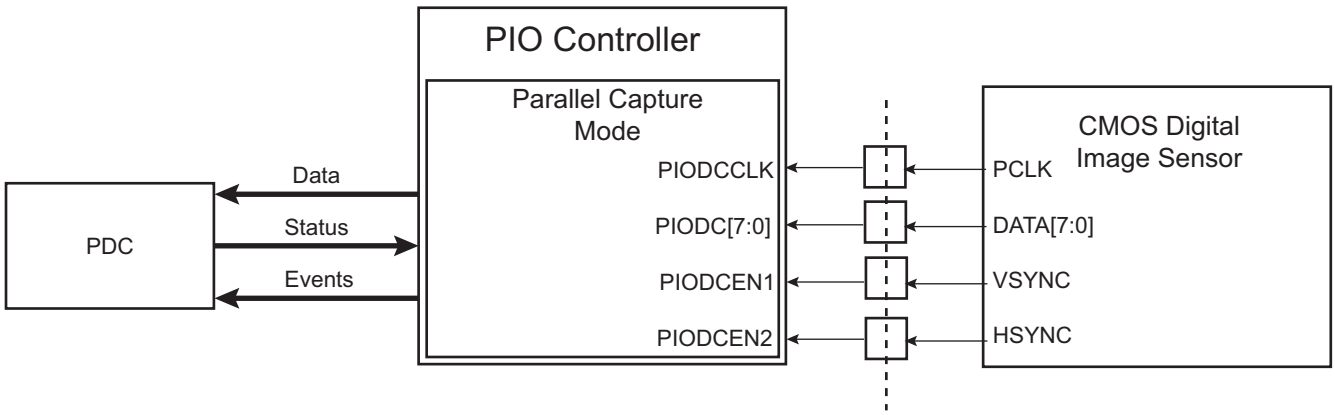
33.5.14.1 Overview

The PIO Controller integrates an interface able to read data from a CMOS digital image sensor, a high-speed parallel ADC, a DSP synchronous port in synchronous mode, etc. For better understanding and to ease reading, the following description uses an example with a CMOS digital image sensor.

33.5.14.2 Functional Description

The CMOS digital image sensor provides a sensor clock, an 8-bit data synchronous with the sensor clock and two data enables which are also synchronous with the sensor clock.

Figure 33-9. PIO Controller Connection with CMOS Digital Image Sensor



As soon as the parallel capture mode is enabled by writing a one to the PCEN bit in PIO_PCMR, the I/O lines connected to the sensor clock (PIODCCLK), the sensor data (PIODC[7:0]) and the sensor data enable signals (PIODCEN1 and PIODCEN2) are configured automatically as inputs. To know which I/O lines are associated with

the sensor clock, the sensor data and the sensor data enable signals, refer to the I/O multiplexing table(s) in the section “Package and Pinout”.

Once enabled, the parallel capture mode samples the data at rising edge of the sensor clock and resynchronizes it with the peripheral clock domain.

The size of the data which can be read in PIO_PCRHR can be programmed using the DSIZE field in PIO_PCMR. If this data size is larger than 8 bits, then the parallel capture mode samples several sensor data to form a concatenated data of size defined by DSIZE. Then this data is stored in PIO_PCRHR and the flag DRDY is set to one in PIO_PCISR.

The parallel capture mode can be associated with a reception channel of the Peripheral DMA Controller (PDC). This performs reception transfer from parallel capture mode to a memory buffer without any intervention from the CPU. Transfer status signals from PDC are available in PIO_PCISR through the flags ENDRX and RXBUFF.

The parallel capture mode can take into account the sensor data enable signals or not. If the bit ALWYS is set to zero in PIO_PCMR, the parallel capture mode samples the sensor data at the rising edge of the sensor clock only if both data enable signals are active (at one). If the bit ALWYS is set to one, the parallel capture mode samples the sensor data at the rising edge of the sensor clock whichever the data enable signals are.

The parallel capture mode can sample the sensor data only one time out of two. This is particularly useful when the user wants only to sample the luminance Y of a CMOS digital image sensor which outputs a YUV422 data stream. If the HALFS bit is set to zero in PIO_PCMR, the parallel capture mode samples the sensor data in the conditions described above. If the HALFS bit is set to one in PIO_PCMR, the parallel capture mode samples the sensor data in the conditions described above, but only one time out of two. Depending on the FRSTS bit in PIO_PCMR, the sensor can either sample the even or odd sensor data. If sensor data are numbered in the order that they are received with an index from zero to n, if FRSTS equals zero then only data with an even index are sampled. If FRSTS equals one, then only data with an odd index are sampled. If data is ready in PIO_PCRHR and it is not read before a new data is stored in PIO_PCRHR, then an overrun error occurs. The previous data is lost and the OVRE flag in PIO_PCISR is set to one. This flag is automatically reset when PIO_PCISR is read (reset after read).

The flags DRDY, OVRE, ENDRX and RXBUFF can be a source of the PIO interrupt.

Figure 33-10. Parallel Capture Mode Waveforms (DSIZE = 2, ALWYS = 0, HALFS = 0)

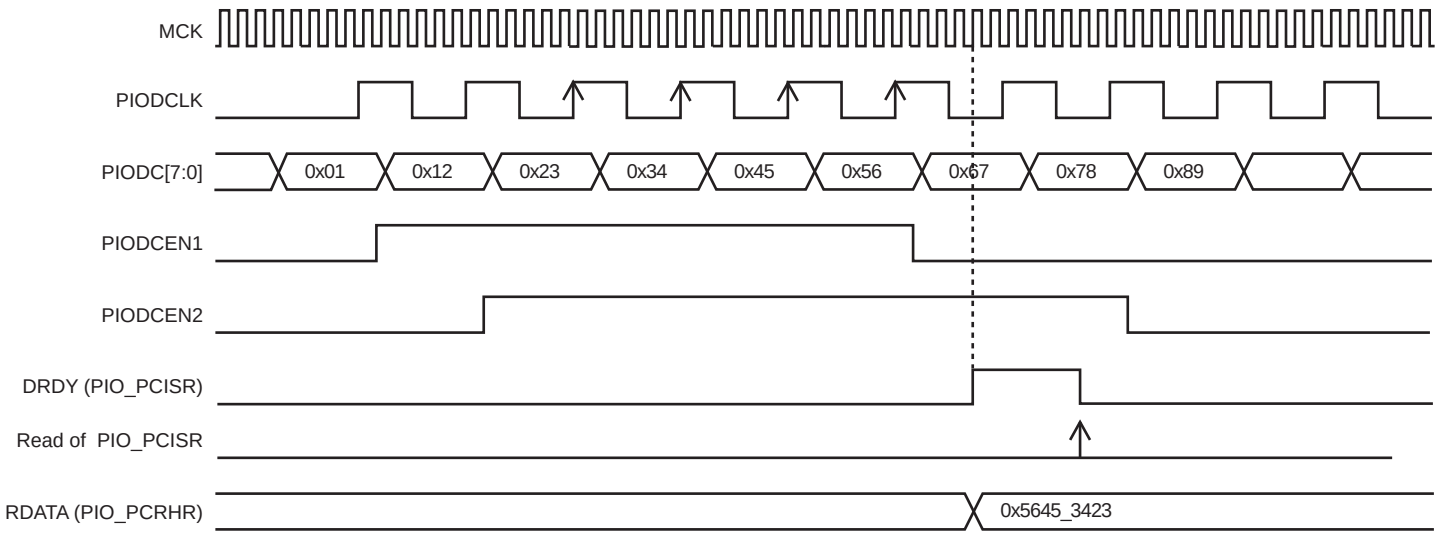


Figure 33-11. Parallel Capture Mode Waveforms (DSIZE = 2, ALWYS = 1, HALFS = 0)

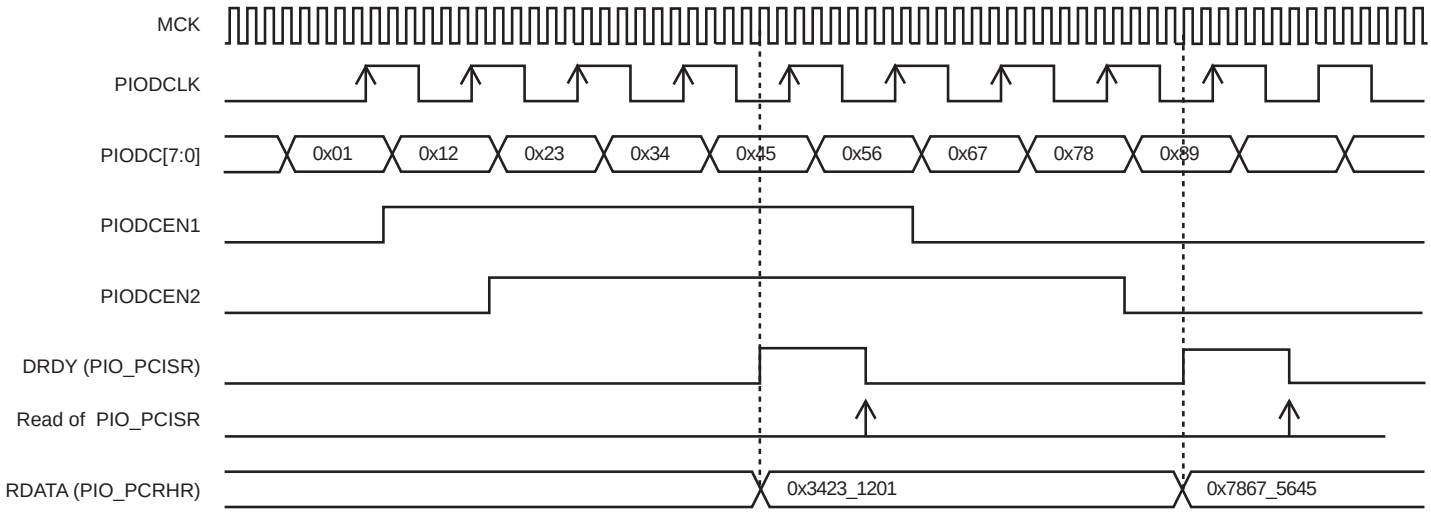


Figure 33-12. Parallel Capture Mode Waveforms (DSIZE = 2, ALWYS = 0, HALFS = 1, FRSTS = 0)

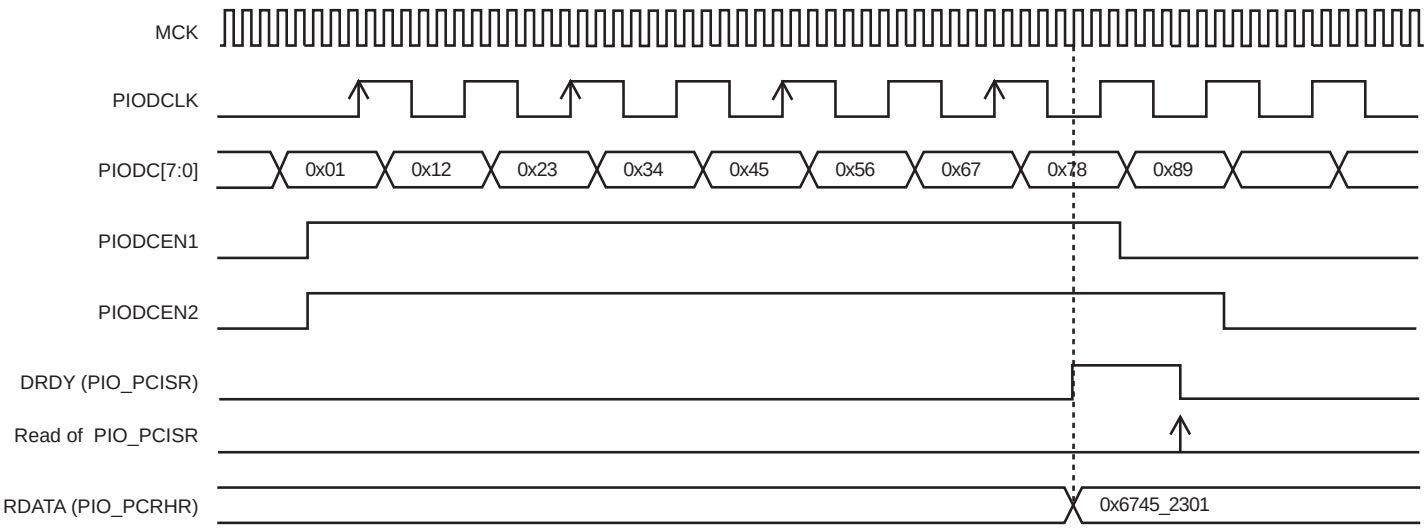
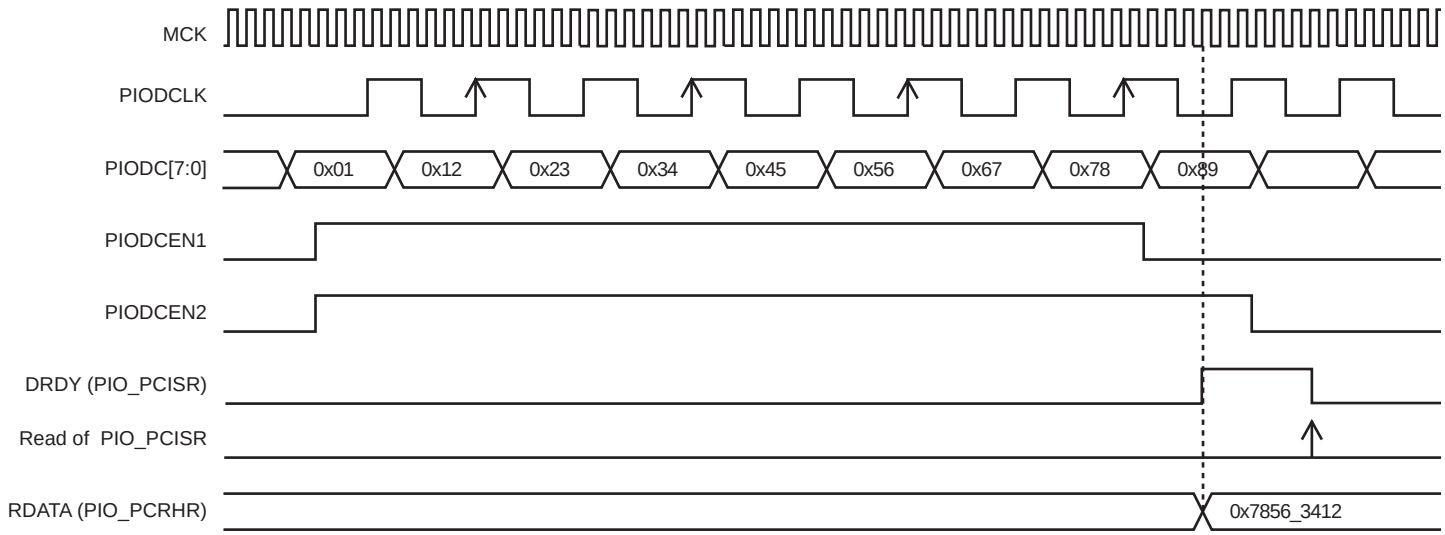


Figure 33-13. Parallel Capture Mode Waveforms (DSIZE = 2, ALWYS = 0, HALFS = 1, FRSTS = 1)



33.5.14.3 Restrictions

- Configuration fields DSIZE, ALWYS, HALFS and FRSTS in PIO_PCMR can be changed **ONLY** if the parallel capture mode is disabled at this time (PCEN = 0 in PIO_PCMR).
- The frequency of peripheral clock must be strictly superior to two times the frequency of the clock of the device which generates the parallel data.

33.5.14.4 Programming Sequence

Without PDC

1. Write PIO_PCIDR and PIO_PCIER in order to configure the parallel capture mode interrupt mask.
2. Write PIO_PCMR to set the fields DSIZE, ALWYS, HALFS and FRSTS in order to configure the parallel capture mode **WITHOUT** enabling the parallel capture mode.
3. Write PIO_PCMR to set the PCEN bit to one in order to enable the parallel capture mode **WITHOUT** changing the previous configuration.
4. Wait for a data ready by polling the DRDY flag in PIO_PCISR or by waiting for the corresponding interrupt.
5. Check OVRE flag in PIO_PCISR.
6. Read the data in PIO_PCRHR.
7. If new data are expected, go to step 4.
8. Write PIO_PCMR to set the PCEN bit to zero in order to disable the parallel capture mode **WITHOUT** changing the previous configuration.

With PDC

1. Write PIO_PCIDR and PIO_PCIER in order to configure the parallel capture mode interrupt mask.
2. Configure PDC transfer in PDC registers.
3. Write PIO_PCMR to set the fields DSIZE, ALWYS, HALFS and FRSTS in order to configure the parallel capture mode **WITHOUT** enabling the parallel capture mode.
4. Write PIO_PCMR to set PCEN bit to one in order to enable the parallel capture mode **WITHOUT** changing the previous configuration.
5. Wait for end of transfer by waiting for the interrupt corresponding to the flag ENDRX in PIO_PCISR.
6. Check OVRE flag in PIO_PCISR.
7. If a new buffer transfer is expected, go to step 5.
8. Write PIO_PCMR to set the PCEN bit to zero in order to disable the parallel capture mode **WITHOUT** changing the previous configuration.

33.5.15 I/O Lines Programming Example

The programming example shown in Table 33-4 is used to obtain the following configuration:

- 4-bit output port on I/O lines 0 to 3 (should be written in a single write operation), open-drain, with pull-up resistor
- Four output signals on I/O lines 4 to 7 (to drive LEDs for example), driven high and low, no pull-up resistor, no pull-down resistor
- Four input signals on I/O lines 8 to 11 (to read push-button states for example), with pull-up resistors, glitch filters and input change interrupts
- Four input signals on I/O line 12 to 15 to read an external device status (polled, thus no input change interrupt), no pull-up resistor, no glitch filter
- I/O lines 16 to 19 assigned to peripheral A functions with pull-up resistor
- I/O lines 20 to 23 assigned to peripheral B functions with pull-down resistor
- I/O lines 24 to 27 assigned to peripheral C with input change interrupt, no pull-up resistor and no pull-down resistor
- I/O lines 28 to 31 assigned to peripheral D, no pull-up resistor and no pull-down resistor

Table 33-4. 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

Table 33-4. Programming Example (Continued)

PIO_PUDR	0xFFF0_00F0
PIO_PUER	0x000F_FF0F
PIO_PPDDR	0xFF0F_FFFF
PIO_PPDER	0x00F0_0000
PIO_ABCDSR1	0xF0F0_0000
PIO_ABCDSR2	0xFF00_0000
PIO_OWER	0x0000_000F
PIO_OWDR	0x0FFF_ FFF0

33.5.16 Register Write Protection

To prevent any single software error from corrupting PIO behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the PIO Write Protection Mode Register (PIO_WPMR).

If a write access to a write-protected register is detected, the WPVS flag in the PIO Write Protection Status Register (PIO_WPSR) is set and the field WPVSRC indicates the register in which the write access has been attempted.

The WPVS bit is automatically cleared after reading the PIO_WPSR.

The following registers can be write-protected:

- PIO Enable Register
- PIO Disable Register
- PIO Output Enable Register
- PIO Output Disable Register
- PIO Input Filter Enable Register
- PIO Input Filter Disable Register
- PIO Multi-driver Enable Register
- PIO Multi-driver Disable Register
- PIO Pull-Up Disable Register
- PIO Pull-Up Enable Register
- PIO Peripheral ABCD Select Register 1
- PIO Peripheral ABCD Select Register 2
- PIO Output Write Enable Register
- PIO Output Write Disable Register
- PIO Pad Pull-Down Disable Register
- PIO Pad Pull-Down Enable Register
- PIO Parallel Capture Mode Register

33.6 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-bit 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 one systematically.

Table 33-5. 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	0x00000000
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	0x00000000
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	(1)
0x006C	Reserved	–	–	–

Table 33-5. Register Mapping (Continued)

Offset	Register	Name	Access	Reset
0x0070	Peripheral Select Register 1	PIO_ABCDSR1	Read/Write	0x00000000
0x0074	Peripheral Select Register 2	PIO_ABCDSR2	Read/Write	0x00000000
0x0078–0x007C	Reserved	–	–	–
0x0080	Input Filter Slow Clock Disable Register	PIO_IFSCDR	Write-only	–
0x0084	Input Filter Slow Clock Enable Register	PIO_IFSCER	Write-only	–
0x0088	Input Filter Slow Clock Status Register	PIO_IFSCSR	Read-only	0x00000000
0x008C	Slow Clock Divider Debouncing Register	PIO_SCDR	Read/Write	0x00000000
0x0090	Pad Pull-down Disable Register	PIO_PPDDR	Write-only	–
0x0094	Pad Pull-down Enable Register	PIO_PPDER	Write-only	–
0x0098	Pad Pull-down Status Register	PIO_PPDSR	Read-only	(1)
0x009C	Reserved	–	–	–
0x00A0	Output Write Enable	PIO_OWER	Write-only	–
0x00A4	Output Write Disable	PIO_OWDR	Write-only	–
0x00A8	Output Write Status Register	PIO_OWSR	Read-only	0x00000000
0x00AC	Reserved	–	–	–
0x00B0	Additional Interrupt Modes Enable Register	PIO_AIMER	Write-only	–
0x00B4	Additional Interrupt Modes Disable 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 Protection Mode Register	PIO_WPMR	Read/Write	0x00000000
0x00E8	Write Protection Status Register	PIO_WPSR	Read-only	0x00000000
0x00EC–0x00FC	Reserved	–	–	–
0x0100	Schmitt Trigger Register	PIO_SCHMITT	Read/Write	0x00000000
0x0104–0x010C	Reserved	–	–	–
0x0110	I/O Delay Register	PIO_DELAYR	Read/Write	0x00000000
0x0114–0x011C	Reserved	–	–	–
0x0120–0x014C	Reserved	–	–	–
0x0150	Parallel Capture Mode Register	PIO_PCMR	Read/Write	0x00000000

Table 33-5. Register Mapping (Continued)

Offset	Register	Name	Access	Reset
0x0154	Parallel Capture Interrupt Enable Register	PIO_PCIER	Write-only	–
0x0158	Parallel Capture Interrupt Disable Register	PIO_PCIDR	Write-only	–
0x015C	Parallel Capture Interrupt Mask Register	PIO_PCIMR	Read-only	0x00000000
0x0160	Parallel Capture Interrupt Status Register	PIO_PCISR	Read-only	0x00000000
0x0164	Parallel Capture Reception Holding Register	PIO_PCRHR	Read-only	0x00000000
0x0168–0x018C	Reserved for PDC Registers	–	–	–

Notes:
1. Reset value depends on the product implementation.
2. PIO_ODSR is Read-only or Read/Write depending on PIO_OWSR I/O lines.
3. Reset value of PIO_PDSR depends on the level of the I/O lines. Reading the I/O line levels requires the clock of the PIO Controller to be enabled, otherwise PIO_PDSR reads the levels present on the I/O line at the time the clock was disabled.
4. PIO_ISR is reset at 0x0. However, the first read of the register may read a different value as input changes may have occurred.
5. If an offset is not listed in the table it must be considered as reserved.

33.6.1 PIO Enable Register

Name: PIO_PER

Address: 0x400E0E00 (PIOA), 0x400E1000 (PIOB), 0x400E1200 (PIOC), 0x400E1400 (PIOD), 0x400E1600 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **P0–P31: PIO Enable**

0: No effect.

1: Enables the PIO to control the corresponding pin (disables peripheral control of the pin).

33.6.2 PIO Disable Register

Name: PIO_PDR

Address: 0x400E0E04 (PIOA), 0x400E1004 (PIOB), 0x400E1204 (PIOC), 0x400E1404 (PIOD), 0x400E1604 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **P0–P31: PIO Disable**

0: No effect.

1: Disables the PIO from controlling the corresponding pin (enables peripheral control of the pin).

33.6.3 PIO Status Register

Name: PIO_PSR

Address: 0x400E0E08 (PIOA), 0x400E1008 (PIOB), 0x400E1208 (PIOC), 0x400E1408 (PIOD), 0x400E1608 (PIOE)

Access: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: PIO Status**

0: PIO is inactive on the corresponding I/O line (peripheral is active).

1: PIO is active on the corresponding I/O line (peripheral is inactive).

33.6.4 PIO Output Enable Register

Name: PIO_OER

Address: 0x400E0E10 (PIOA), 0x400E1010 (PIOB), 0x400E1210 (PIOC), 0x400E1410 (PIOD), 0x400E1610 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **P0–P31: Output Enable**

0: No effect.

1: Enables the output on the I/O line.

33.6.5 PIO Output Disable Register

Name: PIO_ODR

Address: 0x400E0E14 (PIOA), 0x400E1014 (PIOB), 0x400E1214 (PIOC), 0x400E1414 (PIOD), 0x400E1614 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **P0–P31: Output Disable**

0: No effect.

1: Disables the output on the I/O line.

33.6.6 PIO Output Status Register

Name: PIO_OSR

Address: 0x400E0E18 (PIOA), 0x400E1018 (PIOB), 0x400E1218 (PIOC), 0x400E1418 (PIOD), 0x400E1618 (PIOE)

Access: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Output Status**

0: The I/O line is a pure input.

1: The I/O line is enabled in output.

33.6.7 PIO Input Filter Enable Register

Name: PIO_IFER

Address: 0x400E0E20 (PIOA), 0x400E1020 (PIOB), 0x400E1220 (PIOC), 0x400E1420 (PIOD), 0x400E1620 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **P0–P31: Input Filter Enable**

0: No effect.

1: Enables the input glitch filter on the I/O line.

33.6.8 PIO Input Filter Disable Register

Name: PIO_IFDR

Address: 0x400E0E24 (PIOA), 0x400E1024 (PIOB), 0x400E1224 (PIOC), 0x400E1424 (PIOD), 0x400E1624 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **P0–P31: Input Filter Disable**

0: No effect.

1: Disables the input glitch filter on the I/O line.

33.6.9 PIO Input Filter Status Register

Name: PIO_IFSR

Address: 0x400E0E28 (PIOA), 0x400E1028 (PIOB), 0x400E1228 (PIOC), 0x400E1428 (PIOD), 0x400E1628 (PIOE)

Access: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Input Filter Status**

0: The input glitch filter is disabled on the I/O line.

1: The input glitch filter is enabled on the I/O line.

33.6.10 PIO Set Output Data Register

Name: PIO_SODR

Address: 0x400E0E30 (PIOA), 0x400E1030 (PIOB), 0x400E1230 (PIOC), 0x400E1430 (PIOD), 0x400E1630 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Set Output Data**

0: No effect.

1: Sets the data to be driven on the I/O line.

33.6.11 PIO Clear Output Data Register

Name: PIO_CODR

Address: 0x400E0E34 (PIOA), 0x400E1034 (PIOB), 0x400E1234 (PIOC), 0x400E1434 (PIOD), 0x400E1634 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Clear Output Data**

0: No effect.

1: Clears the data to be driven on the I/O line.

33.6.12 PIO Output Data Status Register

Name: PIO_ODSR

Address: 0x400E0E38 (PIOA), 0x400E1038 (PIOB), 0x400E1238 (PIOC), 0x400E1438 (PIOD), 0x400E1638 (PIOE)

Access: Read-only or Read/Write

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Output Data Status**

0: The data to be driven on the I/O line is 0.

1: The data to be driven on the I/O line is 1.

33.6.13 PIO Pin Data Status Register

Name: PIO_PDSR

Address: 0x400E0E3C (PIOA), 0x400E103C (PIOB), 0x400E123C (PIOC), 0x400E143C (PIOD), 0x400E163C (PIOE)

Access: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Output Data Status**

0: The I/O line is at level 0.

1: The I/O line is at level 1.

33.6.14 PIO Interrupt Enable Register

Name: PIO_IER

Address: 0x400E0E40 (PIOA), 0x400E1040 (PIOB), 0x400E1240 (PIOC), 0x400E1440 (PIOD), 0x400E1640 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Input Change Interrupt Enable**

0: No effect.

1: Enables the input change interrupt on the I/O line.

33.6.15 PIO Interrupt Disable Register

Name: PIO_IDR

Address: 0x400E0E44 (PIOA), 0x400E1044 (PIOB), 0x400E1244 (PIOC), 0x400E1444 (PIOD), 0x400E1644 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Input Change Interrupt Disable**

0: No effect.

1: Disables the input change interrupt on the I/O line.

33.6.16 PIO Interrupt Mask Register

Name: PIO_IMR

Address: 0x400E0E48 (PIOA), 0x400E1048 (PIOB), 0x400E1248 (PIOC), 0x400E1448 (PIOD), 0x400E1648 (PIOE)

Access: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Input Change Interrupt Mask**

0: Input change interrupt is disabled on the I/O line.

1: Input change interrupt is enabled on the I/O line.

33.6.17 PIO Interrupt Status Register

Name: PIO_ISR

Address: 0x400E0E4C (PIOA), 0x400E104C (PIOB), 0x400E124C (PIOC), 0x400E144C (PIOD), 0x400E164C (PIOE)

Access: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Input Change Interrupt Status**

0: No input change has been detected on the I/O line since PIO_ISR was last read or since reset.

1: At least one input change has been detected on the I/O line since PIO_ISR was last read or since reset.

33.6.18 PIO Multi-driver Enable Register

Name: PIO_MDER

Address: 0x400E0E50 (PIOA), 0x400E1050 (PIOB), 0x400E1250 (PIOC), 0x400E1450 (PIOD), 0x400E1650 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **P0-P31: Multi-drive Enable**

0: No effect.

1: Enables multi-drive on the I/O line.

33.6.19 PIO Multi-driver Disable Register

Name: PIO_MDDR

Address: 0x400E0E54 (PIOA), 0x400E1054 (PIOB), 0x400E1254 (PIOC), 0x400E1454 (PIOD), 0x400E1654 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **P0–P31: Multi-drive Disable**

0: No effect.

1: Disables multi-drive on the I/O line.

33.6.20 PIO Multi-driver Status Register

Name: PIO_MDSR

Address: 0x400E0E58 (PIOA), 0x400E1058 (PIOB), 0x400E1258 (PIOC), 0x400E1458 (PIOD), 0x400E1658 (PIOE)

Access: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Multi-drive Status**

0: The multi-drive is disabled on the I/O line. The pin is driven at high- and low-level.

1: The multi-drive is enabled on the I/O line. The pin is driven at low-level only.

33.6.21 PIO Pull-Up Disable Register

Name: PIO_PUDR

Address: 0x400E0E60 (PIOA), 0x400E1060 (PIOB), 0x400E1260 (PIOC), 0x400E1460 (PIOD), 0x400E1660 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **P0–P31: Pull-Up Disable**

0: No effect.

1: Disables the pull-up resistor on the I/O line.

33.6.22 PIO Pull-Up Enable Register

Name: PIO_PUER

Address: 0x400E0E64 (PIOA), 0x400E1064 (PIOB), 0x400E1264 (PIOC), 0x400E1464 (PIOD), 0x400E1664 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **P0–P31: Pull-Up Enable**

0: No effect.

1: Enables the pull-up resistor on the I/O line.

33.6.23 PIO Pull-Up Status Register

Name: PIO_PUSR

Address: 0x400E0E68 (PIOA), 0x400E1068 (PIOB), 0x400E1268 (PIOC), 0x400E1468 (PIOD), 0x400E1668 (PIOE)

Access: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Pull-Up Status**

0: Pull-up resistor is enabled on the I/O line.

1: Pull-up resistor is disabled on the I/O line.

33.6.24 PIO Peripheral ABCD Select Register 1

Name: PIO_ABCDSR1

Access: Read/Write

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **P0–P31: Peripheral Select**

If the same bit is set to 0 in PIO_ABCDSR2:

0: Assigns the I/O line to the Peripheral A function.

1: Assigns the I/O line to the Peripheral B function.

If the same bit is set to 1 in PIO_ABCDSR2:

0: Assigns the I/O line to the Peripheral C function.

1: Assigns the I/O line to the Peripheral D function.

33.6.25 PIO Peripheral ABCD Select Register 2

Name: PIO_ABCDSR2

Address: 0x400E0E70 (PIOA), 0x400E1070 (PIOB), 0x400E1270 (PIOC), 0x400E1470 (PIOD), 0x400E1670 (PIOE)

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 the PIO Write Protection Mode Register.

- **P0–P31: Peripheral Select**

If the same bit is set to 0 in PIO_ABCDSR1:

0: Assigns the I/O line to the Peripheral A function.

1: Assigns the I/O line to the Peripheral C function.

If the same bit is set to 1 in PIO_ABCDSR1:

0: Assigns the I/O line to the Peripheral B function.

1: Assigns the I/O line to the Peripheral D function.

33.6.26 PIO Input Filter Slow Clock Disable Register

Name: PIO_IFSCDR

Address: 0x400E0E80 (PIOA), 0x400E1080 (PIOB), 0x400E1280 (PIOC), 0x400E1480 (PIOD), 0x400E1680 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Peripheral Clock Glitch Filtering Select**

0: No effect.

1: The glitch filter is able to filter glitches with a duration < $t_{peripheral\ clock}/2$.

33.6.27 PIO Input Filter Slow Clock Enable Register

Name: PIO_IFSCER

Address: 0x400E0E84 (PIOA), 0x400E1084 (PIOB), 0x400E1284 (PIOC), 0x400E1484 (PIOD), 0x400E1684 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Slow Clock Debouncing Filtering Select**

0: No effect.

1: The debouncing filter is able to filter pulses with a duration < $t_{div\_slck}/2$.

33.6.28 PIO Input Filter Slow Clock Status Register

Name: PIO_IFSCSR

Address: 0x400E0E88 (PIOA), 0x400E1088 (PIOB), 0x400E1288 (PIOC), 0x400E1488 (PIOD), 0x400E1688 (PIOE)

Access: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Glitch or Debouncing Filter Selection Status**

0: The glitch filter is able to filter glitches with a duration < $t_{peripheral\ clock}/2$.

1: The debouncing filter is able to filter pulses with a duration < $t_{div\_slck}/2$.

33.6.29 PIO Slow Clock Divider Debouncing Register

Name: PIO_SCDR

Address: 0x400E0E8C (PIOA), 0x400E108C (PIOB), 0x400E128C (PIOC), 0x400E148C (PIOD), 0x400E168C (PIOE)

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	DIV					

7	6	5	4	3	2	1	0
DIV							

- **DIV: Slow Clock Divider Selection for Debouncing**

$t_{div\_slck} = ((DIV + 1) \times 2) \times t_{slck}$

33.6.30 PIO Pad Pull-Down Disable Register

Name: PIO_PPDDR

Address: 0x400E0E90 (PIOA), 0x400E1090 (PIOB), 0x400E1290 (PIOC), 0x400E1490 (PIOD), 0x400E1690 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **P0–P31: Pull-Down Disable**

0: No effect.

1: Disables the pull-down resistor on the I/O line.

33.6.31 PIO Pad Pull-Down Enable Register

Name: PIO_PPDER

Address: 0x400E0E94 (PIOA), 0x400E1094 (PIOB), 0x400E1294 (PIOC), 0x400E1494 (PIOD), 0x400E1694 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **P0–P31: Pull-Down Enable**

0: No effect.

1: Enables the pull-down resistor on the I/O line.

33.6.32 PIO Pad Pull-Down Status Register

Name: PIO_PPDSR

Address: 0x400E0E98 (PIOA), 0x400E1098 (PIOB), 0x400E1298 (PIOC), 0x400E1498 (PIOD), 0x400E1698 (PIOE)

Access: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Pull-Down Status**

0: Pull-down resistor is enabled on the I/O line.

1: Pull-down resistor is disabled on the I/O line.

33.6.33 PIO Output Write Enable Register

Name: PIO_OWER

Address: 0x400E0EA0 (PIOA), 0x400E10A0 (PIOB), 0x400E12A0 (PIOC), 0x400E14A0 (PIOD), 0x400E16A0 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **P0–P31: Output Write Enable**

0: No effect.

1: Enables writing PIO_ODSR for the I/O line.

33.6.34 PIO Output Write Disable Register

Name: PIO_OWDR

Address: 0x400E0EA4 (PIOA), 0x400E10A4 (PIOB), 0x400E12A4 (PIOC), 0x400E14A4 (PIOD), 0x400E16A4 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **P0–P31: Output Write Disable**

0: No effect.

1: Disables writing PIO_ODSR for the I/O line.

33.6.35 PIO Output Write Status Register

Name: PIO_OWSR

Address: 0x400E0EA8 (PIOA), 0x400E10A8 (PIOB), 0x400E12A8 (PIOC), 0x400E14A8 (PIOD), 0x400E16A8 (PIOE)

Access: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Output Write Status**

0: Writing PIO_ODSR does not affect the I/O line.

1: Writing PIO_ODSR affects the I/O line.

33.6.36 PIO Additional Interrupt Modes Enable Register

Name: PIO_AIMER

Address: 0x400E0EB0 (PIOA), 0x400E10B0 (PIOB), 0x400E12B0 (PIOC), 0x400E14B0 (PIOD), 0x400E16B0 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Additional Interrupt Modes Enable**

0: No effect.

1: The interrupt source is the event described in PIO_ELSR and PIO_FRLHSR.

33.6.37 PIO Additional Interrupt Modes Disable Register

Name: PIO_AIMDR

Address: 0x400E0EB4 (PIOA), 0x400E10B4 (PIOB), 0x400E12B4 (PIOC), 0x400E14B4 (PIOD), 0x400E16B4 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Additional Interrupt Modes Disable**

0: No effect.

1: The interrupt mode is set to the default interrupt mode (both-edge detection).

33.6.38 PIO Additional Interrupt Modes Mask Register

Name: PIO_AIMMR

Address: 0x400E0EB8 (PIOA), 0x400E10B8 (PIOB), 0x400E12B8 (PIOC), 0x400E14B8 (PIOD), 0x400E16B8 (PIOE)

Access: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: IO Line Index**

Selects the IO event type triggering an interrupt.

0: The interrupt source is a both-edge detection event.

1: The interrupt source is described by the registers PIO_ELSR and PIO_FRLHSR.

33.6.39 PIO Edge Select Register

Name: PIO_ESR

Address: 0x400E0EC0 (PIOA), 0x400E10C0 (PIOB), 0x400E12C0 (PIOC), 0x400E14C0 (PIOD), 0x400E16C0 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Edge Interrupt Selection**

0: No effect.

1: The interrupt source is an edge-detection event.

33.6.40 PIO Level Select Register

Name: PIO_LSR

Address: 0x400E0EC4 (PIOA), 0x400E10C4 (PIOB), 0x400E12C4 (PIOC), 0x400E14C4 (PIOD), 0x400E16C4 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Level Interrupt Selection**

0: No effect.

1: The interrupt source is a level-detection event.

33.6.41 PIO Edge/Level Status Register

Name: PIO_ELSR

Address: 0x400E0EC8 (PIOA), 0x400E10C8 (PIOB), 0x400E12C8 (PIOC), 0x400E14C8 (PIOD), 0x400E16C8 (PIOE)

Access: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Edge/Level Interrupt Source Selection**

0: The interrupt source is an edge-detection event.

1: The interrupt source is a level-detection event.

33.6.42 PIO Falling Edge/Low-Level Select Register

Name: PIO_FELLSR

Address: 0x400E0ED0 (PIOA), 0x400E10D0 (PIOB), 0x400E12D0 (PIOC), 0x400E14D0 (PIOD), 0x400E16D0 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Falling Edge/Low-Level Interrupt Selection**

0: No effect.

1: The interrupt source is set to a falling edge detection or low-level detection event, depending on PIO_ELSR.

33.6.43 PIO Rising Edge/High-Level Select Register

Name: PIO_REHLSR

Address: 0x400E0ED4 (PIOA), 0x400E10D4 (PIOB), 0x400E12D4 (PIOC), 0x400E14D4 (PIOD), 0x400E16D4 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Rising Edge/High-Level Interrupt Selection**

0: No effect.

1: The interrupt source is set to a rising edge detection or high-level detection event, depending on PIO_ELSR.

33.6.44 PIO Fall/Rise - Low/High Status Register

Name: PIO_FRLHSR

Address: 0x400E0ED8 (PIOA), 0x400E10D8 (PIOB), 0x400E12D8 (PIOC), 0x400E14D8 (PIOD), 0x400E16D8 (PIOE)

Access: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Edge/Level Interrupt Source Selection**

0: The interrupt source is a falling edge detection (if PIO_ELSR = 0) or low-level detection event (if PIO_ELSR = 1).

1: The interrupt source is a rising edge detection (if PIO_ELSR = 0) or high-level detection event (if PIO_ELSR = 1).

33.6.45 PIO Lock Status Register

Name: PIO_LOCKSR

Address: 0x400E0EE0 (PIOA), 0x400E10E0 (PIOB), 0x400E12E0 (PIOC), 0x400E14E0 (PIOD), 0x400E16E0 (PIOE)

Access: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Lock Status**

0: The I/O line is not locked.

1: The I/O line is locked.

33.6.46 PIO Write Protection Mode Register

Name: PIO_WPMR

Address: 0x400E0EE4 (PIOA), 0x400E10E4 (PIOB), 0x400E12E4 (PIOC), 0x400E14E4 (PIOD), 0x400E16E4 (PIOE)

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 0x50494F (“PIO” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x50494F (“PIO” in ASCII).

See Section 33.5.16 “Register Write Protection” for the list of registers that can be protected.

- **WPKEY: Write Protection Key**

Value	Name	Description
0x50494F	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

33.6.47 PIO Write Protection Status Register

Name: PIO_WPSR

Address: 0x400E0EE8 (PIOA), 0x400E10E8 (PIOB), 0x400E12E8 (PIOC), 0x400E14E8 (PIOD), 0x400E16E8 (PIOE)

Access: Read-only

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

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of the PIO_WPSR.

1: A write protection violation has occurred since the last read of the PIO_WPSR. 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**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

33.6.48 PIO Schmitt Trigger Register

Name: PIO_SCHMITT

Address: 0x400E0F00 (PIOA), 0x400E1100 (PIOB), 0x400E1300 (PIOC), 0x400E1500 (PIOD), 0x400E1700 (PIOE)

Access: Read/Write

31	30	29	28	27	26	25	24
SCHMITT31	SCHMITT30	SCHMITT29	SCHMITT28	SCHMITT27	SCHMITT26	SCHMITT25	SCHMITT24
23	22	21	20	19	18	17	16
SCHMITT23	SCHMITT22	SCHMITT21	SCHMITT20	SCHMITT19	SCHMITT18	SCHMITT17	SCHMITT16
15	14	13	12	11	10	9	8
SCHMITT15	SCHMITT14	SCHMITT13	SCHMITT12	SCHMITT11	SCHMITT10	SCHMITT9	SCHMITT8
7	6	5	4	3	2	1	0
SCHMITT7	SCHMITT6	SCHMITT5	SCHMITT4	SCHMITT3	SCHMITT2	SCHMITT1	SCHMITT0

- **SCHMITTx [x=0..31]: Schmitt Trigger Control**

0: Schmitt trigger is enabled.

1: Schmitt trigger is disabled.

33.6.49 PIO I/O Delay Register

Name: PIO_DELAYR

Address: 0x400E0F10 (PIOA), 0x400E1110 (PIOB), 0x400E1310 (PIOC), 0x400E1510 (PIOD), 0x400E1710 (PIOE)

Access: Read/Write

31	30	29	28	27	26	25	24
Delay7				Delay6			
23	22	21	20	19	18	17	16
Delay5				Delay4			
15	14	13	12	11	10	9	8
Delay3				Delay2			
7	6	5	4	3	2	1	0
Delay1				Delay0			

- **Delayx [x=0..7]: Delay Control for Simultaneous Switch Reduction**

Gives the number of elements in the delay line associated to pad x.

33.6.50 PIO Parallel Capture Mode Register

Name: PIO_PCMR

Address: 0x400E0F50 (PIOA), 0x400E1150 (PIOB), 0x400E1350 (PIOC), 0x400E1550 (PIOD), 0x400E1750 (PIOE)

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **PCEN: Parallel Capture Mode Enable**

0: The parallel capture mode is disabled.

1: The parallel capture mode is enabled.

- **DSIZE: Parallel Capture Mode Data Size**

Value	Name	Description
0	BYTE	The reception data in the PIO_PCRHR is a byte (8-bit)
1	HALF-WORD	The reception data in the PIO_PCRHR is a half-word (16-bit)
2	WORD	The reception data in the PIO_PCRHR is a word (32-bit)
3	–	Reserved

- **ALWYS: Parallel Capture Mode Always Sampling**

0: The parallel capture mode samples the data when both data enables are active.

1: The parallel capture mode samples the data whatever the data enables are.

- **HALFS: Parallel Capture Mode Half Sampling**

Independently from the ALWYS bit:

0: The parallel capture mode samples all the data.

1: The parallel capture mode samples the data only every other time.

- **FRSTS: Parallel Capture Mode First Sample**

This bit is useful only if the HALFS bit is set to 1. If data are numbered in the order that they are received with an index from 0 to n:

0: Only data with an even index are sampled.

1: Only data with an odd index are sampled.

33.6.51 PIO Parallel Capture Interrupt Enable Register

Name: PIO_PCIER

Address: 0x400E0F54 (PIOA), 0x400E1154 (PIOB), 0x400E1354 (PIOC), 0x400E1554 (PIOD), 0x400E1754 (PIOE)

Access: Write-only

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

The following configuration values are valid for all listed bit names of this register:

0: No effect

1: Enables the corresponding interrupt

- **DRDY: Parallel Capture Mode Data Ready Interrupt Enable**
- **OVRE: Parallel Capture Mode Overrun Error Interrupt Enable**
- **ENDRX: End of Reception Transfer Interrupt Enable**
- **RXBUFF: Reception Buffer Full Interrupt Enable**

33.6.52 PIO Parallel Capture Interrupt Disable Register

Name: PIO_PCIDR

Address: 0x400E0F58 (PIOA), 0x400E1158 (PIOB), 0x400E1358 (PIOC), 0x400E1558 (PIOD), 0x400E1758 (PIOE)

Access: Write-only

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

The following configuration values are valid for all listed bit names of this register:

0: No effect

1: Disables the corresponding interrupt

- **DRDY: Parallel Capture Mode Data Ready Interrupt Disable**
- **OVRE: Parallel Capture Mode Overrun Error Interrupt Disable**
- **ENDRX: End of Reception Transfer Interrupt Disable**
- **RXBUFF: Reception Buffer Full Interrupt Disable**

33.6.53 PIO Parallel Capture Interrupt Mask Register

Name: PIO_PCIMR

Address: 0x400E0F5C (PIOA), 0x400E115C (PIOB), 0x400E135C (PIOC), 0x400E155C (PIOD), 0x400E175C (PIOE)

Access: Read-only

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

The following configuration values are valid for all listed bit names of this register:

0: Corresponding interrupt is not enabled.

1: Corresponding interrupt is enabled.

- **DRDY: Parallel Capture Mode Data Ready Interrupt Mask**
- **OVRE: Parallel Capture Mode Overrun Error Interrupt Mask**
- **ENDRX: End of Reception Transfer Interrupt Mask**
- **RXBUFF: Reception Buffer Full Interrupt Mask**

33.6.54 PIO Parallel Capture Interrupt Status Register

Name: PIO_PCISR

Address: 0x400E0F60 (PIOA), 0x400E1160 (PIOB), 0x400E1360 (PIOC), 0x400E1560 (PIOD), 0x400E1760 (PIOE)

Access: Read-only

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

- **DRDY: Parallel Capture Mode Data Ready**

0: No new data is ready to be read since the last read of PIO_PCRHR.

1: A new data is ready to be read since the last read of PIO_PCRHR.

The DRDY flag is automatically reset when PIO_PCRHR is read or when the parallel capture mode is disabled.

- **OVRE: Parallel Capture Mode Overrun Error**

0: No overrun error occurred since the last read of this register.

1: At least one overrun error occurred since the last read of this register.

The OVRE flag is automatically reset when this register is read or when the parallel capture mode is disabled.

- **ENDRX: End of Reception Transfer**

0: The End of Transfer signal from the reception PDC channel is inactive.

1: The End of Transfer signal from the reception PDC channel is active.

- **RXBUFF: Reception Buffer Full**

0: The signal Buffer Full from the reception PDC channel is inactive.

1: The signal Buffer Full from the reception PDC channel is active.

33.6.55 PIO Parallel Capture Reception Holding Register

Name: PIO_PCRHR

Address: 0x400E0F64 (PIOA), 0x400E1164 (PIOB), 0x400E1364 (PIOC), 0x400E1564 (PIOD), 0x400E1764 (PIOE)

Access: Read-only

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

- **RDATA: Parallel Capture Mode Reception Data**

If DSIZE = 0 in PIO_PCMR, only the 8 LSBs of RDATA are useful.

If DSIZE = 1 in PIO_PCMR, only the 16 LSBs of RDATA are useful.

34. Serial Peripheral Interface (SPI)

34.1 Description

The Serial Peripheral Interface (SPI) circuit is a synchronous serial data link that provides communication with external devices in Master or Slave mode. It also enables communication between processors if an external processor is connected to the system.

The Serial Peripheral Interface is essentially a Shift register that serially transmits data bits to other SPIs. During a data transfer, one SPI system acts as the “master”' which controls the data flow, while the other devices act as “slaves'' which have data shifted into and out by the master. Different CPUs can take turn being masters (multiple master protocol, contrary to single master protocol where one CPU is always the master while all of the others are always slaves). One master can simultaneously shift data into multiple slaves. However, only one slave can 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 can transmit data at a variety of baud rates; there is one SPCK pulse for each bit that is transmitted.
- Slave Select (NSS)—This control line allows slaves to be turned on and off by hardware.

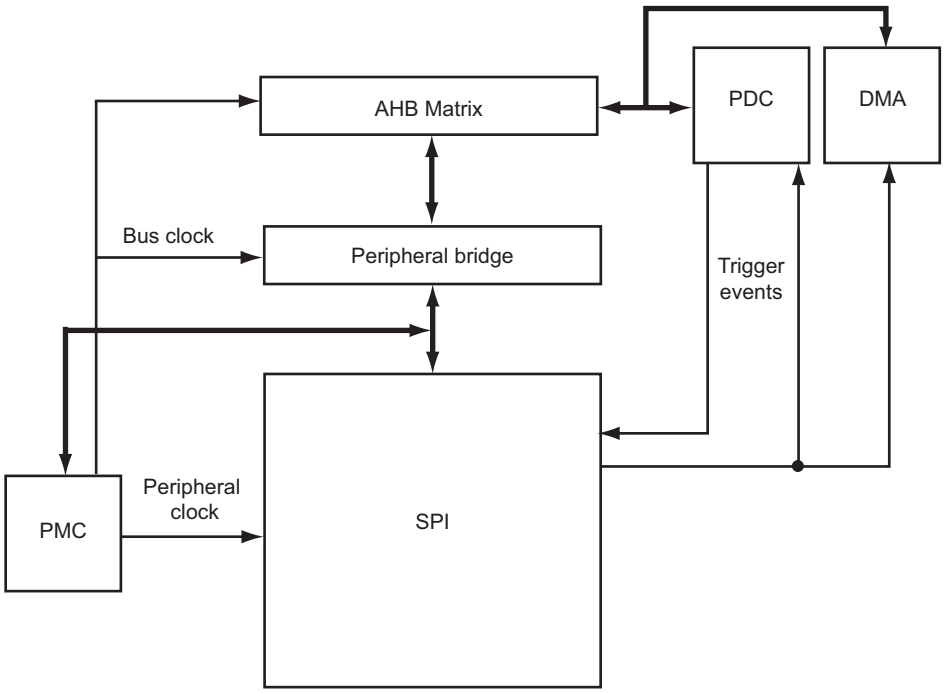
34.2 Embedded Characteristics

- Master or Slave Serial Peripheral Bus Interface
  - 8-bit to 16-bit programmable data length per chip select
  - Programmable phase and polarity per chip select
  - Programmable transfer delay between consecutive transfers and delay before SPI clock per chip select
  - Programmable delay between chip selects
  - Selectable mode fault detection
- Master Mode can drive SPCK up to Peripheral Clock
- Master Mode Bit Rate can be Independent of the Processor/Peripheral Clock
- Slave mode operates on SPCK, asynchronously with core and bus clock
- Four chip selects with external decoder support allow communication with up to 15 peripherals
- Communication with Serial External Devices Supported
  - Serial memories, such as DataFlash and 3-wire EEPROMs
  - Serial peripherals, such as ADCs, DACs, LCD controllers, CAN controllers and sensors
  - External coprocessors
- Connection to PDC Channel Capabilities, Optimizing Data Transfers
  - One channel for the receiver
  - One channel for the transmitter
- Connection to DMA Channel Capabilities, Optimizing Data Transfers
  - One channel for the receiver
  - One channel for the transmitter

- Register Write Protection

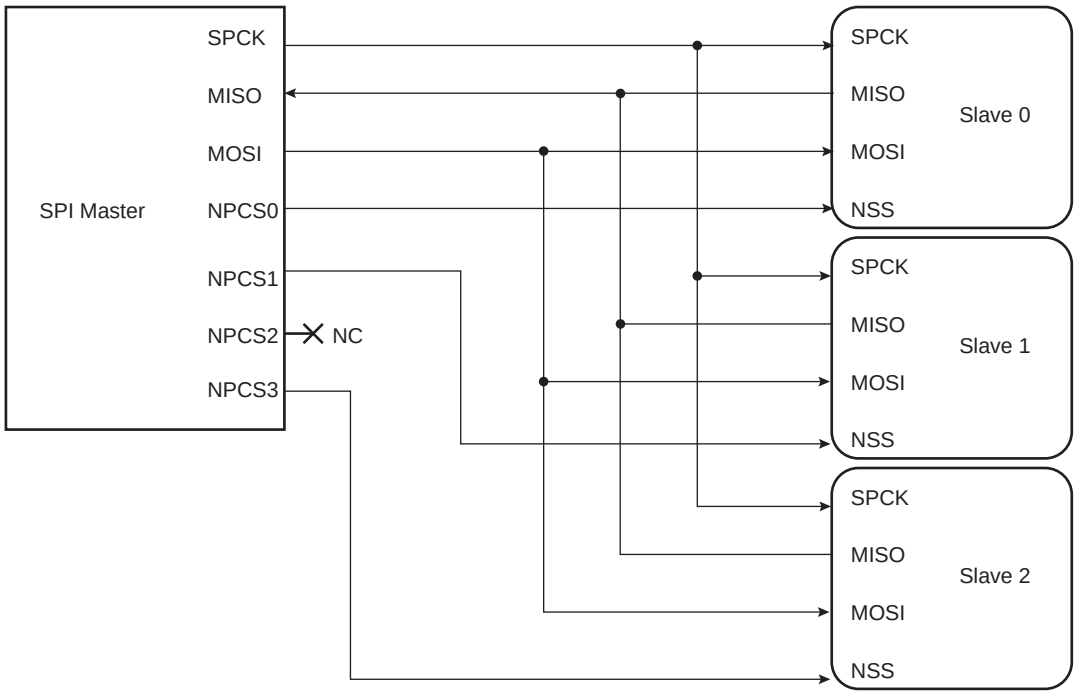
34.3 Block Diagram

Figure 34-1. Block Diagram



34.4 Application Block Diagram

Figure 34-2. Application Block Diagram: Single Master/Multiple Slave Implementation



34.5 Signal Description

Table 34-1. Signal Description

Pin Name	Pin Description	Type	
		Master	Slave
MISO	Master In Slave Out	Input	Output
MOSI	Master Out Slave In	Output	Input
SPCK	Serial Clock	Output	Input
NPCS1–NPCS3	Peripheral Chip Selects	Output	Unused
NPCS0/NSS	Peripheral Chip Select/Slave Select	Output	Input

34.6 Product Dependencies

34.6.1 I/O Lines

The pins used for interfacing the compliant external devices can be multiplexed with PIO lines. The programmer must first program the PIO controllers to assign the SPI pins to their peripheral functions.

Table 34-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
SPI	MISO	PA12	A
SPI	MOSI	PA13	A
SPI	NPCS0	PA11	A
SPI	NPCS1	PA9	B
SPI	NPCS1	PA31	A
SPI	NPCS1	PB14	A
SPI	NPCS1	PC4	B
SPI	NPCS2	PA10	B
SPI	NPCS2	PA30	B
SPI	NPCS2	PB2	B
SPI	NPCS3	PA3	B
SPI	NPCS3	PA5	B
SPI	NPCS3	PA22	B
SPI	SPCK	PA14	A

34.6.2 Power Management

The SPI can be clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the SPI clock.

34.6.3 Interrupt

The SPI interface has an interrupt line connected to the interrupt controller. Handling the SPI interrupt requires programming the interrupt controller before configuring the SPI.

Table 34-3. Peripheral IDs

Instance	ID
SPI	19

34.6.4 Peripheral DMA Controller (PDC) or Direct Memory Access Controller (DMAC)

The SPI interface can be used in conjunction with the PDC or DMAC in order to reduce processor overhead. For a full description of the PDC or DMAC, refer to the relevant section.

34.7 Functional Description

34.7.1 Modes of Operation

The SPI operates in Master mode or in Slave mode.

- The SPI operates in Master mode by setting the MSTR bit in the SPI Mode Register (SPI_MR):
  - Pins NPCS0 to NPCS3 are all configured as outputs
  - The SPCK pin is driven
  - The MISO line is wired on the receiver input
  - The MOSI line is driven as an output by the transmitter.
- The SPI operates in Slave mode if the MSTR bit in the SPI_MR is written to 0:
  - 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)
  - NPCS1 to NPCS3 are not driven and can be used for other purposes.

The data transfers are identically programmable for both modes of operation. The baud rate generator is activated only in Master mode.

34.7.2 Data Transfer

Four combinations of polarity and phase are available for data transfers. The clock polarity is programmed with the CPOL bit in the SPI chip select registers (SPI_CSRx). 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. Consequently, a master/slave pair must use the same parameter pair values to communicate. If multiple slaves are connected and require different configurations, the master must reconfigure itself each time it needs to communicate with a different slave.

Table 34-4 shows the four modes and corresponding parameter settings.

Table 34-4. SPI Bus Protocol Modes

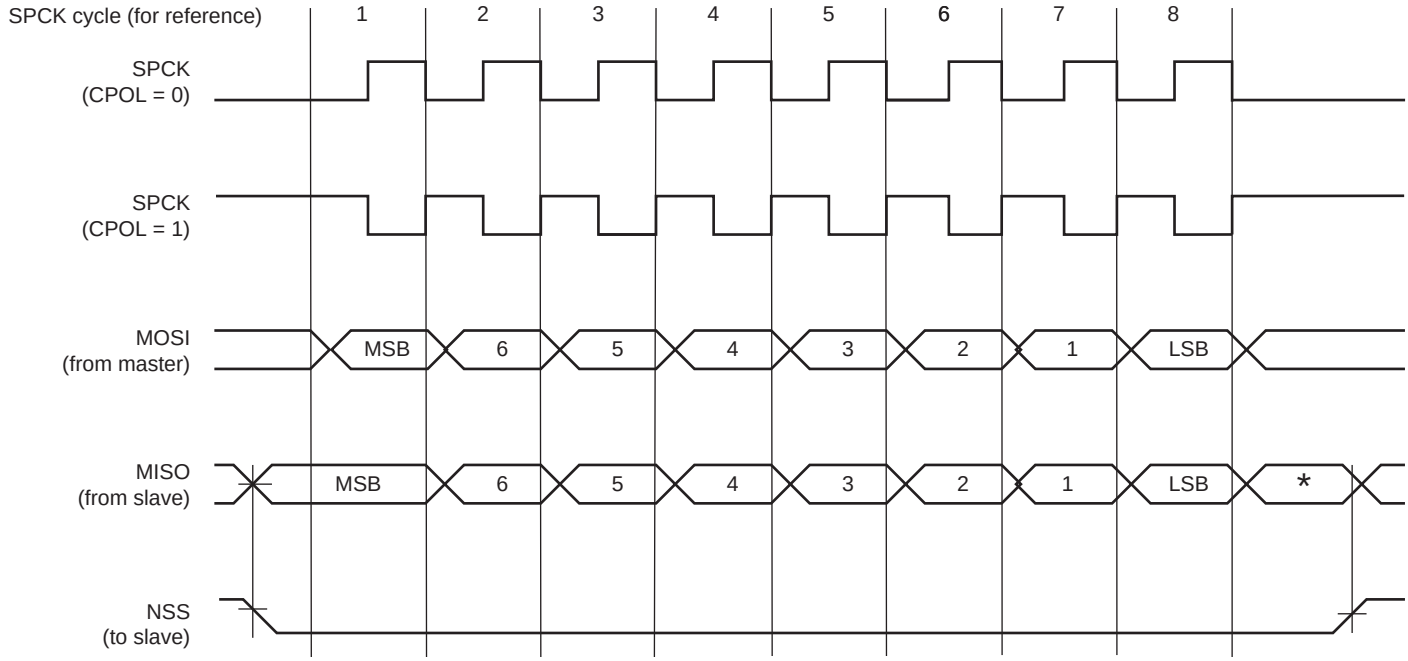
SPI Mode	CPOL	NCPHA	Shift SPCK Edge	Capture SPCK Edge	SPCK Inactive Level
0	0	1	Falling	Rising	Low

Table 34-4. SPI Bus Protocol Modes

SPI Mode	CPOL	NCPHA	Shift SPCK Edge	Capture SPCK Edge	SPCK Inactive Level
1	0	0	Rising	Falling	Low
2	1	1	Rising	Falling	High
3	1	0	Falling	Rising	High

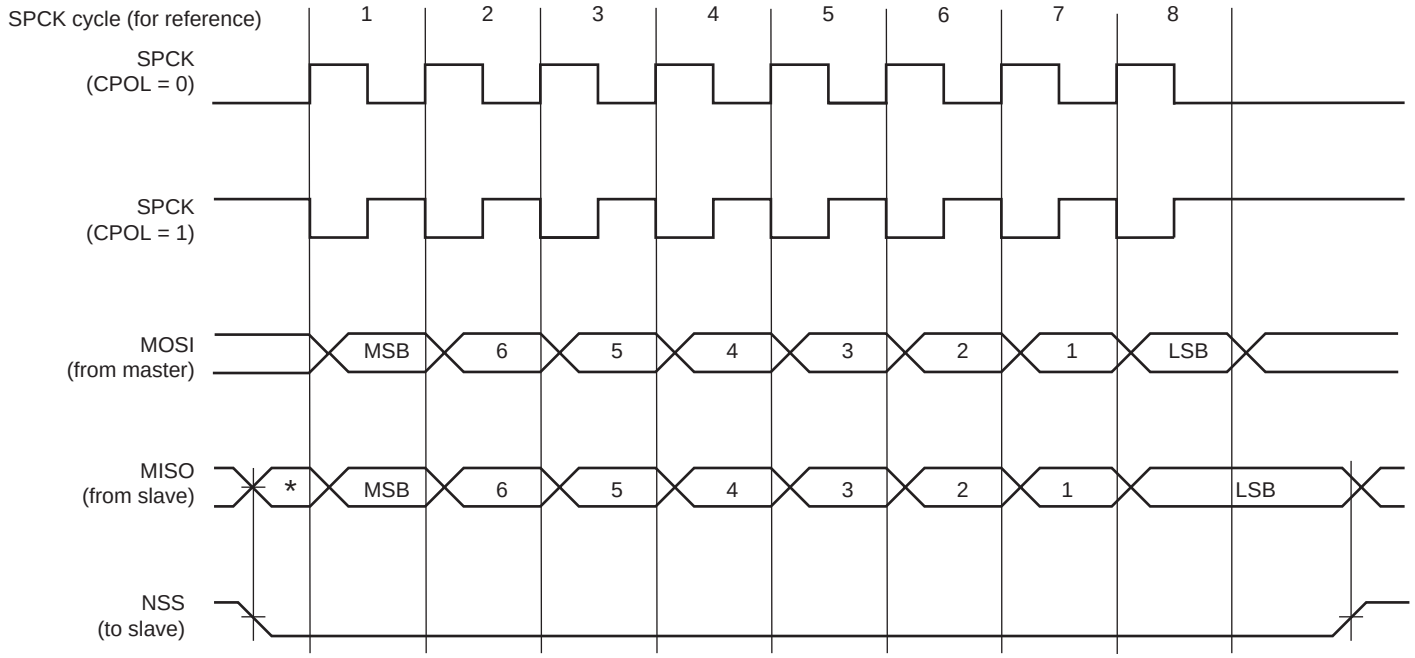
Figure 34-3 and Figure 34-4 show examples of data transfers.

Figure 34-3. SPI Transfer Format (NCPHA = 1, 8 bits per transfer)



* Not defined.

Figure 34-4. SPI Transfer Format (NCPHA = 0, 8 bits per transfer)



* Not defined.

34.7.3 Master Mode Operations

When configured in Master mode, the SPI operates on the clock generated by the internal programmable baud rate generator. It fully controls the data transfers to and from the slave(s) connected to the SPI bus. The SPI drives the chip select line to the slave and the serial clock signal (SPCK).

The SPI features two holding registers, the Transmit Data Register (SPI_TDR) and the Receive Data Register (SPI_RDR), and a single shift register. The holding registers maintain the data flow at a constant rate.

After enabling the SPI, a data transfer starts when the processor writes to the SPI_TDR. The written data is immediately transferred in the Shift register and the 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. Data cannot be loaded in the SPI_RDR without transmitting data. If there is no data to transmit, dummy data can be used (SPI_TDR filled with ones). If the SPI_MR.WDRBT bit is set, transmission can occur only if the SPI_RDR has been read. If Receiving mode is not required, for example when communicating with a slave receiver only (such as an LCD), the receive status flags in the SPI Status register (SPI_SR) can be discarded.

Before writing the SPI_TDR, the PCS field in the SPI_MR must be set in order to select a slave.

If new data is written in the SPI_TDR during the transfer, it is kept in the SPI_TDR until the current transfer is completed. Then, the received data is transferred from the Shift register to the SPI_RDR, the data in the SPI_TDR is loaded in the Shift register and a new transfer starts.

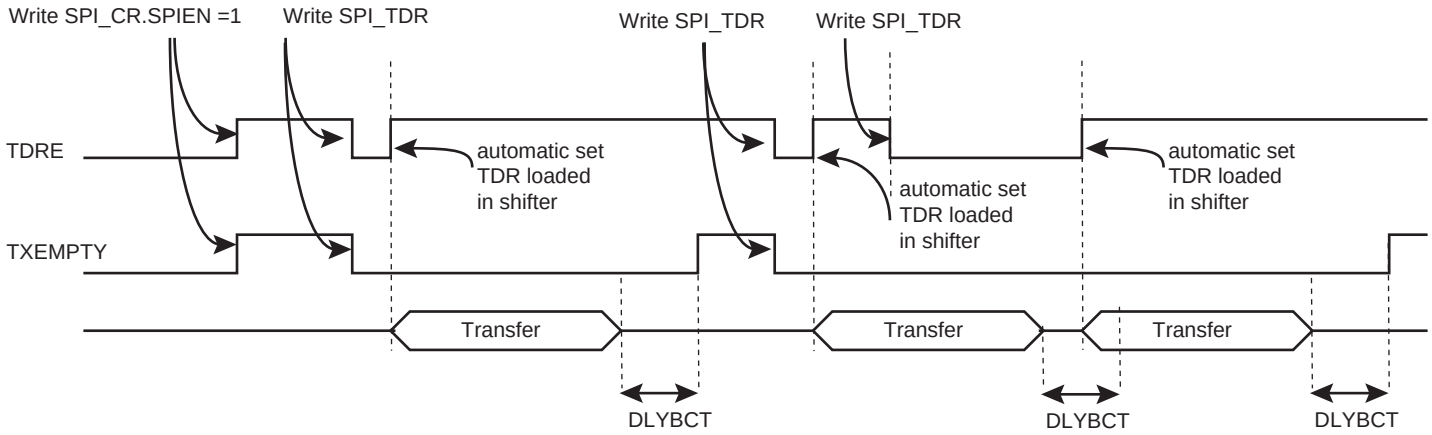
As soon as the SPI_TDR is written, the Transmit Data Register Empty (TDRE) flag in the SPI_SR is cleared. When the data written in the SPI_TDR is loaded into the Shift register, the TDRE flag in the SPI_SR is set. The TDRE bit is used to trigger the Transmit PDC or DMA channel.

See Figure 34-5.

The end of transfer is indicated by the TXEMPTY flag in the SPI_SR. If a transfer delay (DLYBCT) is greater than 0 for the last transfer, TXEMPTY is set after the completion of this delay. The peripheral clock can be switched off at this time.

Note: When the SPI is enabled, the TDRE and TXEMPTY flags are set.

Figure 34-5. TDRE and TXEMPTY flag behavior



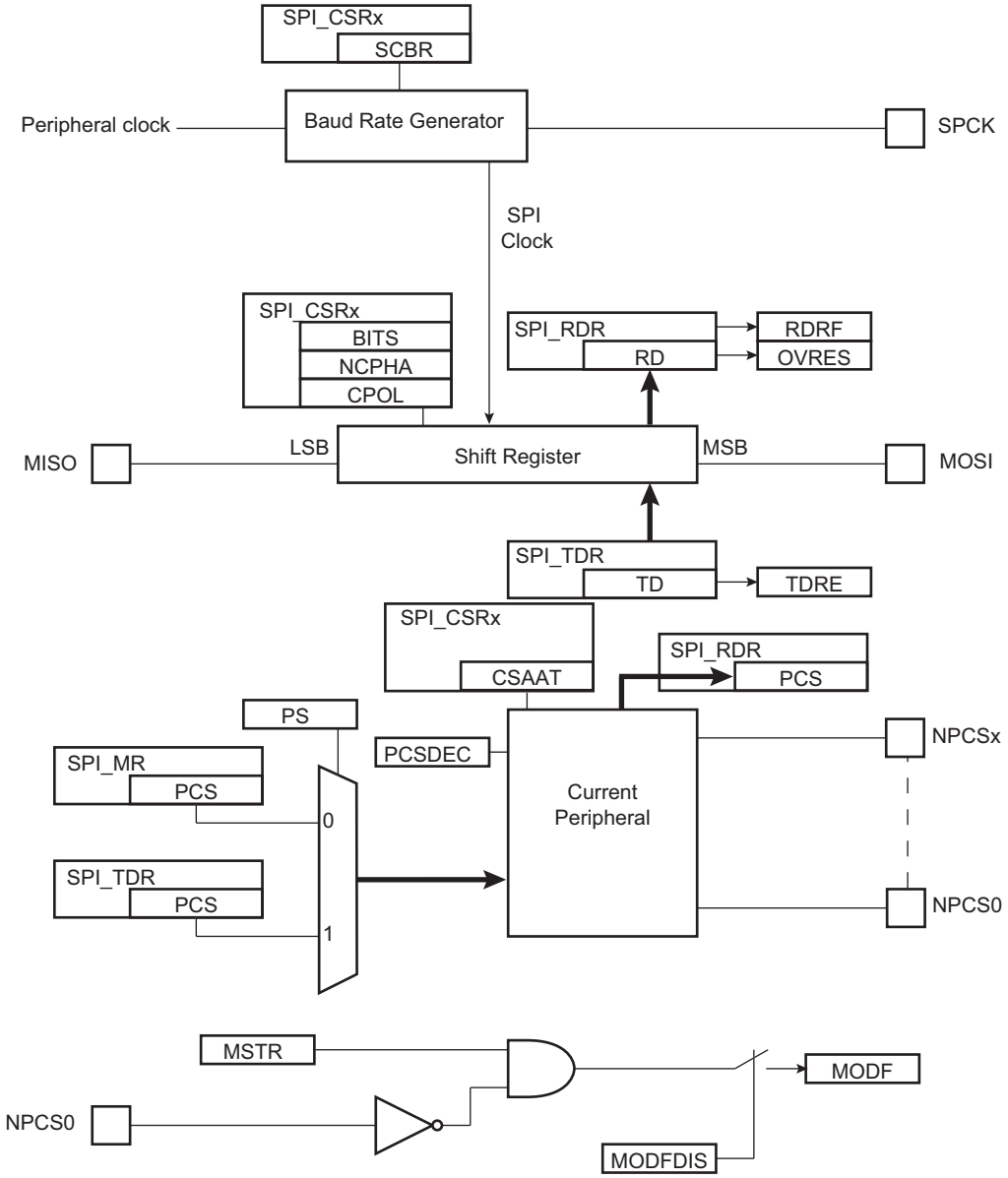
The transfer of received data from the Shift register to the SPI_RDR is indicated by the Receive Data Register Full (RDRF) bit in the SPI_SR. When the received data is read, the RDRF bit is cleared.

If the SPI_RDR has not been read before new data is received, the Overrun Error (OVRES) bit in the SPI_SR is set. As long as this flag is set, data is loaded in the SPI_RDR. The user has to read the SPI_SR to clear the OVRES bit.

Figure 34-6 shows a block diagram of the SPI when operating in Master mode. Figure 34-7 shows a flow chart describing how transfers are handled.

34.7.3.1 Master Mode Block Diagram

Figure 34-6. Master Mode Block Diagram



34.7.3.2 Master Mode Flow Diagram

Figure 34-7. Master Mode Flow Diagram

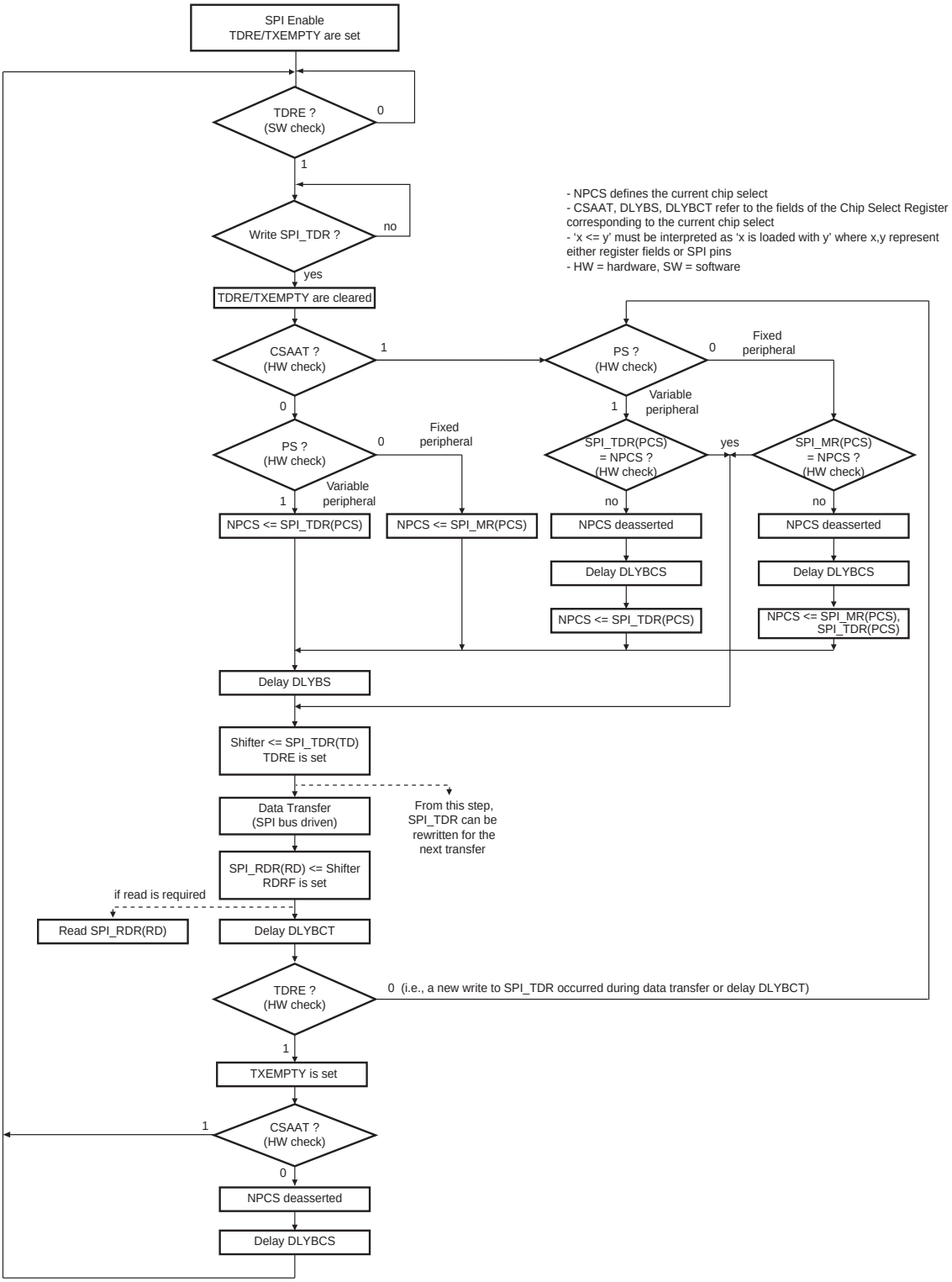


Figure 34-8 shows the behavior of Transmit Data Register Empty (TDRE), Receive Data Register (RDRF) and Transmission Register Empty (TXEMPTY) status flags within the SPI_SR during an 8-bit data transfer in Fixed mode without the PDC or DMA involved.

Figure 34-8. Status Register Flags Behavior

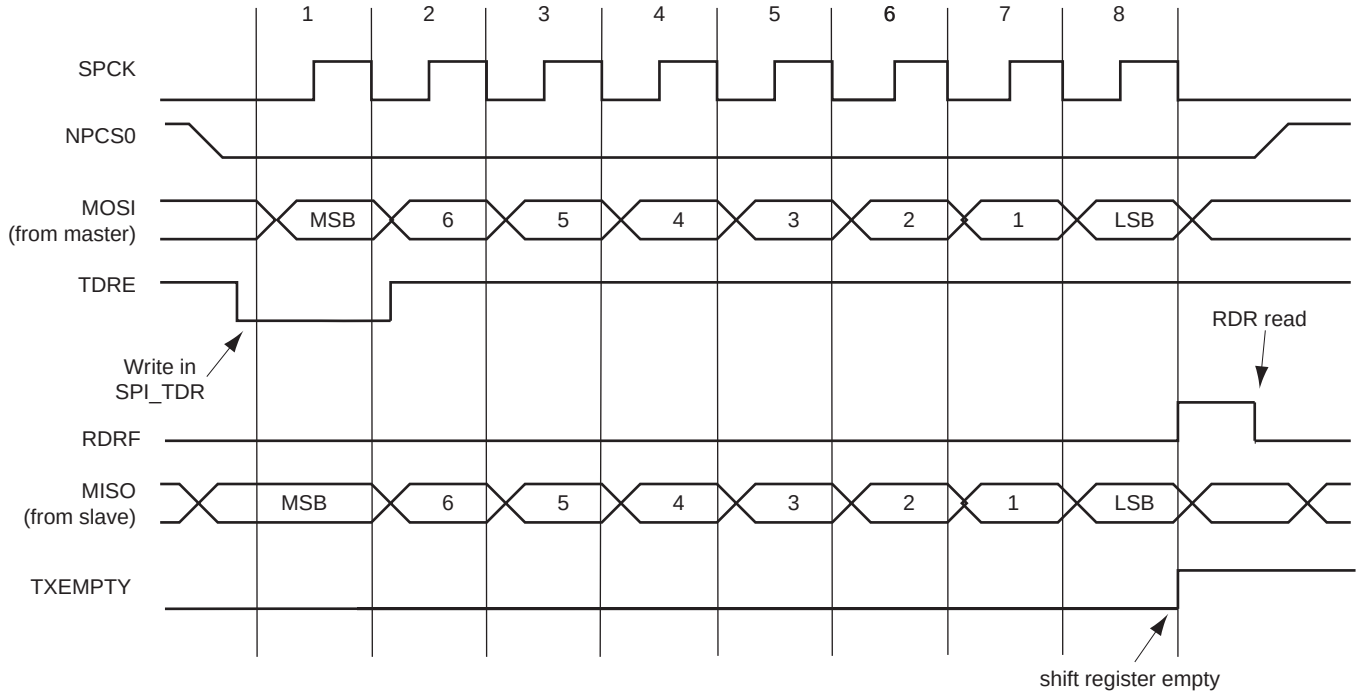
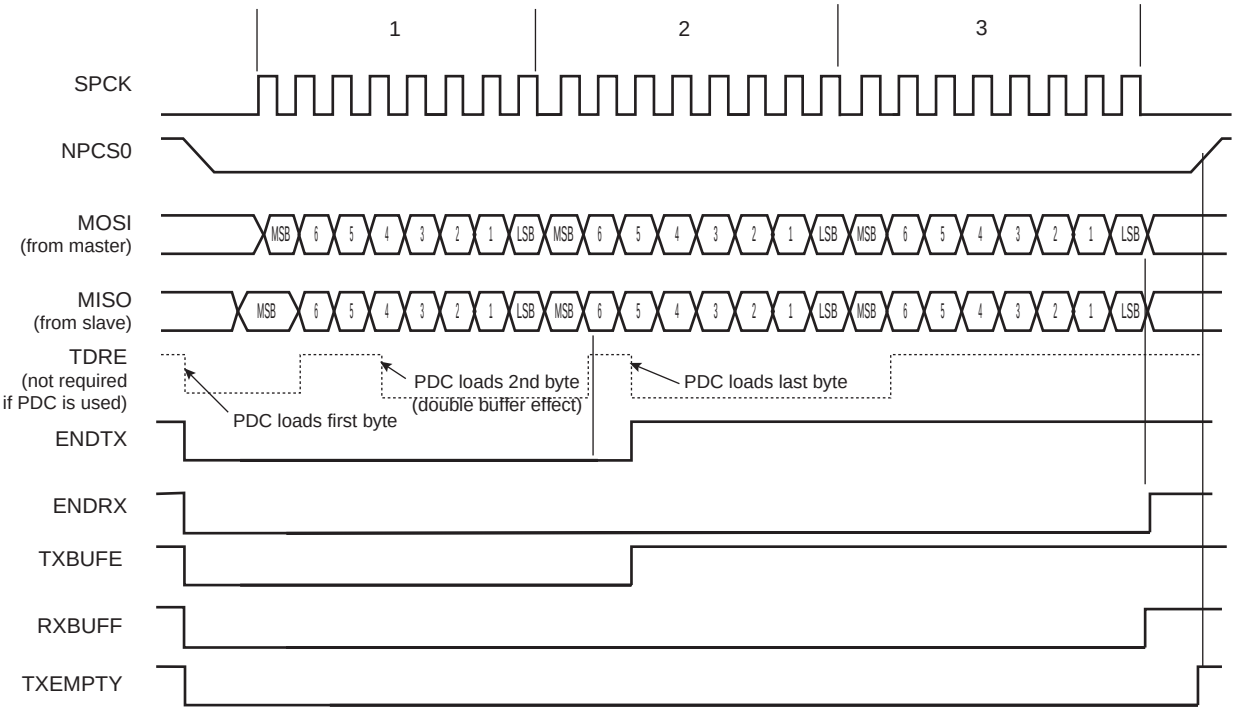


Figure 34-9 shows the behavior of Transmission Register Empty (TXEMPTY), End of RX buffer (ENDRX), End of TX buffer (ENDTX), RX Buffer Full (RXBUFF) and TX Buffer Empty (TXBUFE) status flags within the SPI_SR during an 8-bit data transfer in Fixed mode with the PDC involved. The PDC is programmed to transfer and receive three units of data. The next pointer and counter are not used. The RDRF and TDRE are not shown because these flags are managed by the PDC when using the PDC.

Figure 34-9. PDC Status Register Flags Behavior



34.7.3.3 Clock Generation

The SPI Baud rate clock is generated by dividing the peripheral clock by a value between 1 and 255.

If the SCBR field in the SPI_CSR is programmed to 1, the operating baud rate is peripheral clock (see the electrical characteristics section for the SPCK maximum frequency). 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 to 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. This allows the SPI to automatically adapt the baud rate for each interfaced peripheral without reprogramming.

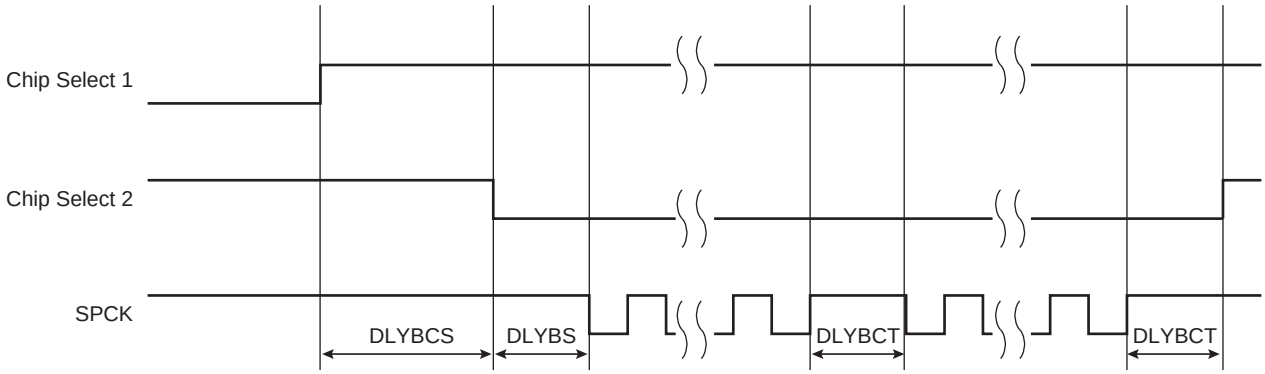
34.7.3.4 Transfer Delays

Figure 34-10 shows a chip select transfer change and consecutive transfers on the same chip select. Three delays can be programmed to modify the transfer waveforms:

- Delay between the chip selects—programmable only once for all chip selects by writing the DLYBCS field in the SPI_MR. The SPI slave device deactivation delay is managed through DLYBCS. If there is only one SPI slave device connected to the master, the DLYBCS field does not need to be configured. If several slave devices are connected to a master, DLYBCS must be configured depending on the highest deactivation delay. Refer to the SPI slave device electrical characteristics.
- Delay before SPCK—independently programmable for each chip select by writing the DLYBS field. The SPI slave device activation delay is managed through DLYBS. Refer to the SPI slave device electrical characteristics to define DLYBS.
- Delay between consecutive transfers—independently programmable for each chip select by writing the DLYBCT field. The time required by the SPI slave device to process received data is managed through DLYBCT. This time depends on the SPI slave system activity.

These delays allow the SPI to be adapted to the interfaced peripherals and their speed and bus release time.

Figure 34-10. Programmable Delays



34.7.3.5 Peripheral Selection

The serial peripherals are selected through the assertion of the NPCS0 to NPCS3 signals. By default, all NPCS signals are high before and after each transfer.

- **Fixed Peripheral Select Mode**: SPI exchanges data with only one peripheral.
  Fixed Peripheral Select mode is enabled by clearing the PS bit in the SPI_MR. In this case, the current peripheral is defined by the PCS field in the SPI_MR and the PCS field in the SPI_TDR has no effect.
- **Variable Peripheral Select Mode:** Data can be exchanged with more than one peripheral without having to reprogram the NPCS field in the SPI_MR.
  Variable Peripheral Select mode is enabled by setting the PS bit in the SPI_MR. The PCS field in the 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 has the following format:

  [xxxxxxx(7-bit) + LASTXFER(1-bit)(1)+ xxxx(4-bit) + PCS (4-bit) + DATA (8 to 16-bit)] with PCS equals the chip select to assert, as defined in Section 34.8.4 “SPI Transmit Data Register” and LASTXFER bit at 0 or 1 depending on the CSAAT bit.

Note: 1. Optional

CSAAT, LASTXFER and CSNAAT bits are discussed in Section 34.7.3.10 “Peripheral Deselection with DMA or PDC”.

If LASTXFER is used, the command must be issued after writing the last character. Instead of LASTXFER, the user can use the SPIDIS command. After the end of the DMA or PDC transfer, it is necessary to wait for the TXEMPTY flag and then write SPIDIS into the SPI Control Register (SPI_CR). This does not change the configuration register values). The NPCS is disabled after the last character transfer. Then, another DMA or PDC transfer can be started if the SPIEN has previously been written in the SPI_CR.

34.7.3.6 SPI Peripheral DMA Controller (PDC)

In both Fixed and Variable Peripheral Select modes, the Peripheral DMA Controller (PDC) can be used to reduce processor overhead.

The fixed peripheral selection allows buffer transfers with a single peripheral. Using the PDC is an optimal means, as the size of the data transfer between the memory and the SPI is either 8 bits or 16 bits. However, if the peripheral selection is modified, the SPI_MR must be reprogrammed.

The variable peripheral selection allows buffer transfers with multiple peripherals without reprogramming the SPI_MR. Data written in the SPI_TDR is 32 bits wide and defines the real data to be transmitted and the destination peripheral. Using the PDC in this mode requires 32-bit wide buffers, with the data in the LSBs and the PCS and LASTXFER fields in the MSBs. However, the SPI still controls the number of bits (8 to16) to be transferred through MISO and MOSI lines with the chip select configuration registers (SPI_CSRx). This is not the

optimal means in terms of memory size for the buffers, but it provides a very effective means to exchange data with several peripherals without any intervention of the processor.

Transfer Size

Depending on the data size to transmit, from 8 to 16 bits, the PDC manages automatically the type of pointer size it has to point to. The PDC performs the following transfer, depending on the mode and number of bits per data.

- Fixed mode:
  - 8-bit data:
    1-byte transfer,
    PDC pointer address = address + 1 byte,
    PDC counter = counter - 1
  - 9-bit to 16-bit data:
    2-byte transfer. n-bit data transfer with don't care data (MSB) filled with 0's,
    PDC pointer address = address + 2 bytes,
    PDC counter = counter - 1
- Variable mode:
  - In Variable mode, PDC pointer address = address + 4 bytes and PDC counter = counter - 1 for 8 to 16-bit transfer size.
  - When using the PDC, the TDRE and RDRF flags are handled by the PDC. The user's application does not have to check these bits. Only End of RX Buffer (ENDRX), End of TX Buffer (ENDTX), Buffer Full (RXBUFF), TX Buffer Empty (TXBUFE) are significant. For further details about the Peripheral DMA Controller and user interface, refer to the PDC section.

34.7.3.7 SPI Direct Access Memory Controller (DMAC)

In both Fixed and Variable modes, 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, if the peripheral selection is modified, the SPI_MR must be reprogrammed.

The variable peripheral selection allows buffer transfers with multiple peripherals without reprogramming the SPI_MR. Data written in the SPI_TDR is 32 bits wide and defines the real data to be transmitted and the destination peripheral. 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 to 16) to be transferred through MISO and MOSI lines with the chip select configuration registers. This is not the optimal means in terms of memory size for the buffers, but it provides a very effective means to exchange data with several peripherals without any intervention of the processor.

34.7.3.8 Peripheral Chip Select Decoding

The user can program the SPI to operate with up to 15 slave peripherals by decoding the four chip select lines, NPCS0 to NPCS3 with an external decoder/demultiplexer (refer to Figure 34-11). This can be enabled by setting the PCSDEC bit in the 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 the NPCS lines of either SPI_MR or SPI_TDR (depending on PS).

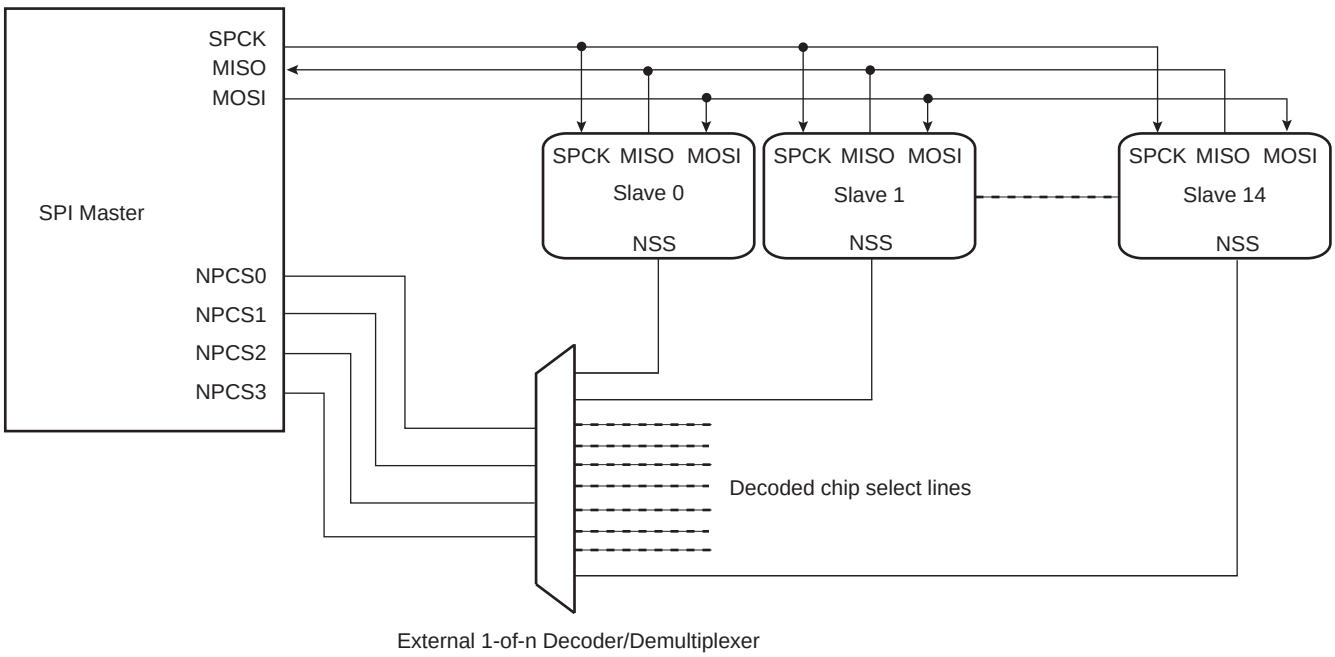
As the SPI sets a default value of 0xF on the chip select lines (i.e., all chip select lines at 1) when not processing any transfer, only 15 peripherals can be decoded.

The SPI has four chip select registers (SPI_CSR0...SPI_CSR3). As a result, when external decoding is activated, each NPCS 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. Consequently, the user has to make sure to connect compatible peripherals on the decoded chip select lines 0 to 3, 4 to 7, 8 to 11 and 12 to 14. Figure 34-11 shows this type of implementation.

If the CSAAT bit is used, with or without the PDC, the Mode Fault detection for NPCS0 line must be disabled. This is not required for all other chip select lines since mode fault detection is only on NPCS0.

If the CSAAT bit is used, with or without the DMAC, the Mode Fault detection for NPCS0 line must be disabled. This is not needed for all other chip select lines since Mode Fault detection is only on NPCS0.

Figure 34-11. Chip Select Decoding Application Block Diagram: Single Master/Multiple Slave Implementation



34.7.3.9 Peripheral Deselection without DMA nor PDC

During a transfer of more than one unit of data on a chip select without the DMA nor PDC, the SPI_TDR is loaded by the processor, the TDRE flag 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 DLYBCT value (delay between consecutive transfers) in the SPI_CSR, gives even less time for the processor to reload the SPI_TDR. With some SPI slave peripherals, if the chip select line must remain active (low) during a full set of transfers, communication errors can occur.

To facilitate interfacing with such devices, the chip select registers [SPI_CSR0...SPI_CSR3] can be programmed with the Chip Select Active After Transfer (CSAAT) bit at 1. This allows the chip select lines to remain in their current state (low = active) until a transfer to another chip select is required. Even if the SPI_TDR is not reloaded, the chip select remains active. To de-assert the chip select line at the end of the transfer, the Last Transfer (LASTXFER) bit in SPI_CR must be set after writing the last data to transmit into SPI_TDR.

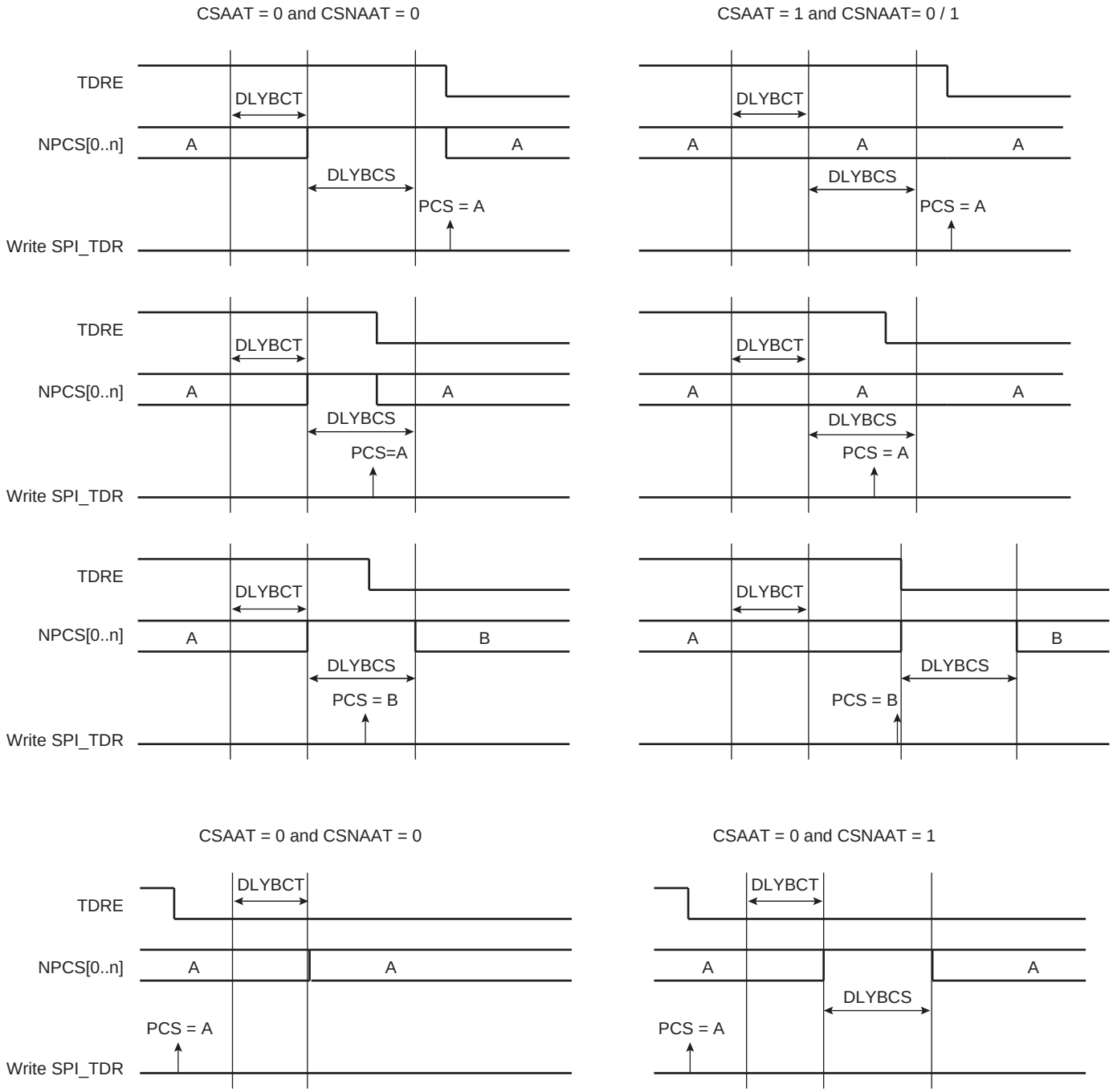
34.7.3.10 Peripheral Deselection with DMA or PDC

DMA or PDC provides faster reloads of the SPI_TDR compared to software. However, depending on the system activity, it is not guaranteed that the SPI_TDR is written with the next data before the end of the current transfer. Consequently, data can be lost by the de-assertion of the NPCS line for SPI slave peripherals requiring the chip select line to remain active between two transfers. The only way to guarantee a safe transfer in this case is the use of the CSAAT and LASTXFER bits.

When the CSAAT bit is configured to 0, the NPCS does not rise in all cases between two transfers on the same peripheral. During a transfer on a chip select, the TDRE flag rises as soon as the content of the SPI_TDR is transferred into the internal shift register. 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 can lead to difficulties to interface with some serial peripherals requiring the chip select to be de-asserted after each transfer. To facilitate interfacing with such devices, the SPI_CSR can be programmed with the Chip Select Not Active After Transfer (CSNAAT) bit at 1. This allows the chip select lines to be de-asserted systematically during a time “DLYBCS” (the value of the CSNAAT bit is processed only if the CSAAT bit is configured to 0 for the same chip select).

Figure 34-12 shows different peripheral deselection cases and the effect of the CSAAT and CSNAAT bits.

Figure 34-12. Peripheral Deselection



34.7.3.11 Mode Fault Detection

The SPI has the capability to operate in multi-master environment. Consequently, the NPCS0/NSS line must be monitored. If one of the masters on the SPI bus is currently transmitting, the NPCS0/NSS line is low and the SPI must not transmit any data. 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 multi-master environment, NPCS0, MOSI, MISO and SPCK pins must be configured in open drain (through the PIO controller). When a mode fault is detected, the SPI_SR.MODF bit is set until SPI_SR is read and the SPI is automatically disabled until it is re-enabled by setting the SPI_CR.SPIEN bit.

By default, the mode fault detection is enabled. The user can disable it by setting the SPI_MR.MODFDIS bit.

34.7.4 SPI Slave Mode

When operating in Slave mode, the SPI processes data bits on the clock provided on the SPI clock pin (SPCK).

The SPI waits until NSS goes active before receiving the serial clock from an external master. When NSS falls, the clock is validated and the data is loaded in the SPI_RDR depending on the BITS field configured in SPI_CSR0. These bits are processed following a phase and a polarity defined respectively by the NCPHA and CPOL bits in SPI_CSR0. Note that the fileds BITS, CPOL and NCPHA of the other chip select registers (SPI_CSR1...SPI_CSR3) 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.

Note: For more information on the BITS field, see also the note below the SPI_CSRx register bitmap (Section 34.8.9 “SPI Chip Select Register”).

When all bits are processed, the received data is transferred in the SPI_RDR and the RDRF bit rises. If the SPI_RDR has not been read before new data is received, the Overrun Error Status (OVRES) bit in the SPI_SR is set. As long as this flag is set, data is loaded in the SPI_RDR. The user must read SPI_SR to clear the OVRES bit.

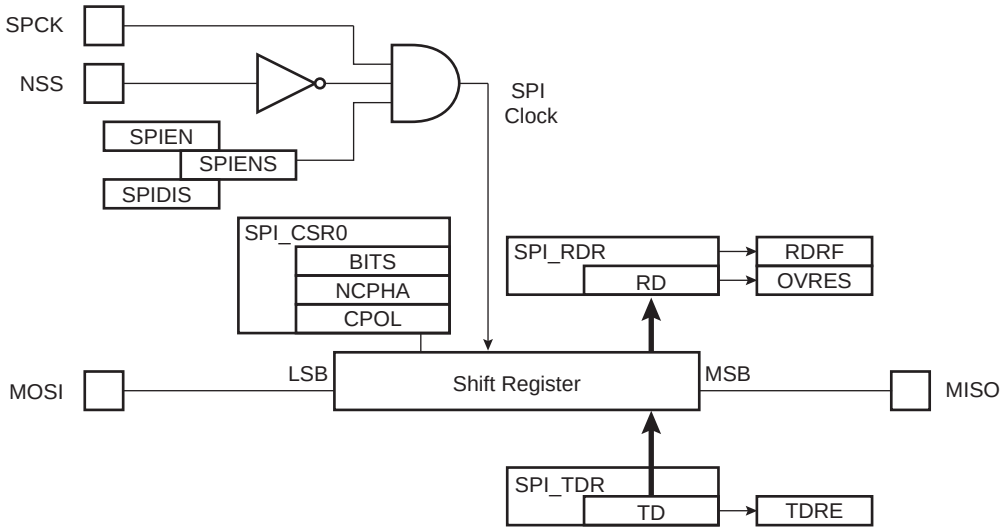
When a transfer starts, the data shifted out is the data present in the Shift register. If no data has been written in the SPI_TDR, the last data received is transferred. If no data has been received since the last reset, all bits are transmitted low, as the Shift register resets to 0.

When a first data is written in the SPI_TDR, it is transferred immediately in the Shift register and the TDRE flag rises. If new data is written, it remains in the 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 the SPI_TDR is transferred in the Shift register and the TDRE flag rises. This enables frequent updates of critical variables with single transfers.

Then, new data is loaded in the Shift register from the SPI_TDR. If no character is ready to be transmitted, i.e., no character has been written in the SPI_TDR since the last load from the SPI_TDR to the Shift register, the SPI_TDR is retransmitted. In this case the Underrun Error Status Flag (UNDES) is set in the SPI_SR.

Figure 34-13 shows a block diagram of the SPI when operating in Slave mode.

Figure 34-13. Slave Mode Functional Block Diagram



34.7.5 Register Write Protection

To prevent any single software error from corrupting SPI behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the SPI Write Protection Mode Register (SPI_WPMR).

If a write access to a write-protected register is detected, the WPVS flag in the SPI Write Protection Status Register (SPI_WPSR) is set and the WPVSRC field indicates the register in which the write access has been attempted.

The WPVS bit is automatically cleared after reading SPI_WPSR.

The following registers can be write-protected:

- SPI Mode Register
- SPI Chip Select Register

34.8 Serial Peripheral Interface (SPI) User Interface

In the “Offset” column of Table 34-5, ‘CS_number’ denotes the chip select number.

Table 34-5. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	SPI_CR	Write-only	–
0x04	Mode Register	SPI_MR	Read/Write	0x0
0x08	Receive Data Register	SPI_RDR	Read-only	0x0
0x0C	Transmit Data Register	SPI_TDR	Write-only	–
0x10	Status Register	SPI_SR	Read-only	0x000000F0
0x14	Interrupt Enable Register	SPI_IER	Write-only	–
0x18	Interrupt Disable Register	SPI_IDR	Write-only	–
0x1C	Interrupt Mask Register	SPI_IMR	Read-only	0x0
0x20–0x2C	Reserved	–	–	–
0x30 + (CS_number * 0x04)	Chip Select Register	SPI_CSR	Read/Write	0x0
0x40–0x48	Reserved	–	–	–
0x4C–0xE0	Reserved	–	–	–
0xE4	Write Protection Mode Register	SPI_WPMR	Read/Write	0x0
0xE8	Write Protection Status Register	SPI_WPSR	Read-only	0x0
0xEC–0xF8	Reserved	–	–	–
0xFC	Reserved	–	–	–
0x100–0x124	Reserved for PDC Registers	–	–	–

34.8.1 SPI Control Register

Name: SPI_CR

Address: 0x40088000

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	LASTXFER
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	REQCLR	–	–	–	–
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.f

1: Disables the SPI.

All pins are set in Input mode after completion of the transmission in progress, if any.

If a transfer is in progress when SPIDIS is set, the SPI completes the transmission of the shifter register and does not start any new transfer, even if the SPI_THR is loaded.

Note: If both SPIEN and SPIDIS are equal to one when the SPI_CR is written, the SPI is disabled.

- **SWRST: SPI Software Reset**

0: No effect.

1: Reset the SPI. A software-triggered hardware reset of the SPI interface is performed.

The SPI is in Slave mode after software reset.

PDC channels are not affected by software reset.

- **REQCLR: Request to Clear the Comparison Trigger**

0: No effect.

1: Restarts the comparison trigger to enable SPI_RDR loading.

- **LASTXFER: Last Transfer**

0: No effect.

1: The current NPCS is de-asserted after the character written in TD has been transferred. When SPI_CSRx.CSAAT is set, the communication with the current serial peripheral can be closed by raising the corresponding NPCS line as soon as TD transfer is completed.

Refer to Section 34.7.3.5 “Peripheral Selection” for more details.

34.8.2 SPI Mode Register

Name: SPI_MR

Address: 0x40088004

Access: Read/Write

31	30	29	28	27	26	25	24
DLYBCS							
23	22	21	20	19	18	17	16
–	–	–	–	PCS			
15	14	13	12	11	10	9	8
–	–	–	-	–	–	–	–
7	6	5	4	3	2	1	0
LLB	–	WDRBT	MODFDIS	–	PCSDEC	PS	MSTR

This register can only be written if the WPEN bit is cleared in the 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 select lines are directly connected to a peripheral device.

1: The four NPCS chip select lines are connected to a 4-bit to 16-bit decoder.

When PCSDEC = 1, up to 15 chip select signals can be generated with the four NPCS lines using an external 4-bit to 16-bit decoder. The chip select registers define the characteristics of the 15 chip selects, with 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 enabled

1: Mode fault detection disabled

- **WDRBT: Wait Data Read Before Transfer**

0: No Effect. In Master mode, a transfer can be initiated regardless of the SPI_RDR state.

1: In Master mode, a transfer can start only if the SPI_RDR 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 shift register 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 SPI_MR.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 SPI_MR.PCSDEC = 1:

NPCS[3:0] output signals = PCS.

- **DLYBCS: Delay Between Chip Selects**

This field defines the delay between the inactivation and the activation of 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 lower than 6, six peripheral clock periods are inserted by default.

Otherwise, the following equation determines the delay:

:

$$\text{Delay Between Chip Selects} = \frac{\text{DLYBCS}}{f_{\text{peripheral clock}}}$$

34.8.3 SPI Receive Data Register

Name: SPI_RDR

Address: 0x40088008

Access: Read-only

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

- **RD: Receive Data**

Data received by the SPI Interface is stored in this register in a right-justified format. Unused bits are read as 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 are read as zero.

Note: When using Variable Peripheral Select mode (PS = 1 in SPI_MR), it is mandatory to set the SPI_MR.WDRBT bit if the PCS field must be processed in SPI_RDR.

34.8.4 SPI Transmit Data Register

Name: SPI_TDR

Address: 0x4008800C

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	LASTXFER

23	22	21	20	19	18	17	16
–	–	–	–	PCS			

15	14	13	12	11	10	9	8
TD							

7	6	5	4	3	2	1	0
TD							

- **TD: Transmit Data**

Data to be transmitted by the SPI Interface is stored in this register. Information to be transmitted must be written to the transmit data register in a right-justified format.

- **PCS: Peripheral Chip Select**

This field is only used if variable peripheral select is active (PS = 1).

If SPI_MR.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 SPI_MR.PCSDEC = 1:

NPCS[3:0] output signals = PCS.

- **LASTXFER: Last Transfer**

0: No effect

1: The current NPCS is de-asserted after the transfer of the character written in TD. When SPI_CSRx.CSAAT is set, the communication with the current serial peripheral can be closed by raising the corresponding NPCS line as soon as TD transfer is completed.

This field is only used if variable peripheral select is active (SPI_MR.PS = 1).

34.8.5 SPI Status Register

Name: SPI_SR

Address: 0x40088010

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	SPIENS
15	14	13	12	11	10	9	8
–	–	–	–	–	UNDES	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

- **RDRF: Receive Data Register Full (cleared by reading SPI_RDR)**

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 shift register to SPI_RDR since the last read of SPI_RDR.

- **TDRE: Transmit Data Register Empty (cleared by writing SPI_TDR)**

0: Data has been written to SPI_TDR and not yet transferred to the shift register.

1: The last data written in the SPI_TDR has been transferred to the shift register.

TDRE equals zero when the SPI is disabled or at reset. The SPI enable command sets this bit to 1.

- **MODF: Mode Fault Error (cleared on read)**

0: No mode fault has been detected since the last read of SPI_SR.

1: A mode fault occurred since the last read of SPI_SR.

- **OVRES: Overrun Error Status (cleared on read)**

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 shift register since the last read of the SPI_RDR.

- **ENDRX: End of RX Buffer (cleared by writing SPI_RCR or SPI_RNCR)**

0: The Receive Counter register has not reached 0 since the last write in SPI_RCR(1) or SPI_RNCR(1).

1: The Receive Counter register has reached 0 since the last write in SPI_RCR(1) or SPI_RNCR(1).

- **ENDTX: End of TX Buffer (cleared by writing SPI_TCR or SPI_TNCR)**

0: The Transmit Counter register has not reached 0 since the last write in SPI_TCR(1) or SPI_TNCR(1).

1: The Transmit Counter register has reached 0 since the last write in SPI_TCR(1) or SPI_TNCR(1).

- **RXBUFF: RX Buffer Full (cleared by writing SPI_RCR or SPI_RNCR)**

0: SPI_RCR(1) or SPI_RNCR(1) has a value other than 0.

1: Both SPI_RCR(1) and SPI_RNCR(1) have a value of 0.

- **TXBUFE: TX Buffer Empty (cleared by writing SPI_TCR or SPI_TNCR)**

0: SPI_TCR(1) or SPI_TNCR(1) has a value other than 0.

1: Both SPI_TCR(1) and SPI_TNCR(1) have a value of 0.

- **NSSR: NSS Rising (cleared on read)**

0: No rising edge detected on NSS pin since the last read of SPI_SR.

1: A rising edge occurred on NSS pin since the last read of SPI_SR.

- **TXEMPTY: Transmission Registers Empty (cleared by writing SPI_TDR)**

0: As soon as data is written in SPI_TDR.

1: SPI_TDR and internal shift register are empty. If a transfer delay has been defined, TXEMPTY is set after the end of this delay.

- **UNDES: Underrun Error Status (Slave mode only) (cleared on read)**

0: No underrun has been detected since the last read of SPI_SR.

1: A transfer starts whereas no data has been loaded in SPI_TDR.

- **SPIENS: SPI Enable Status**

0: SPI is disabled.

1: SPI is enabled.

Note: 1. SPI_RCR, SPI_RNCR, SPI_TCR, SPI_TNCR are PDC registers.

34.8.6 SPI Interrupt Enable Register

Name: SPI_IER

Address: 0x40088014

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	UNDES	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **RDRF: Receive Data Register Full Interrupt Enable**
- **TDRE: SPI Transmit Data Register Empty Interrupt Enable**
- **MODF: Mode Fault Error Interrupt Enable**
- **OVRES: Overrun Error Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable**
- **ENDTX: End of Transmit Buffer Interrupt Enable**
- **RXBUFF: Receive Buffer Full Interrupt Enable**
- **TXBUFE: Transmit Buffer Empty Interrupt Enable**
- **NSSR: NSS Rising Interrupt Enable**
- **TXEMPTY: Transmission Registers Empty Enable**
- **UNDES: Underrun Error Interrupt Enable**

34.8.7 SPI Interrupt Disable Register

Name: SPI_IDR

Address: 0x40088018

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	UNDES	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Disables the corresponding interrupt.

- **RDRF: Receive Data Register Full Interrupt Disable**
- **TDRE: SPI Transmit Data Register Empty Interrupt Disable**
- **MODF: Mode Fault Error Interrupt Disable**
- **OVRES: Overrun Error Interrupt Disable**
- **ENDRX: End of Receive Buffer Interrupt Disable**
- **ENDTX: End of Transmit Buffer Interrupt Disable**
- **RXBUFF: Receive Buffer Full Interrupt Disable**
- **TXBUFE: Transmit Buffer Empty Interrupt Disable**
- **NSSR: NSS Rising Interrupt Disable**
- **TXEMPTY: Transmission Registers Empty Disable**
- **UNDES: Underrun Error Interrupt Disable**

34.8.8 SPI Interrupt Mask Register

Name: SPI_IMR

Address: 0x4008801C

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	UNDES	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

- **RDRF: Receive Data Register Full Interrupt Mask**
- **TDRE: SPI Transmit Data Register Empty Interrupt Mask**
- **MODF: Mode Fault Error Interrupt Mask**
- **OVRES: Overrun Error Interrupt Mask**
- **ENDRX: End of Receive Buffer Interrupt Mask**
- **ENDTX: End of Transmit Buffer Interrupt Mask**
- **RXBUFF: Receive Buffer Full Interrupt Mask**
- **TXBUFE: Transmit Buffer Empty Interrupt Mask**
- **NSSR: NSS Rising Interrupt Mask**
- **TXEMPTY: Transmission Registers Empty Mask**
- **UNDES: Underrun Error Interrupt Mask**

34.8.9 SPI Chip Select Register

Name: SPI_CSRx [x=0..3]

Address: 0x40088030

Access: Read/Write

31	30	29	28	27	26	25	24
DLYBCT							
23	22	21	20	19	18	17	16
DLYBS							
15	14	13	12	11	10	9	8
SCBR							
7	6	5	4	3	2	1	0
BITS				CSAAT	CSNAAT	NCPHA	CPOL

This register can only be written if the WPEN bit is cleared in the SPI Write Protection Mode Register.

Note: SPI_CSRx registers must be written even if the user wants to use the default reset values. The BITS field is not 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 Line 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 Line rises systematically after each transfer performed on the same slave. It remains inactive after the end of transfer for a minimal duration of:

$$\frac{DLYBCS}{f_{peripheral\ clock}}$$

(If field DLYBCS is lower than 6, a minimum of six periods is introduced.)

- **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 Line 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 bitmap.)

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 Bit Rate**

In Master mode, the SPI Interface uses a modulus counter to derive the SPCK bit rate from the peripheral clock. The bit rate is selected by writing a value from1 to 255 in the SCBR field. The following equation determines the SPCK bit rate:

SCBR = $f_{peripheral\ clock}$ / SPCK Bit Rate

Programming the SCBR field to 0 is forbidden. Triggering a transfer while SCBR is at 0 can lead to unpredictable results.

If BRSRCCLK = 1 in SPI_MR, SCBR must be programmed with a value greater than 1.

At reset, SCBR is 0 and the user has to program it at a valid value before performing the first transfer.

Note: If one of the SCBR fields in SPI_CSRx is set to 1, the other SCBR fields in SPI_CSRx must be set to 1 as well, if they are used 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 falling edge (activation) to the first valid SPCK transition.

When DLYBS = 0, the delay is half the SPCK clock period.

Otherwise, the following equation determines the delay:

DLYBS = Delay Before SPCK × $f_{peripheral\ clock}$

- **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 = 0, 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:

DLYBCT = Delay Between Consecutive Transfers × $f_{peripheral\ clock}$ / 32

34.8.10 SPI Write Protection Mode Register

Name: SPI_WPMR

Address: 0x400880E4

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 0x535049 (“SPI” in ASCII)

1: Enables the write protection if WPKEY corresponds to 0x535049 (“SPI” in ASCII)

See Section 34.7.5 “Register Write Protection” for the list of registers that can be write-protected.

- **WPKEY: Write Protection Key**

Value	Name	Description
0x535049	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

34.8.11 SPI Write Protection Status Register

Name: SPI_WPSR

Address: 0x400880E8

Access: Read-only

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

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of SPI_WPSR.

1: A write protection violation has occurred since the last read of SPI_WPSR. 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**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

35. Two-wire Interface (TWI)

35.1 Description

The Atmel Two-wire Interface (TWI) interconnects components on a unique two-wire bus, made up of one clock line and one data line with speeds of up to 400 Kbits per second, based on a byte-oriented transfer format. It can be used with any Atmel Two-wire Interface bus Serial EEPROM and I²C compatible device such as a Real Time Clock (RTC), Dot Matrix/Graphic LCD Controllers and temperature sensor. The TWI is programmable as a master or a slave with sequential or single-byte access. Multiple master capability is supported.

A configurable baud rate generator permits the output data rate to be adapted to a wide range of core clock frequencies.

Table 35-1 lists the compatibility level of the Atmel Two-wire Interface in Master mode and a full I²C compatible device.

Table 35-1. 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-bit 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/Synchronization	Supported
Multi Master Capability	Supported

Note: 1. START + b000000001 + Ack + Sr

35.2 Embedded Characteristics

- Compatible with Atmel Two-wire Interface Serial Memory and I²C Compatible Devices(1)
- One, Two or Three Bytes for Slave Address
- Sequential Read/Write Operations
- Master, Multi-master and Slave Mode Operation
- Bit Rate: Up to 400 Kbit/s
- General Call Supported in Slave Mode
- SMBus Quick Command Supported in Master Mode
- Connection to Peripheral DMA Controller (PDC) Channel Capabilities Optimizes Data Transfers
  - One Channel for the Receiver, One Channel for the Transmitter
- Register Write Protection

Note: 1. See Table 35-1 for details on compatibility with I²C Standard.

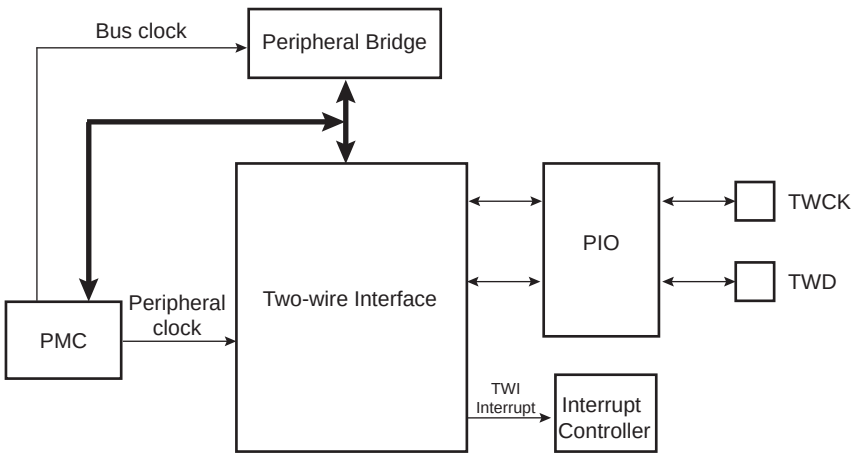
35.3 List of Abbreviations

Table 35-2. Abbreviations

Abbreviation	Description
TWI	Two-wire Interface
A	Acknowledge
NA	Non Acknowledge
P	Stop
S	Start
Sr	Repeated Start
SADR	Slave Address
ADR	Any address except SADR
R	Read
W	Write

35.4 Block Diagram

Figure 35-1. Block Diagram



35.5 I/O Lines Description

Table 35-3. I/O Lines Description

Name	Description	Type
TWD	Two-wire Serial Data (drives external serial data line – SDA)	Input/Output
TWCK	Two-wire Serial Clock (drives external serial clock line – SCL)	Input/Output

35.6 Product Dependencies

35.6.1 I/O Lines

Both TWD and TWCK are bidirectional lines, connected to a positive supply voltage via a current source or pull-up resistor. 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 user must program the PIO Controller to dedicate TWD and TWCK as peripheral lines.

The user must not program TWD and TWCK as open-drain. This is already done by the hardware.

Table 35-4. I/O Lines

Instance	Signal	I/O Line	Peripheral
TWI0	TWCK0	PA4	A
TWI0	TWD0	PA3	A
TWI1	TWCK1	PB5	A
TWI1	TWD1	PB4	A

35.6.2 Power Management

The TWI may be clocked through the Power Management Controller (PMC), thus the user must first configure the PMC to enable the TWI clock.

35.6.3 Interrupt Sources

The TWI has an interrupt line connected to the Interrupt Controller. In order to handle interrupts, the Interrupt Controller must be programmed before configuring the TWI.

Table 35-5. Peripheral IDs

Instance	ID
TWI0	17
TWI1	18

35.7 Functional Description

35.7.1 Transfer Format

The data put on the TWD line must be 8 bits long. Data is transferred MSB first; each byte must be followed by an acknowledgement. The number of bytes per transfer is unlimited (see Figure 35-3).

Each transfer begins with a START condition and terminates with a STOP condition (see Figure 35-2).

- 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 the STOP condition.

Figure 35-2. START and STOP Conditions

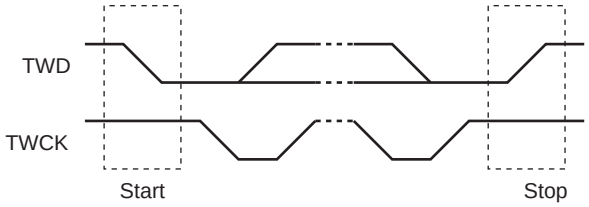
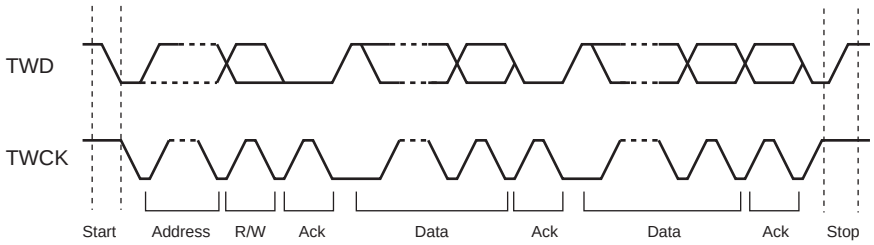


Figure 35-3. Transfer Format



35.7.2 Modes of Operation

The TWI has different modes of operations:

- Master transmitter mode
- Master receiver mode
- Multi-master transmitter mode
- Multi-master receiver mode
- Slave transmitter mode
- Slave receiver mode

These modes are described in the following sections.

35.7.3 Master Mode

35.7.3.1 Definition

The master is the device that starts a transfer, generates a clock and stops it.

35.7.3.2 Programming Master Mode

The following fields must be programmed before entering Master mode:

1. TWI_MMR.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. TWI_CWGR.CKDIV + CHDIV + CLDIV: Clock waveform.
3. TWI_CR.SVDIS: Disables the Slave mode
4. TWI_CR.MSEN: Enables the Master mode

Note: If the TWI is already in Master mode, the device address (DADR) can be configured without disabling the Master mode.

35.7.3.3 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. If the slave does not acknowledge the byte, then the Not Acknowledge flag (NACK) is set in the TWI Status Register (TWI_SR) of the master and a STOP condition is sent. The NACK flag must be cleared by reading the TWI Status Register (TWI_SR) before the next write into the TWI Transmit Holding Register (TWI_THR). 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 (SCL) is tied low. When new data is written in the TWI_THR, the TWCK/SCL is released and the data is sent. Setting the STOP bit in TWI_CR generates a STOP condition.

After a master write transfer, the SCL is stretched (tied low) as long as no new data is written in the TWI_THR or until a STOP command is performed. See Figure 35-4, Figure 35-5, and Figure 35-6.

To clear the TXRDY flag, first set the bit TWI_CR.MSDIS, then set the bit TWI_CR.MSEN.

Figure 35-4. Master Write with One Data Byte

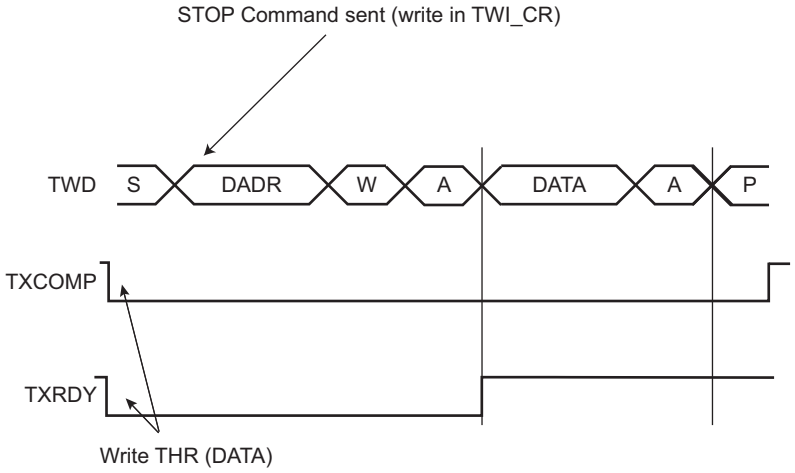


Figure 35-5. Master Write with Multiple Data Bytes

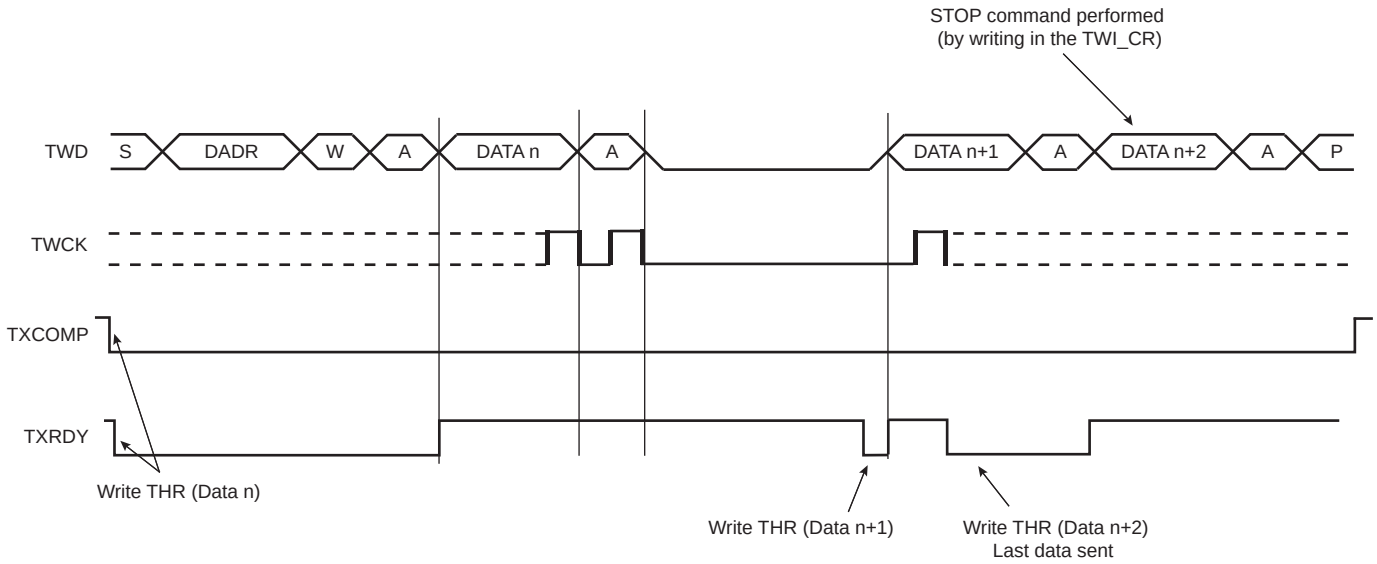
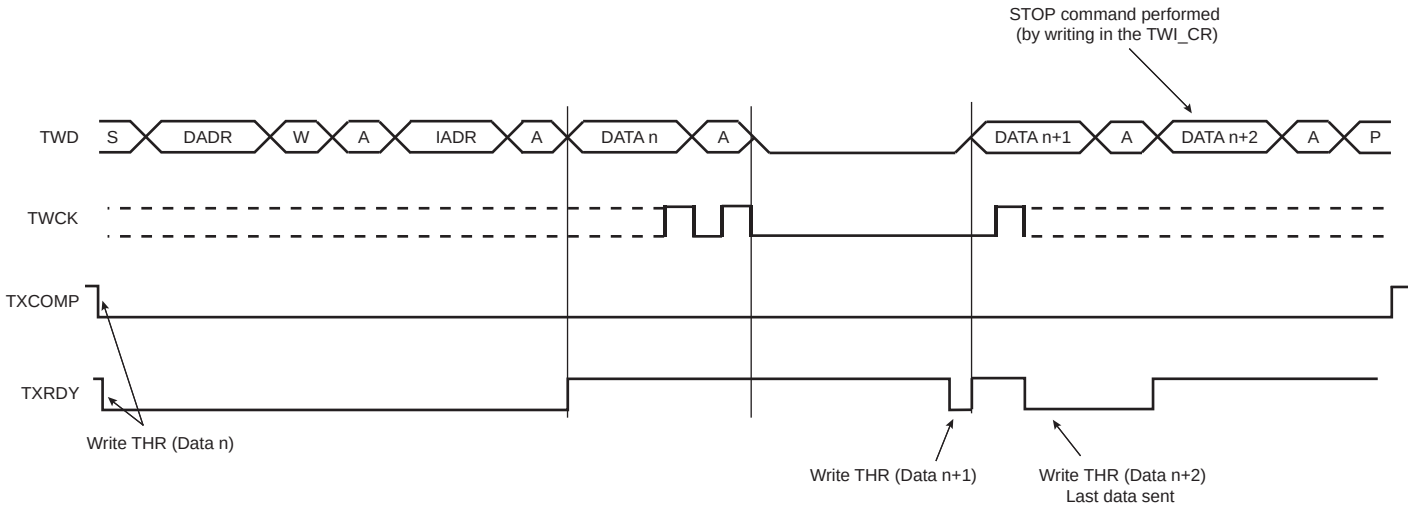


Figure 35-6. Master Write with One Byte Internal Address and Multiple Data Bytes



35.7.3.4 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 TWI_SR 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. See Figure 35-7. When the RXRDY bit is set in the TWI_SR, a character has been received in the Receive Holding Register (TWI_RHR). The RXRDY bit is reset when reading the TWI_RHR.

RXRDY is used as Receive Ready for the PDC receive channel.

When a single data byte read is performed, with or without internal address (IADR), the START and STOP bits must be set at the same time. See Figure 35-7. When a multiple data byte read is performed, with or without internal address (IADR), the STOP bit must be set after the next-to-last data received. See Figure 35-8. For internal address usage, see Section 35.7.3.5.

If the Receive Holding Register (TWI_RHR) is full (RXRDY high) and the master is receiving data, the serial clock line is tied low before receiving the last bit of the data and until the TWI_RHR is read. Once the TWI_RHR is read, the master stops stretching the serial clock line and ends the data reception. See Figure 35-9.

Warning: When receiving multiple bytes in Master read mode, if the next-to-last access is not read (the RXRDY flag remains high), the last access is not completed until TWI_RHR is read. The last access stops on the next-to-last bit. When the TWI_RHR is read, the STOP bit command must be sent within a period of half a bit only, otherwise another read access might occur (spurious access).

A possible workaround is to set the STOP bit before reading the TWI_RHR on the next-to-last access (within the interrupt handler).

Figure 35-7. Master Read with One Data Byte

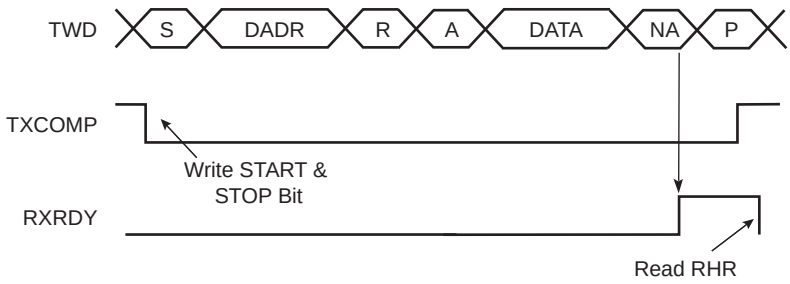


Figure 35-8. Master Read with Multiple Data Bytes

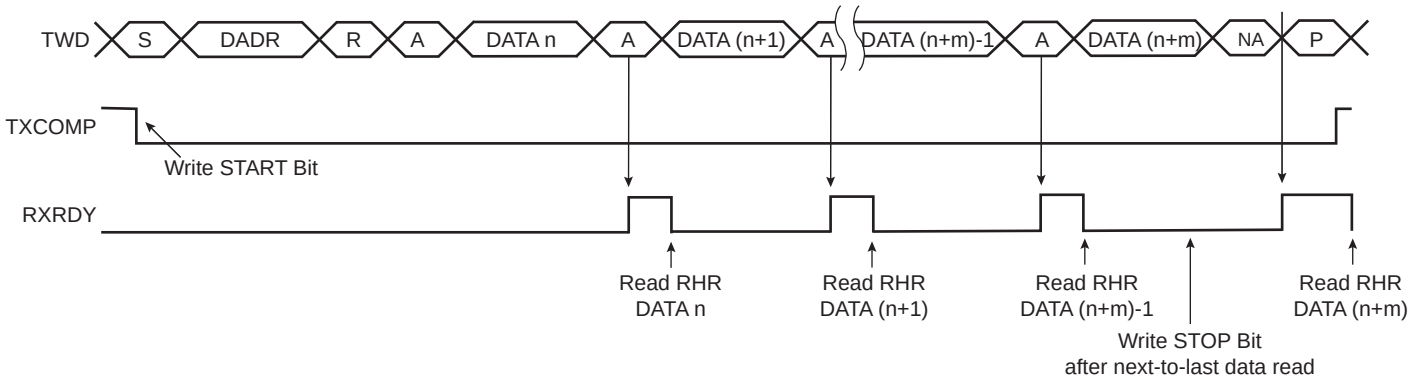
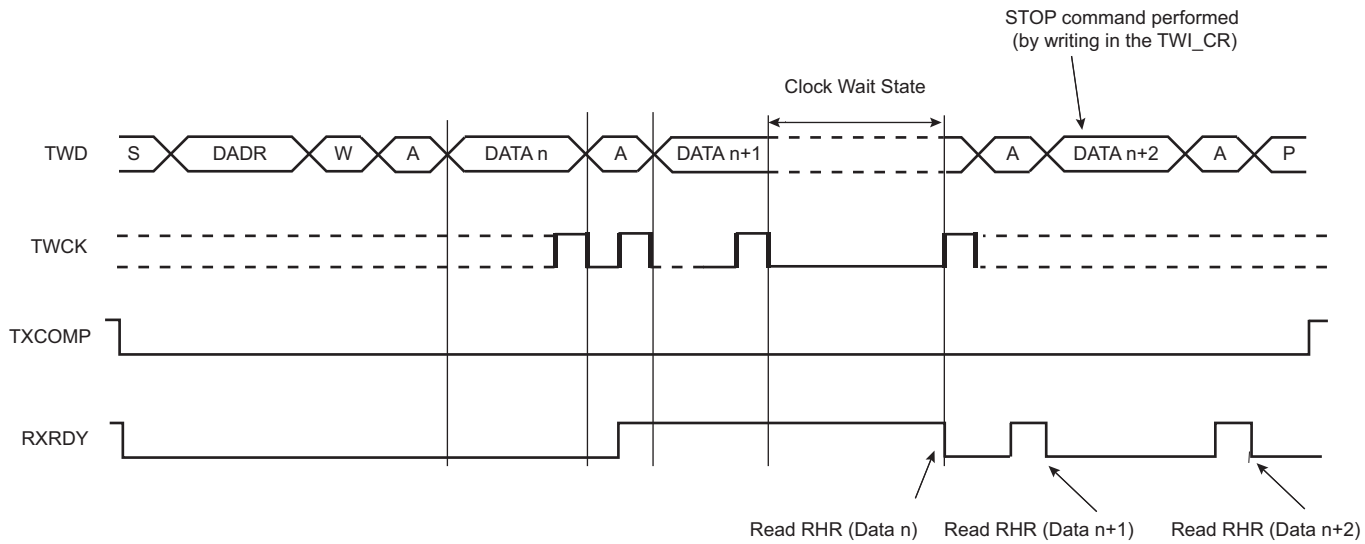


Figure 35-9. Master Read Wait State with Multiple Data Bytes



35.7.3.5 Internal Address

The TWI can perform transfers with 7-bit slave address devices and 10-bit slave address devices.

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, e.g. within a memory page location in a serial memory. When performing read operations with an internal address, the TWI performs a write operation to set the internal address into the slave device, and then switch to Master receiver mode. Note that the second START condition (after

sending the IADR) is sometimes called “repeated start” (Sr) in I^{2}C fully-compatible devices. See Figure 35-11. See Figure 35-10 and Figure 35-12 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.

Table 35-6 shows the abbreviations used in Figure 35-10 and Figure 35-11.

Table 35-6. Abbreviations

Abbreviation	Definition
S	Start
Sr	Repeated Start
P	Stop
W	Write
R	Read
A	Acknowledge
NA	Not Acknowledge
DADR	Device Address
IADR	Internal Address

Figure 35-10. Master Write with One, Two or Three Bytes Internal Address and One Data Byte

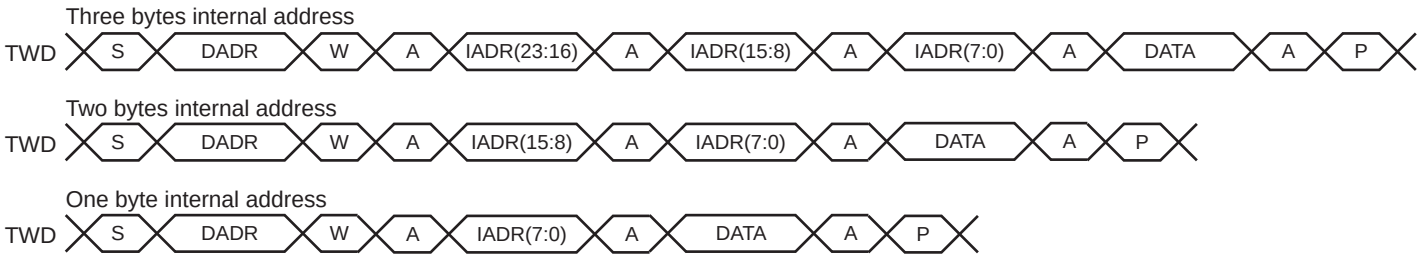
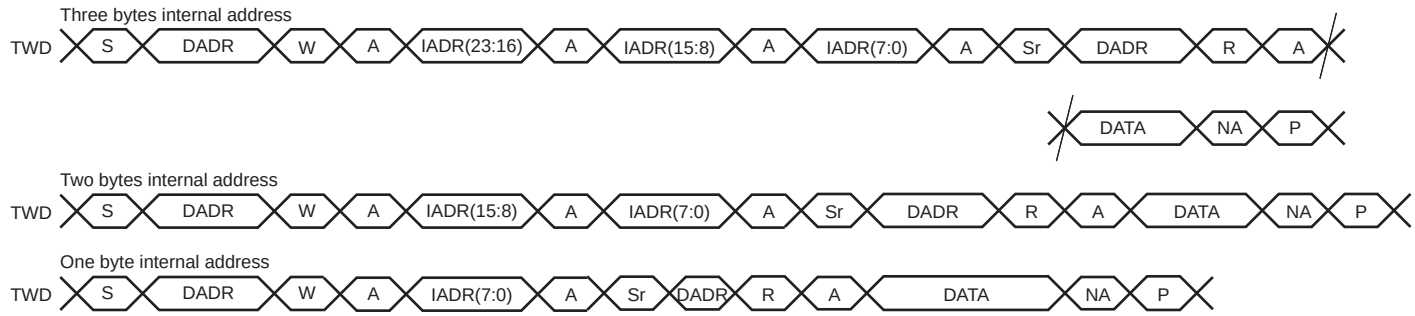


Figure 35-11. Master Read with One, Two or Three Bytes Internal Address and One Data Byte



10-bit Slave Addressing

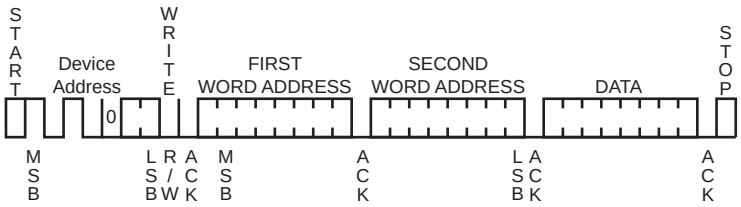
For a slave address higher than seven bits, the user must configure the address size **(**IADRSZ) and set the other slave address bits in the Internal Address register (TWI_IADR). The two remaining internal address bytes, IADR[15:8] and IADR[23:16] can be used the same way as in 7-bit slave addressing.

Example: Address a 10-bit device (10-bit device address is b1 b2 b3 b4 b5 b6 b7 b8 b9 b10)

1. Program IADRSZ = 1,
2. Program DADR with 1 1 1 1 0 b1 b2 (b1 is the MSB of the 10-bit address, b2, etc.)
3. Program TWI_IADR with b3 b4 b5 b6 b7 b8 b9 b10 (b10 is the LSB of the 10-bit address)

Figure 35-12 below shows a byte write to a memory device. This demonstrates the use of internal addresses to access the device.

Figure 35-12. Internal Address Usage



35.7.3.6 Using the Peripheral DMA Controller (PDC)

The use of the PDC significantly reduces the CPU load.

To ensure correct implementation, proceed as follows.

Data Transmit with the PDC

1. Initialize the transmit PDC (memory pointers, transfer size - 1).
2. Configure the master (DADR, CKDIV, MREAD = 0, etc.)
3. Start the transfer by setting the PDC TXTEN bit.
4. Wait for the PDC ENDTX Flag either by using the polling method or ENDTX interrupt.
5. Disable the PDC by setting the PDC TXTDIS bit.
6. Wait for the TXRDY flag in TWI_SR.
7. Set the STOP bit in TWI_CR.
8. Write the last character in TWI_THR.
9. (Only if peripheral clock must be disabled) Wait for the TXCOMP flag to be raised in TWI_SR.

Data Receive with the PDC

The PDC transfer size must be defined with the buffer size minus 2. The two remaining characters must be managed without PDC to ensure that the exact number of bytes are received regardless of system bus latency conditions encountered during the end of buffer transfer period.

In Slave mode, the number of characters to receive must be known in order to configure the PDC.

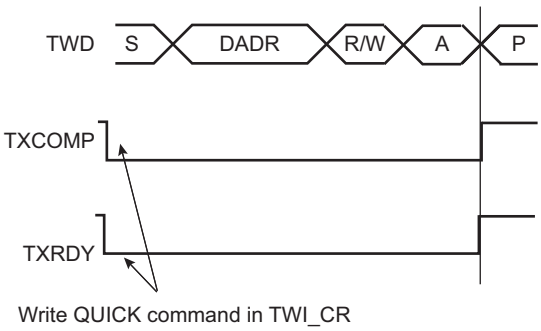
1. Initialize the receive PDC (memory pointers, transfer size - 2).
2. Configure the master (DADR, CKDIV, MREAD = 1, etc.)
3. Set the PDC RXTEN bit.
4. (Master Only) Write the START bit in the TWI_CR to start the transfer.
5. Wait for the PDC ENDRX Flag either by using polling method or ENDRX interrupt.
6. Disable the PDC by setting the PDC RXTDIS bit.
7. Wait for the RXRDY flag in TWI_SR.
8. Set the STOP bit in TWI_CR.
9. Read the penultimate character in TWI_RHR.
10. Wait for the RXRDY flag in TWI_SR.
11. Read the last character in TWI_RHR.
12. (Only if peripheral clock must be disabled) Wait for the TXCOMP flag to be raised in TWI_SR.

35.7.3.7 SMBus Quick Command (Master Mode Only)

The TWI can perform a quick command:

1. Configure the Master mode (DADR, CKDIV, etc.).
2. Write the MREAD bit in the TWI_MMR at the value of the one-bit command to be sent.
3. Start the transfer by setting the QUICK bit in the TWI_CR.

Figure 35-13. SMBus Quick Command



35.7.3.8 Read/Write Flowcharts

The flowcharts in the following figures provide examples of read and write operations. A polling or interrupt method can be used to check the status bits. The interrupt method requires that the Interrupt Enable Register (TWI_IER) be configured first.

Figure 35-14. TWI Write Operation with Single Data Byte without Internal Address

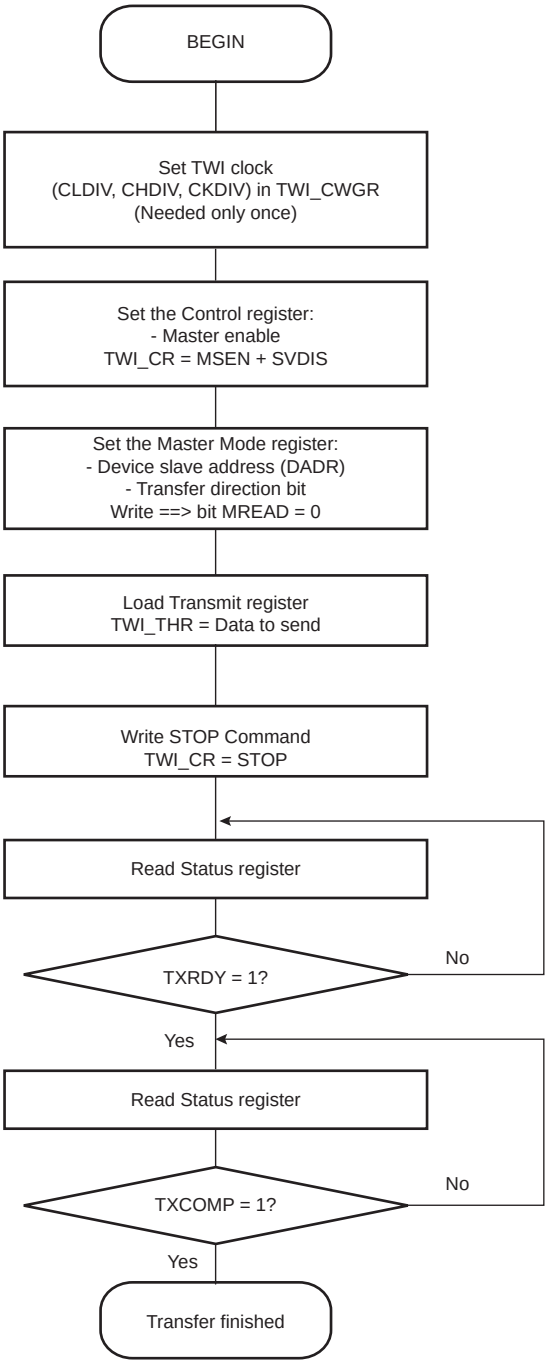


Figure 35-15. TWI Write Operation with Single Data Byte and Internal Address

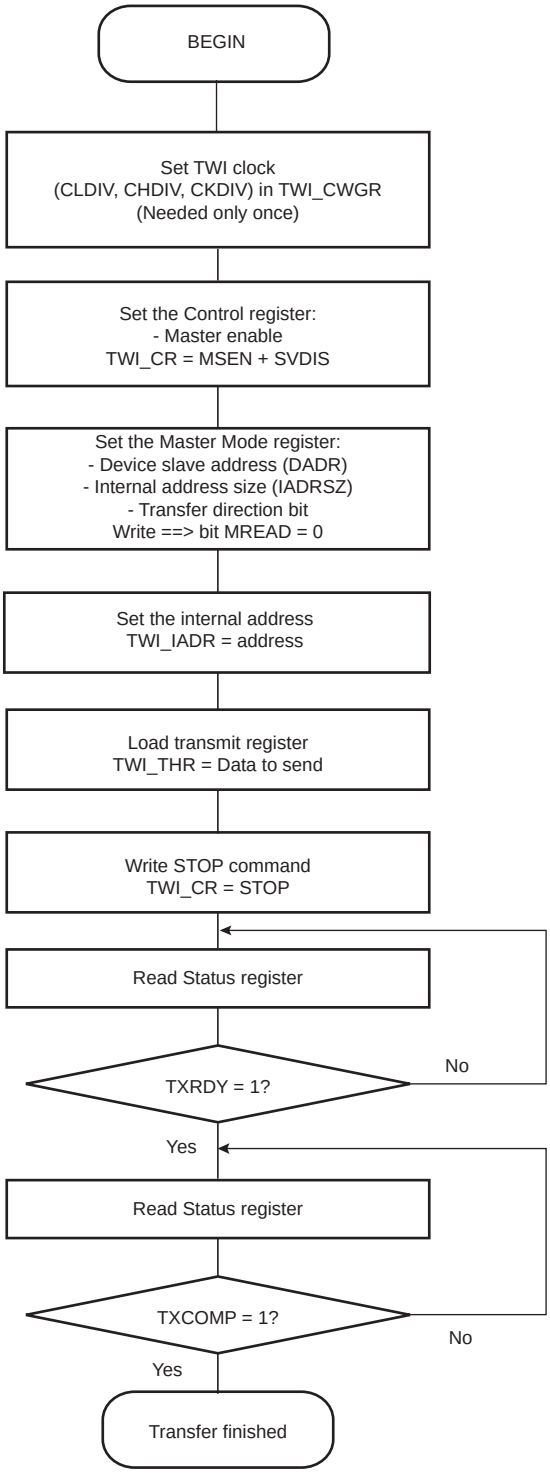


Figure 35-16. TWI Write Operation with Multiple Data Bytes with or without Internal Address

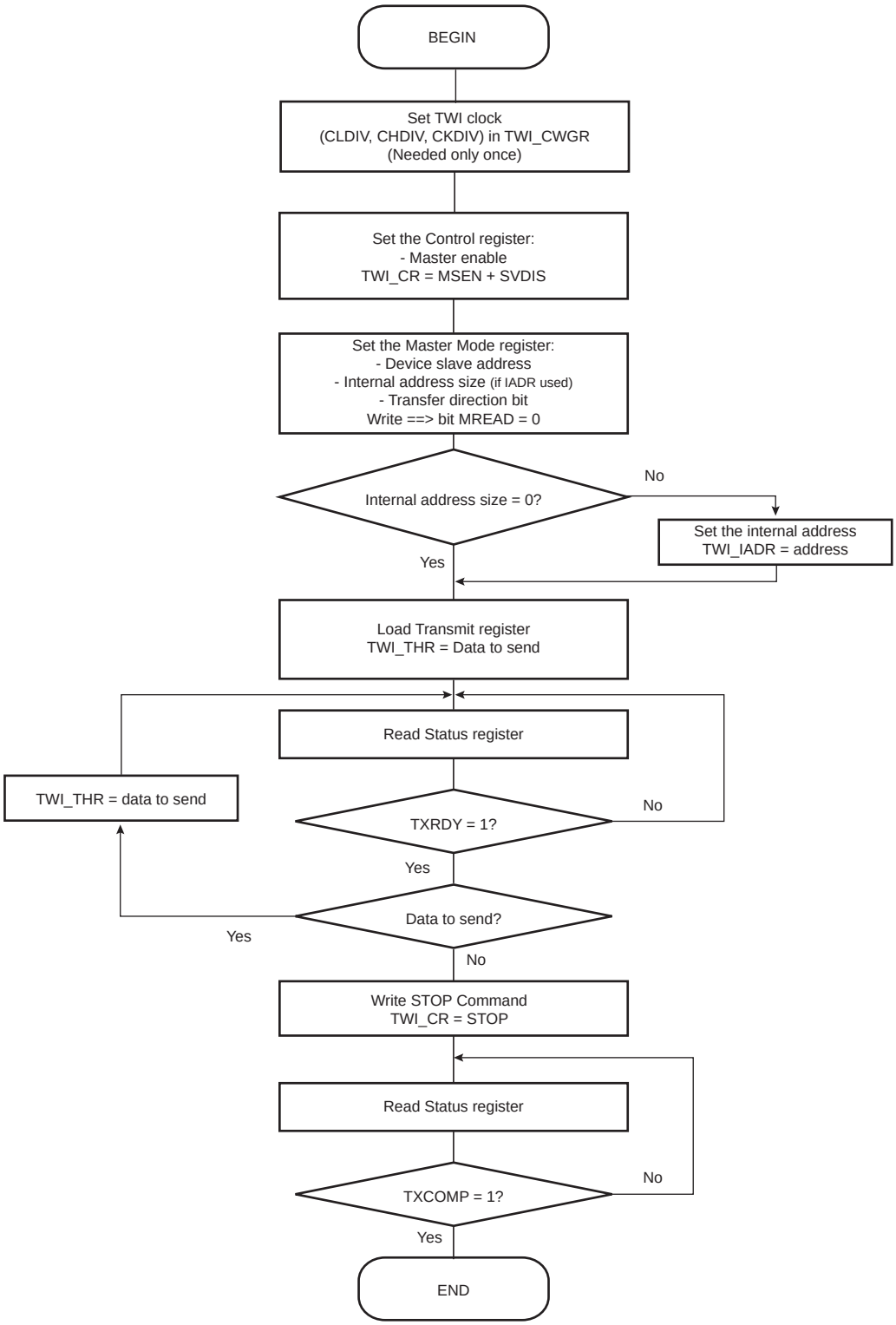


Figure 35-17. TWI Read Operation with Single Data Byte without Internal Address

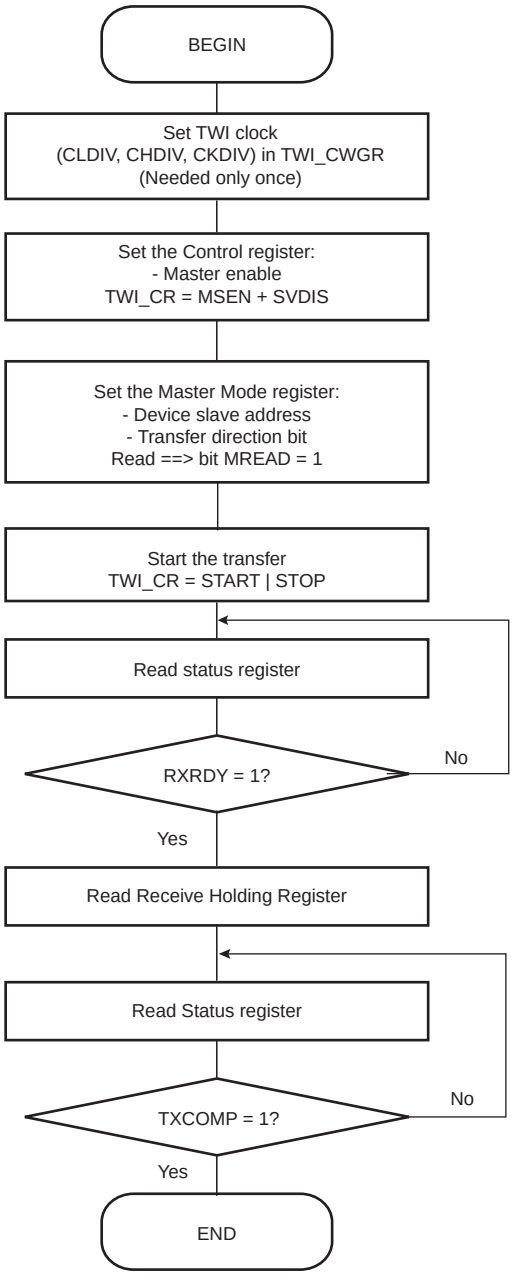


Figure 35-18. TWI Read Operation with Single Data Byte and Internal Address

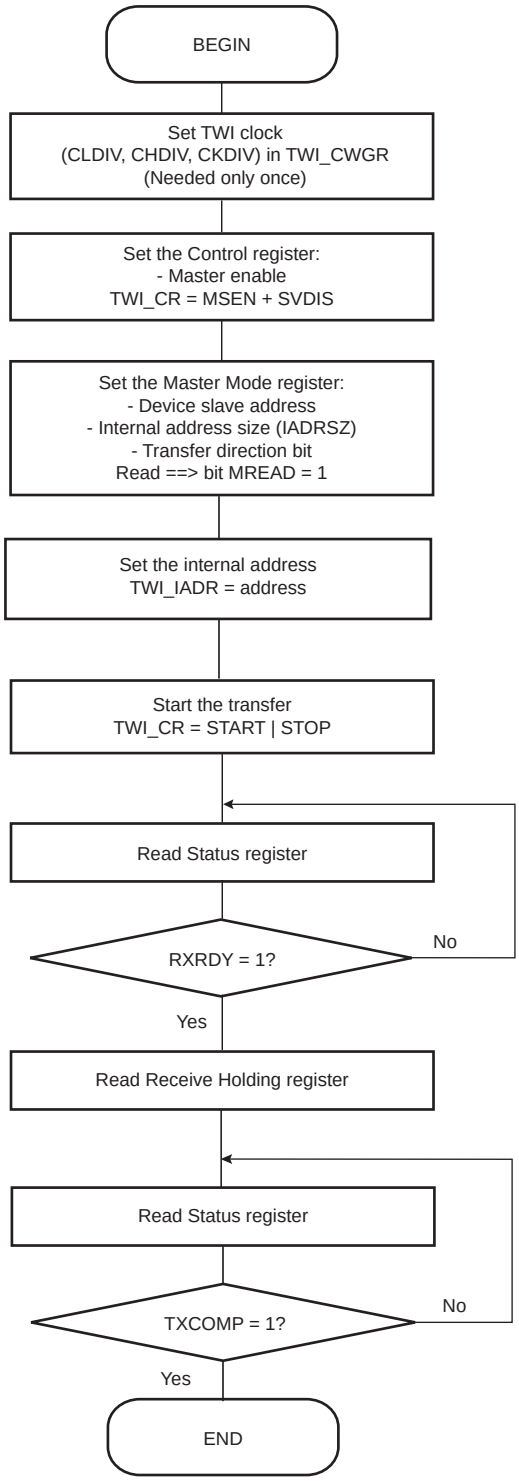
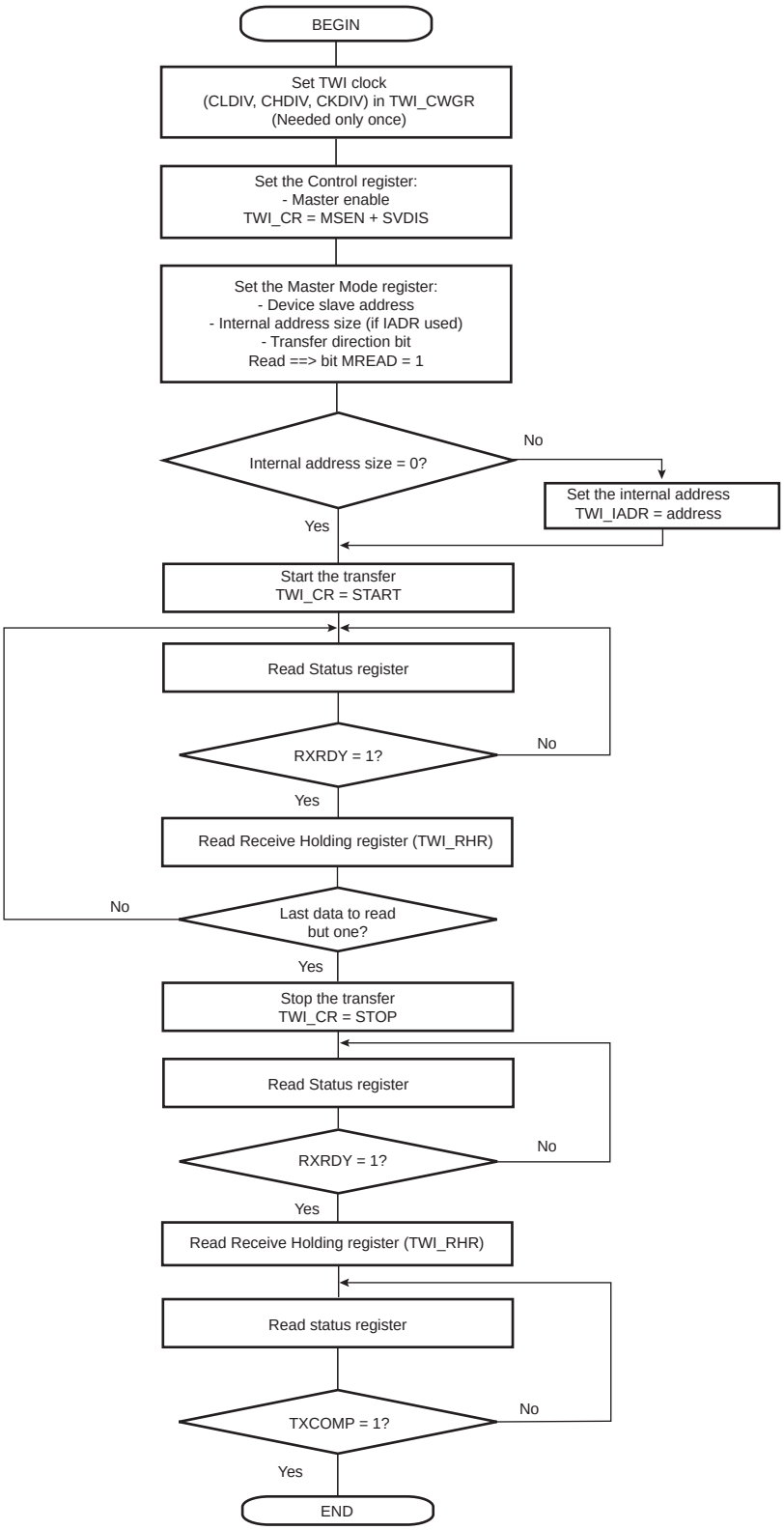


Figure 35-19. TWI Read Operation with Multiple Data Bytes with or without Internal Address



35.7.4 Multi-master Mode

35.7.4.1 Definition

In Multi-master mode, 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 a master lose arbitration, it stops sending data and listens to the bus in order to detect a stop. When the stop is detected, the master may put its data on the bus by performing arbitration.

Arbitration is illustrated in Figure 35-21.

35.7.4.2 Two 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: Arbitration is supported in both Multi-master modes.

TWI as Master Only

In this mode, TWI is considered as a Master only (MSEN is always one) and must be driven like a Master with the ARBLST (Arbitration Lost) flag in addition.

If arbitration is lost (ARBLST = 1), the user must reinitiate the data transfer.

If the user starts a transfer (ex.: DADR + START + W + Write in THR) and if the bus is busy, the TWI automatically waits for a STOP condition on the bus to initiate the transfer (see Figure 35-20).

Note: The state of the bus (busy or free) is not shown in the user interface.

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 user must manage the pseudo Multi-master mode described in the steps below.

1. Program TWI in Slave mode (SADR + MSDIS + SVEN) and perform a slave access (if TWI is addressed).
2. If the TWI has to be set in Master mode, wait until the TXCOMP flag is at 1.
3. Program the Master mode (DADR + SVDIS + MSEN) and start the transfer (ex: START + Write in THR).
4. As soon as the Master mode is enabled, the TWI scans the bus in order to detect if it is busy or free. When the bus is considered 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 case the Master that won the arbitration is required to access the TWI.
7. If the TWI has to be set in Slave mode, wait until TXCOMP flag is at 1 and then program the Slave mode.

Note: If the arbitration is lost and the TWI is addressed, the TWI will not acknowledge even if it is programmed in Slave mode as soon as ARBLST is set to 1. Then the Master must repeat SADR.

Figure 35-20. Programmer Sends Data While the Bus is Busy

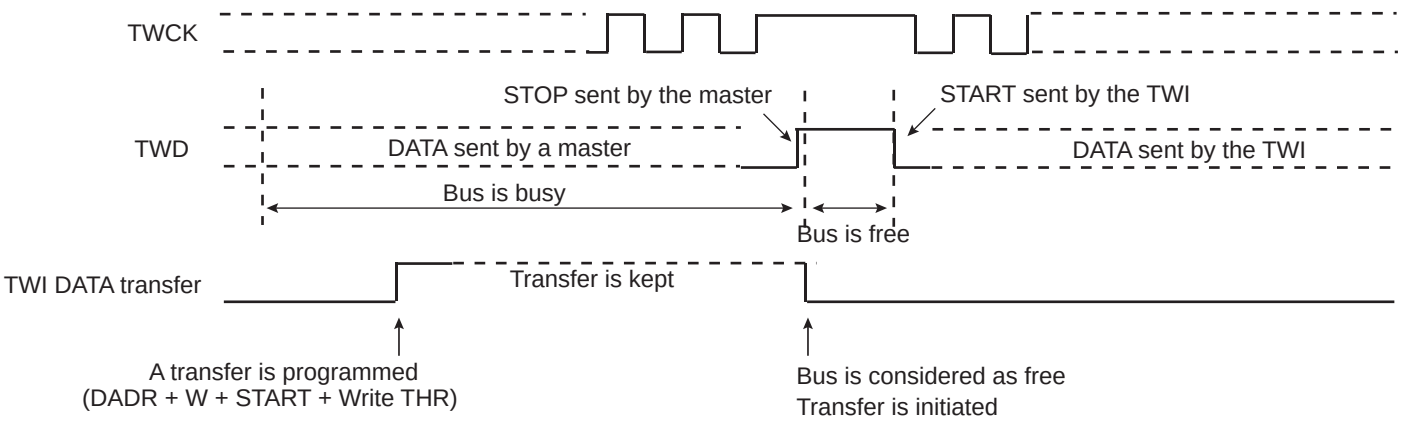
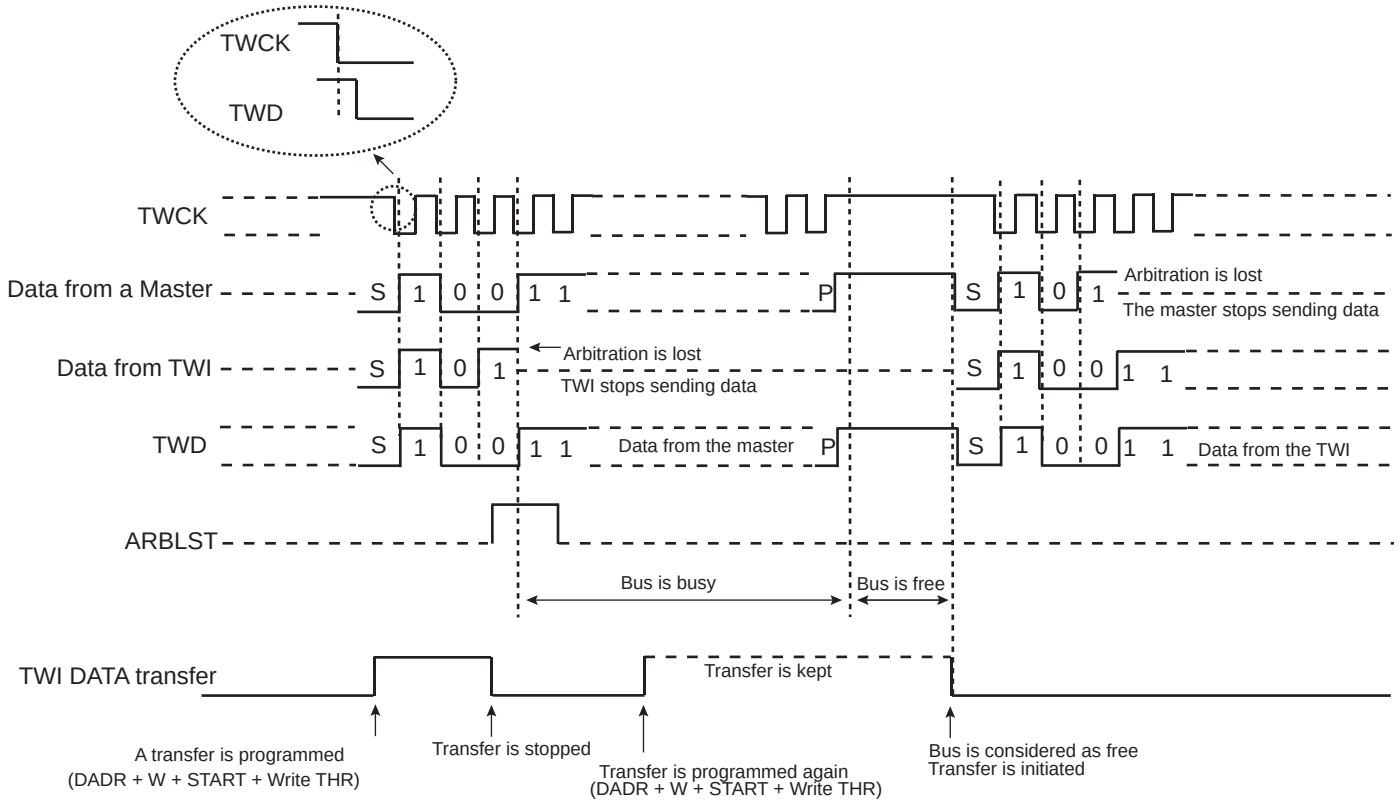
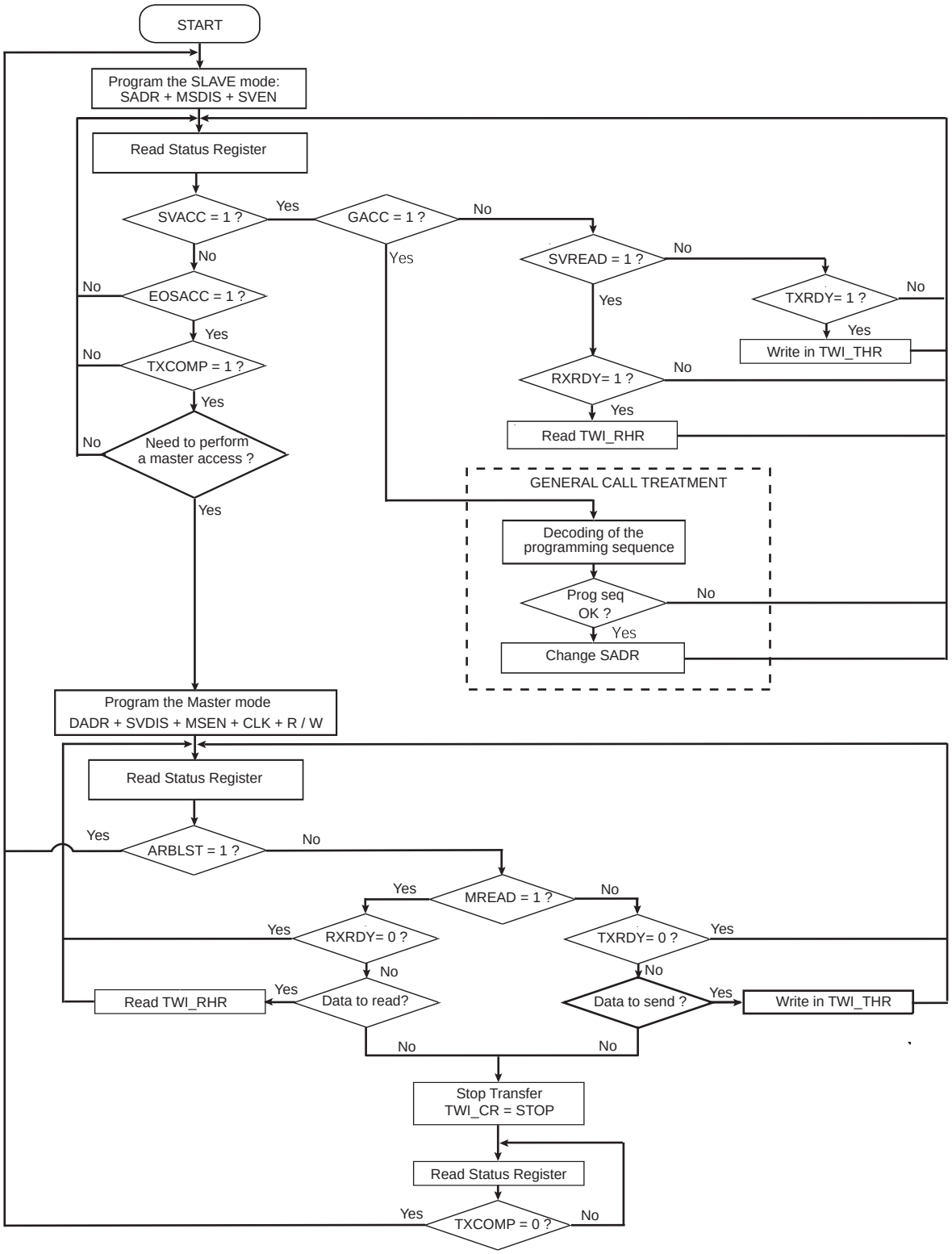


Figure 35-21. Arbitration Cases



The flowchart shown in Figure 35-22 gives an example of read and write operations in Multi-master mode.

Figure 35-22. Multi-master Flowchart



35.7.5 Slave Mode

35.7.5.1 Definition

Slave mode is defined as a mode where the device receives the clock and the address from another device called the master.

In this mode, the device never initiates and never completes the transmission (START, REPEATED START and STOP conditions are always provided by the master).

35.7.5.2 Programming Slave Mode

The following fields must be programmed before entering Slave mode:

1. TWI_SMR.SADR: The slave device address is used in order to be accessed by master devices in Read or Write mode.
2. TWI_CR.MSDIS: Disables the Master mode.
3. TWI_CR.SVEN: Enables the Slave mode.

As the device receives the clock, values written in TWI_CWGR are ignored.

35.7.5.3 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, the EOSACC (End Of Slave Access) flag is set.

Read Sequence

In the case of a read sequence (SVREAD is high), TWI transfers data written in TWI_THR (TWI Transmit Holding Register) until a STOP condition or a REPEATED_START and 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 TWI_THR, the TXRDY (Transmit Holding Register Ready) flag is reset, and it is set when the internal shifter 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.

To clear the TXRDY flag, first set the bit TWI_CR.SVDIS, then set the bit TWI_CR.SVEN.

See Figure 35-23.

Write Sequence

In the case of a write sequence (SVREAD is low), the RXRDY (Receive Holding Register Ready) flag is set as soon as a character has been received in the TWI_RHR (TWI Receive Holding Register). RXRDY is reset when reading the TWI_RHR.

TWI continues receiving data until a STOP condition or a REPEATED_START + an address different from SADR is detected. Note that at the end of the write sequence TXCOMP flag is set and SVACC reset.

See Figure 35-24.

Clock Synchronization Sequence

If TWI_RHR is not read in time, the TWI performs a clock synchronization.

Clock synchronization information is given by the bit SCLWS (Clock Wait State).

See Figure 35-27.

Clock Stretching Sequence

If TWI_THR is not written in time, the TWI performs a clock stretching.

Clock stretching information is given by the bit SCLWS (Clock Wait State).

See Figure 35-26.

General Call

In the case where a GENERAL CALL is performed, the GACC (General Call Access) flag is set.

After GACC is set, the user must interpret the meaning of the GENERAL CALL and decode the new address programming sequence.

See Figure 35-25.

35.7.5.4 Data Transfer

Read Operation

The Read mode is defined as a data requirement from the master.

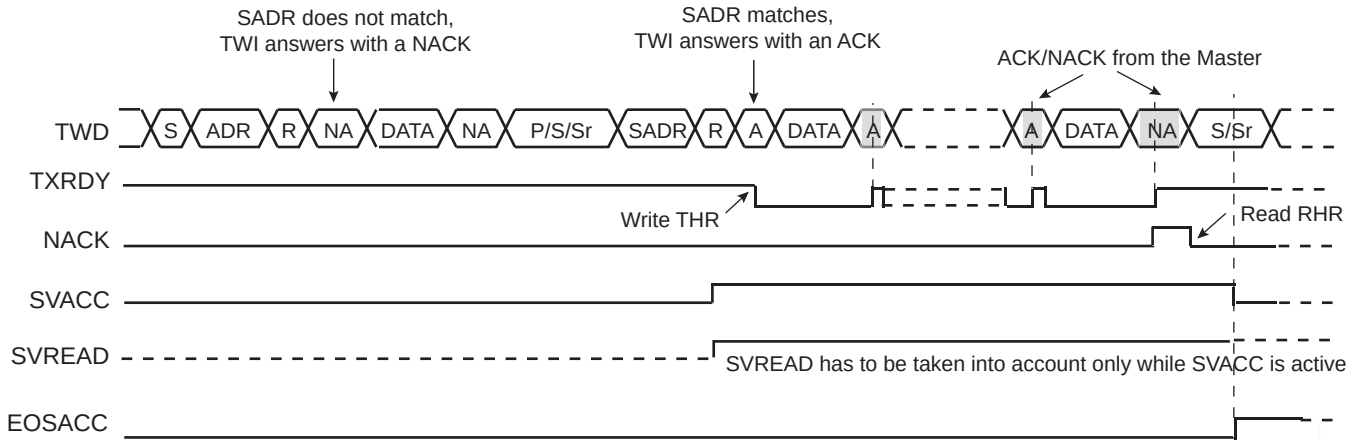
After a START or a REPEATED START condition is detected, the decoding of the address starts. If the slave address (SADR) is decoded, SVACC is set and SVREAD indicates the direction of the transfer.

Until a STOP or REPEATED START condition is detected, TWI continues sending data loaded in the TWI_THR.

If a STOP condition or a REPEATED START + an address different from SADR is detected, SVACC is reset.

Figure 35-23 describes the write operation.

Figure 35-23. 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 internal shifter and set when this data has been acknowledged or non acknowledged.

Write Operation

The Write mode is defined as a data transmission from the master.

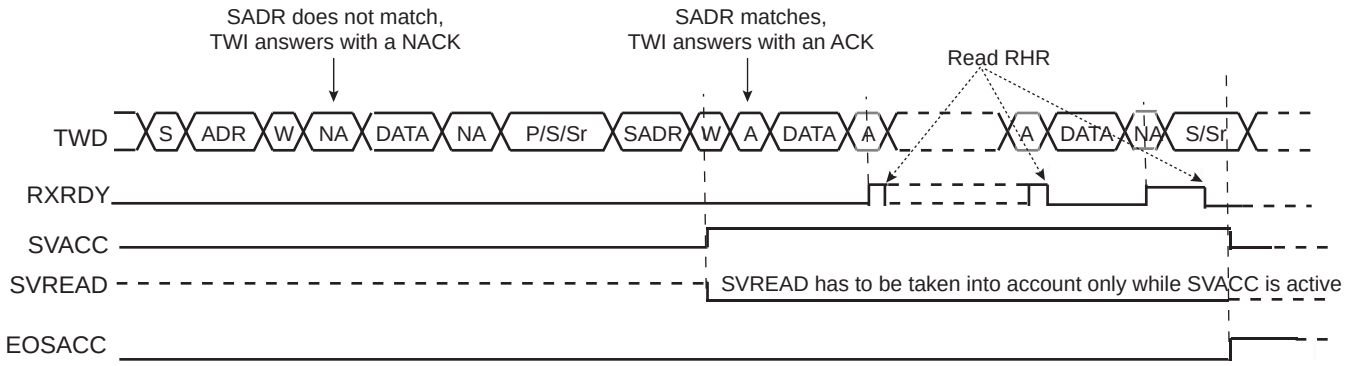
After a START or a REPEATED START, the decoding of the address starts. If the slave address is decoded, SVACC is set and SVREAD indicates the direction of the transfer (SVREAD is low in this case).

Until a STOP or REPEATED START condition is detected, TWI stores the received data in the TWI_RHR.

If a STOP condition or a REPEATED START + an address different from SADR is detected, SVACC is reset.

Figure 35-24 describes the write operation.

Figure 35-24. 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 internal shifter to the TWI_RHR and reset when this data is read.

General Call

The general call is performed in order to change the address of the slave.

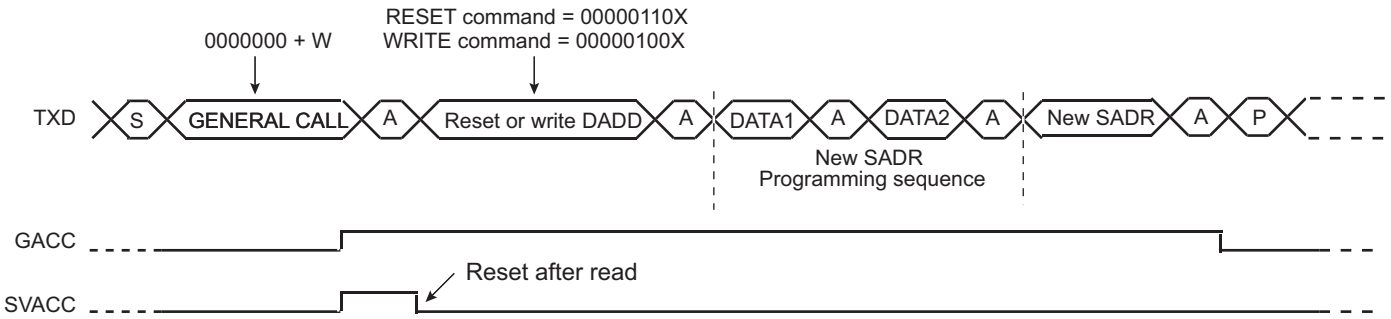
If a GENERAL CALL is detected, GACC is set.

After the detection of GENERAL CALL, it is up to the programmer to decode the commands which come afterwards.

In case of a WRITE command, the programmer has to decode the programming sequence and program a new SADR if the programming sequence matches.

Figure 35-25 describes the GENERAL CALL access.

Figure 35-25. Master Performs a General Call



Note: This method allows the user to create a personal programming sequence by choosing the programming bytes and the number of them. The programming sequence has to be provided to the master.

Clock Synchronization/Stretching

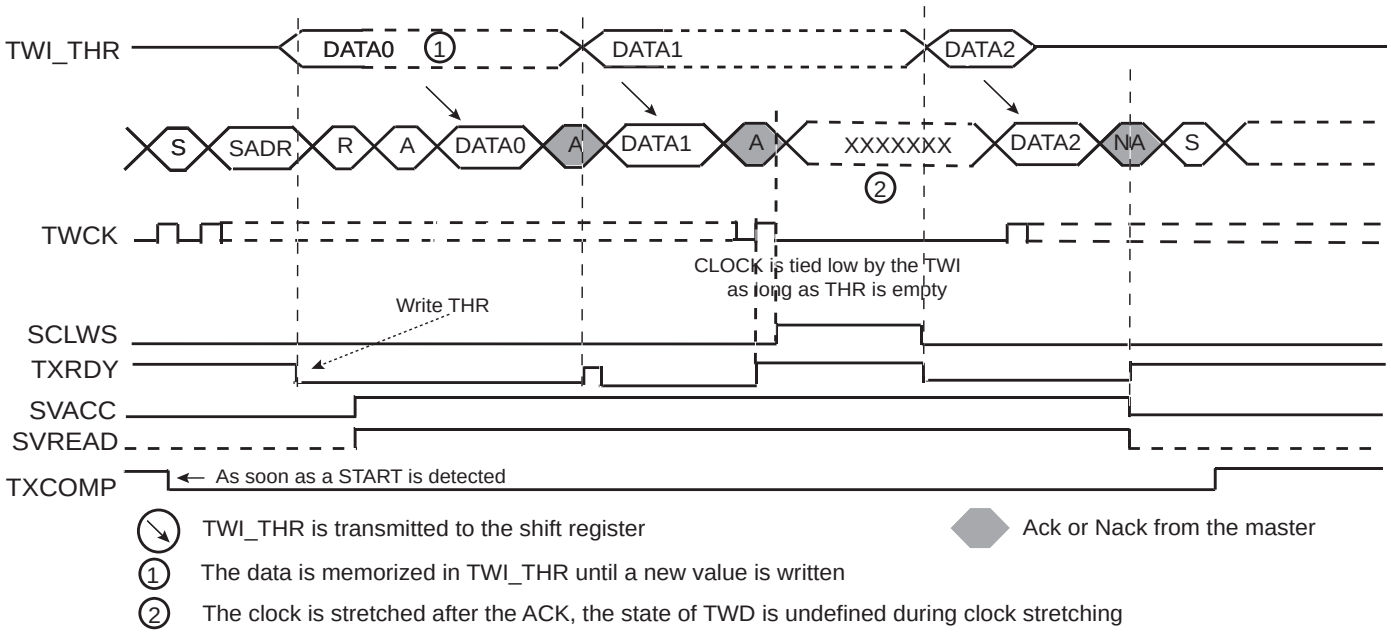
In both Read and Write modes, it may occur that TWI_THR/TWI_RHR buffer is not filled /emptied before transmission/reception of a new character. In this case, to avoid sending/receiving undesired data, a clock stretching/synchronization mechanism is implemented.

Clock Stretching in Read Mode

The clock is tied low during the acknowledge phase if the internal shifter is empty and if a STOP or REPEATED START condition was not detected. It is tied low until the internal shifter is loaded.

Figure 35-26 describes clock stretching in Read mode.

Figure 35-26. Clock Stretching in Read Mode



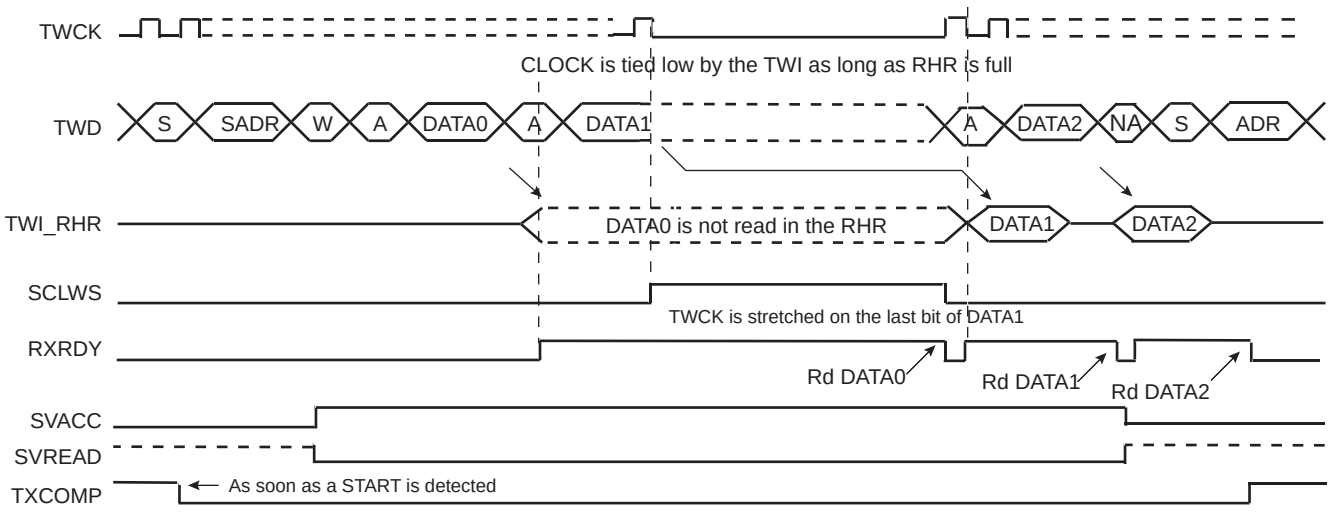
Notes:
1. TXRDY is reset when data has been written in the TWI_THR to the internal shifter and set when this data has been acknowledged or non acknowledged.
2. At the end of the read sequence, TXCOMP is set after a STOP or after a REPEATED_START + an address different from SADR.
3. SCLWS is automatically set when the clock stretching mechanism is started.

Clock Synchronization in Write Mode

The clock is tied low outside of the acknowledge phase if the internal shifter and the TWI_RHR is full. If a STOP or REPEATED_START condition was not detected, it is tied low until TWI_RHR is read.

Figure 35-27 describes the clock synchronization in Write mode.

Figure 35-27. 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.

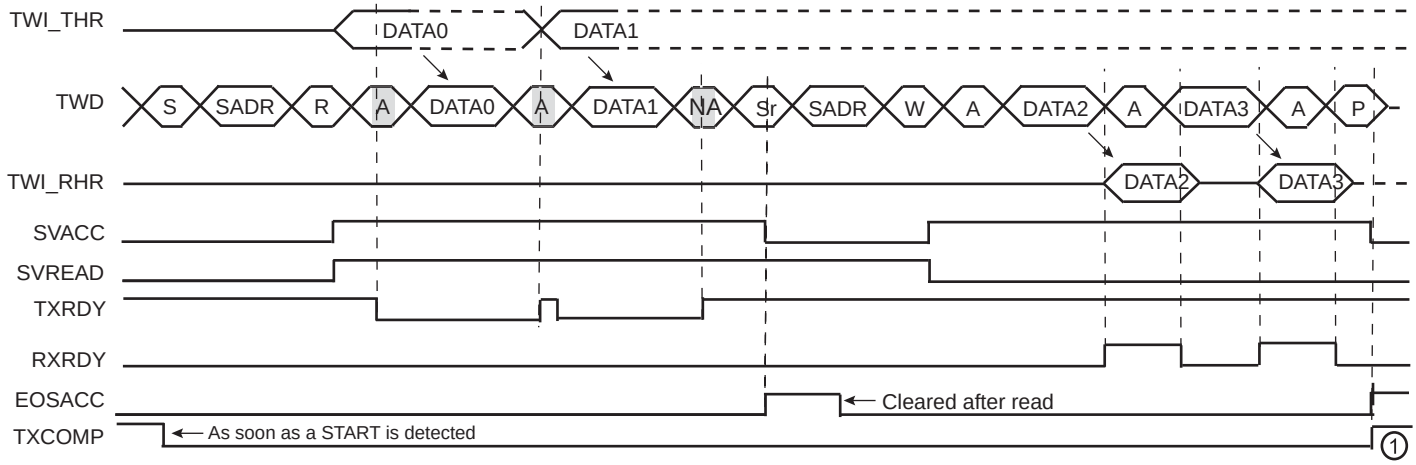
Reversal After a Repeated Start

Reversal of Read to Write

The master initiates the communication by a read command and finishes it by a write command.

Figure 35-28 describes the repeated start + reversal from Read to Write mode.

Figure 35-28. Repeated Start + Reversal from Read to Write Mode



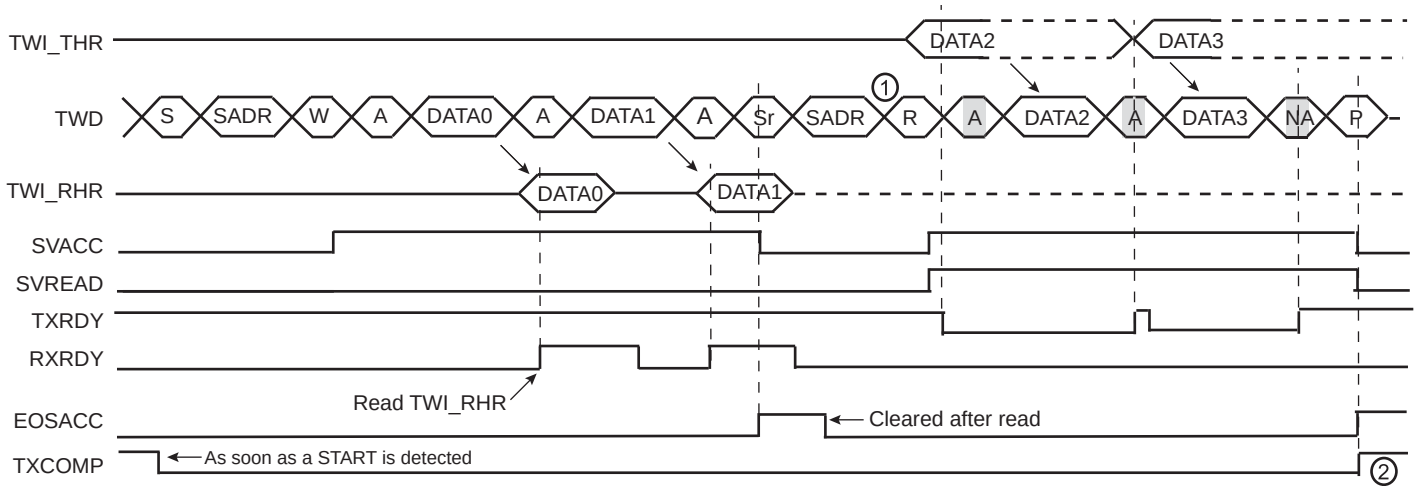
Note: 1. TXCOMP is only set at the end of the transmission because after the repeated start, SADR is detected again.

Reversal of Write to Read

The master initiates the communication by a write command and finishes it by a read command.

Figure 35-29 describes the repeated start + reversal from Write to Read mode.

Figure 35-29. Repeated Start + Reversal from Write to Read Mode



Notes: 1. In this case, if TWI_THR has not been written at the end of the read command, the clock is automatically stretched before the ACK.
2. TXCOMP is only set at the end of the transmission because after the repeated start, SADR is detected again.

35.7.5.5 Using the Peripheral DMA Controller (PDC) in Slave Mode

The use of the PDC significantly reduces the CPU load.

Data Transmit with the PDC in Slave Mode

The following procedure shows an example of data transmission with PDC.

1. Initialize the transmit PDC (memory pointers, transfer size).
2. Start the transfer by setting the PDC TXTEN bit.
3. Wait for the PDC ENDTX flag by using either the polling method or the ENDTX interrupt.
4. Disable the PDC by setting the PDC TXTDIS bit.
5. (Only if peripheral clock must be disabled) Wait for the TXCOMP flag to be raised in TWI_SR.

Data Receive with the PDC in Slave Mode

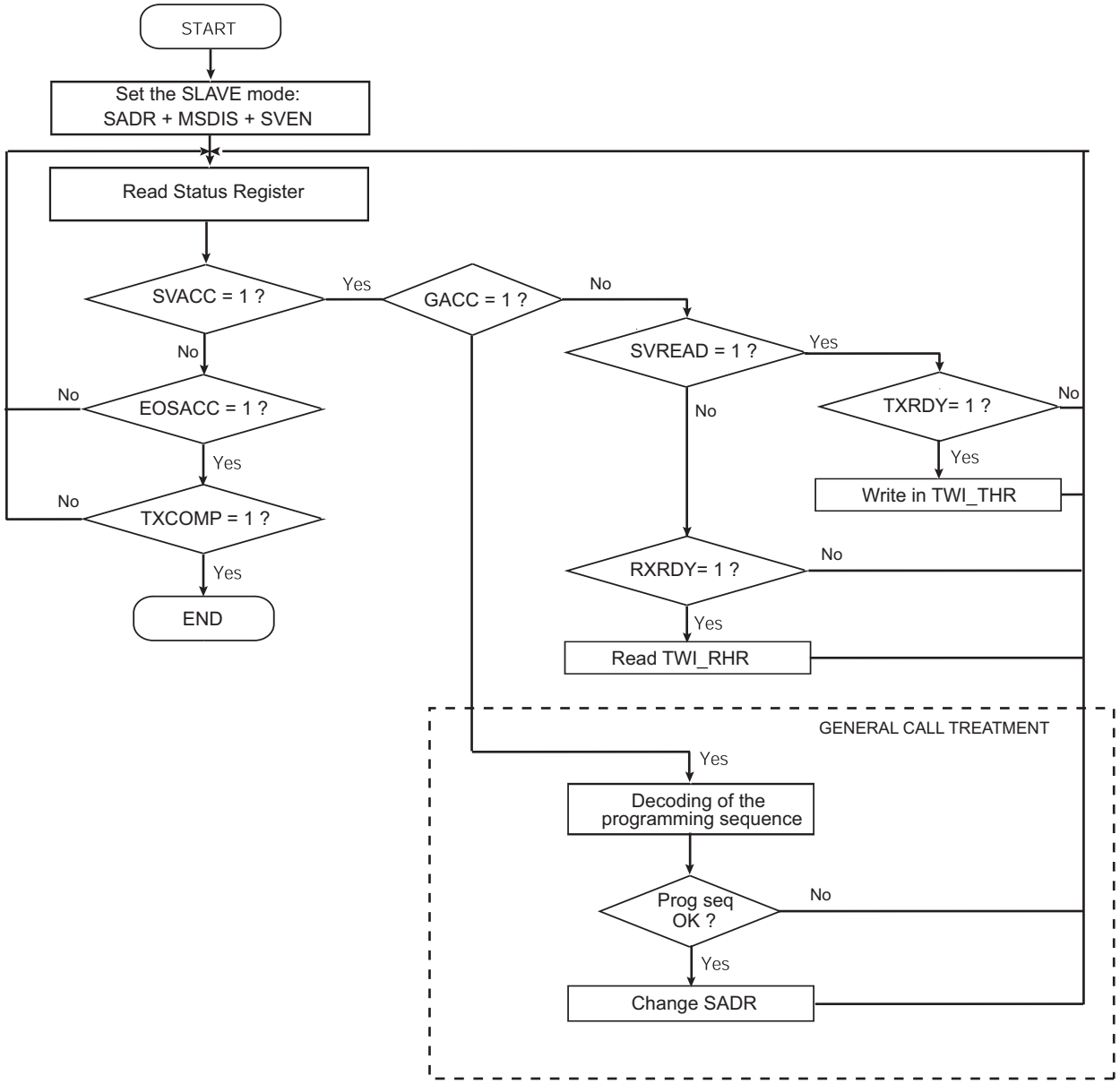
The following procedure shows an example of data transmission with PDC where the number of characters to be received is known.

1. Initialize the receive PDC (memory pointers, transfer size).
2. Set the PDC RXTEN bit.
3. Wait for the PDC ENDRX flag by using either the polling method or the ENDRX interrupt.
4. Disable the PDC by setting the PDC RXTDIS bit.
5. (Only if peripheral clock must be disabled) Wait for the TXCOMP flag to be raised in TWI_SR.

35.7.5.6 Read Write Flowcharts

The flowchart shown in Figure 35-30 gives an example of read and write operations in Slave mode. A polling or interrupt method can be used to check the status bits. The interrupt method requires that the Interrupt Enable Register (TWI_IER) be configured first.

Figure 35-30. Read Write Flowchart in Slave Mode



35.7.6 Register Write Protection

To prevent any single software error from corrupting TWI behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the TWI Write Protection Mode Register (TWI_WPMR).

If a write access to a write-protected register is detected, the WPVS flag in the TWI Write Protection Status Register (TWI_WPSR) is set and the WPVSRC field shows the register in which the write access has been attempted.

The WPVS bit is automatically cleared after reading the TWI_WPSR.

The following registers can be write-protected:

- TWI Slave Mode Register
- TWI Clock Waveform Generator Register

35.8 Two-wire Interface (TWI) User Interface

Table 35-7. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	TWI_CR	Write-only	–
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	–
0x28	Interrupt Disable Register	TWI_IDR	Write-only	–
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	–
0x38–0xE0	Reserved	–	–	–
0xE4	Write Protection Mode Register	TWI_WPMR	Read/Write	0x00000000
0xE8	Write Protection Status Register	TWI_WPSR	Read-only	0x00000000
0xEC–0xFC	Reserved	–	–	–
0x100–0x128	Reserved for PDC registers	–	–	–

Note: All unlisted offset values are considered as “reserved”.

35.8.1 TWI Control Register

Name: TWI_CR

Address: 0x400A8000 (0), 0x400AC000 (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
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 TWI Master Mode Register (TWI_MMR).

This action is necessary for the TWI 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 is sent when transmission of the current data has ended.

- **MSEN: TWI Master Mode Enabled**

0: No effect.

1: Enables the Master mode (MSDIS must be written to 0).

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: Enables the Slave mode (SVDIS must be written to 0)

Note: Switching from master to Slave mode is only permitted when TXCOMP = 1.

- **SVDIS: TWI Slave Mode Disabled**

0: No effect.

1: The Slave mode is disabled. The shifter and holding characters (if it contains data) are transmitted in case of read operation. In write operation, the character being transferred must be completely received before disabling.

- **QUICK: SMBus Quick Command**

0: No effect.

1: If Master mode is enabled, a SMBus Quick Command is sent.

- **SWRST: Software Reset**

0: No effect.

1: Equivalent to a system reset.

35.8.2 TWI Master Mode Register

Name: TWI_MMR

Address: 0x400A8004 (0), 0x400AC004 (1)

Access: Read/Write

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. These bits are only used in Master mode.

35.8.3 TWI Slave Mode Register

Name: TWI_SMR

Address: 0x400A8008 (0), 0x400AC008 (1)

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	SADR						

15	14	13	12	11	10	9	8
–	–	–	–	–	–		

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

This register can only be written if the WPEN bit is cleared in the TWI Write Protection Mode Register.

- **SADR: Slave Address**

The slave device address is used in Slave mode in order to be accessed by master devices in Read or Write mode.

SADR must be programmed before enabling the Slave mode or after a general call. Writes at other times have no effect.

35.8.4 TWI Internal Address Register

Name: TWI_IADR

Address: 0x400A800C (0), 0x400AC00C (1)

Access: Read/Write

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.

35.8.5 TWI Clock Waveform Generator Register

Name: TWI_CWGR

Address: 0x400A8010 (0), 0x400AC010 (1)

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	CKDIV		

15	14	13	12	11	10	9	8
CHDIV							

7	6	5	4	3	2	1	0
CLDIV							

This register can only be written if the WPEN bit is cleared in the TWI Write Protection Mode Register.

TWI_CWGR is only used in Master mode.

- **CLDIV: Clock Low Divider**

The TWCK low period is defined as follows: $t_{low} = ((CLDIV \times 2^{CKDIV}) + 4 \times t_{peripheral\ clock}$

- **CHDIV: Clock High Divider**

The TWCK high period is defined as follows: $t_{high} = ((CHDIV \times 2^{CKDIV}) + 4 \times t_{peripheral\ clock}$

- **CKDIV: Clock Divider**

The CKDIV field is used to increase both TWCK high and low periods.

35.8.6 TWI Status Register

Name: TWI_SR

Address: 0x400A8020 (0), 0x400AC020 (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
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 (cleared by writing TWI_THR)**

TXCOMP used in Master mode:

0: During the length of the current frame.

1: When both holding register and internal shifter are empty and STOP condition has been sent.

TXCOMP behavior in Master mode can be seen in Figure 35-6 and in Figure 35-8.

TXCOMP used in Slave mode:

0: As soon as a START is detected.

1: After a STOP or a REPEATED START + an address different from SADR is detected.

TXCOMP behavior in Slave mode can be seen in Figure 35-26, Figure 35-27, Figure 35-28 and Figure 35-29.

- **RXRDY: Receive Holding Register Ready (cleared by reading TWI_RHR)**

0: No character has been received since the last TWI_RHR read operation.

1: A byte has been received in the TWI_RHR since the last read.

RXRDY behavior in Master mode can be seen in Figure 35-8.

RXRDY behavior in Slave mode can be seen in Figure 35-24, Figure 35-27, Figure 35-28 and Figure 35-29.

- **TXRDY: Transmit Holding Register Ready (cleared by writing TWI_THR)**

TXRDY used in Master mode:

0: The transmit holding register has not been transferred into internal shifter. Set to 0 when writing into TWI_THR.

1: As soon as a data byte is transferred from TWI_THR to internal shifter or if a NACK error is detected, TXRDY is set at the same time as TXCOMP and NACK. TXRDY is also set when MSEN is set (enable TWI).

TXRDY behavior in Master mode can be seen in Figure 35.7.3.3.

TXRDY used in Slave mode:

0: As soon as data is written in the TWI_THR, until this data has been transmitted and acknowledged (ACK or NACK).

1: It indicates that the TWI_THR is empty and that data has been transmitted and acknowledged.

If TXRDY is high and if a NACK has been detected, the transmission will be stopped. Thus when TRDY = NACK = 1, the programmer must not fill TWI_THR to avoid losing it.

TXRDY behavior in Slave mode can be seen in Figure 35-23, Figure 35-26, Figure 35-28 and Figure 35-29.

- **SVREAD: Slave Read**

This bit is only used in Slave mode. When SVACC is low (no Slave access has been detected) SVREAD is irrelevant.

0: Indicates that a write access is performed by a Master.

1: Indicates that a read access is performed by a Master.

SVREAD behavior can be seen in Figure 35-23, Figure 35-24, Figure 35-28 and Figure 35-29.

- **SVACC: Slave Access**

This bit is only used in Slave mode.

0: TWI is not addressed. SVACC is automatically cleared after a NACK or a STOP condition is detected.

1: Indicates that the address decoding sequence has matched (A Master has sent SADR). SVACC remains high until a NACK or a STOP condition is detected.

SVACC behavior can be seen in Figure 35-23, Figure 35-24, Figure 35-28 and Figure 35-29.

- **GACC: General Call Access (cleared on read)**

This bit is only used in Slave mode.

0: No General Call has been detected.

1: A General Call has been detected. After the detection of General Call, if need be, the programmer may acknowledge this access and decode the following bytes and respond according to the value of the bytes.

GACC behavior can be seen in Figure 35-25.

- **OVRE: Overrun Error (cleared 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 (cleared 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 or an address 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 that means that the Master will stop the data transfer or reinitiate it.

Note that in Slave write mode all data are acknowledged by the TWI.

- **ARBLST: Arbitration Lost (cleared 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**

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 transmission / reception of a new character.

SCLWS behavior can be seen in Figure 35-26 and Figure 35-27.

- **EOSACC: End Of Slave Access (cleared on read)**

This bit is only used in Slave mode.

0: A slave access is being performed.

1: The Slave access is finished. End Of Slave Access is automatically set as soon as SVACC is reset.

EOSACC behavior can be seen in Figure 35-28 and Figure 35-29.

- **ENDRX: End of RX buffer (cleared by writing TWI_RCR or TWI_RNCR)**

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 (cleared by writing TWI_TCR or TWI_TNCR)**

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 (cleared by writing TWI_RCR or TWI_RNCR)**

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 (cleared by writing TWI_TCR or TWI_TNCR)**

0: TWI_TCR or TWI_TNCR have a value other than 0.

1: Both TWI_TCR and TWI_TNCR have a value of 0.

35.8.7 TWI Interrupt Enable Register

Name: TWI_IER

Address: 0x400A8024 (0), 0x400AC024 (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
TXBUFE	RXBUFF	ENDTX	ENDRX	EOSACC	SCL_WS	ARBLST	NACK
7	6	5	4	3	2	1	0
–	OVRE	GACC	SVACC	–	TXRDY	RXRDY	TXCOMP

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **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**

35.8.8 TWI Interrupt Disable Register

Name: TWI_IDR

Address: 0x400A8028 (0), 0x400AC028 (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
TXBUFE	RXBUFF	ENDTX	ENDRX	EOSACC	SCL_WS	ARBLST	NACK
7	6	5	4	3	2	1	0
–	OVRE	GACC	SVACC	–	TXRDY	RXRDY	TXCOMP

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Disables the corresponding interrupt.

- **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**

35.8.9 TWI Interrupt Mask Register

Name: TWI_IMR

Address: 0x400A802C (0), 0x400AC02C (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
TXBUFE	RXBUFF	ENDTX	ENDRX	EOSACC	SCL_WS	ARBLST	NACK

7	6	5	4	3	2	1	0
–	OVRE	GACC	SVACC	–	TXRDY	RXRDY	TXCOMP

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is disabled.

1: The corresponding interrupt is enabled.

- **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**

35.8.10 TWI Receive Holding Register

Name: TWI_RHR

Address: 0x400A8030 (0), 0x400AC030 (1)

Access: Read-only

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

- **RXDATA: Master or Slave Receive Holding Data**

35.8.11 TWI Transmit Holding Register

Name: TWI_THR

Address: 0x400A8034 (0), 0x400AC034 (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
TXDATA							

- **TXDATA: Master or Slave Transmit Holding Data**

35.8.12 TWI Write Protection Mode Register

Name: TWI_WPMR

Address: 0x400A80E4 (0), 0x400AC0E4 (1)

Access: Read/Write

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

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x545749 (“TWI” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x545749 (“TWI” in ASCII).

See Section 35.7.6 “Register Write Protection” for the list of registers that can be write-protected.

- **WPKEY: Write Protection Key**

Value	Name	Description
0x545749	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0

35.8.13 TWI Write Protection Status Register

Name: TWI_WPSR

Address: 0x400A80E8 (0), 0x400AC0E8 (1)

Access: Read-only

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

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of the TWI_WPSR.

1: A write protection violation has occurred since the last read of the TWI_WPSR. If this violation is an unauthorized attempt to write a protected register, the violation is reported into field WPVSRC.

- **WPVSRC: Write Protection Violation Source**

When WPVS = 1, WPVSRC shows the register address offset at which a write access has been attempted.

36. Universal Asynchronous Receiver Transmitter (UART)

36.1 Description

The Universal Asynchronous Receiver Transmitter (UART) features a two-pin UART that can be used for communication and trace purposes and offers an ideal medium for in-situ programming solutions.

Moreover, the association with a peripheral DMA controller (PDC) permits packet handling for these tasks with processor time reduced to a minimum.

36.2 Embedded Characteristics

- Two-pin UART
  - Independent Receiver and Transmitter with a Common Programmable Baud Rate Generator
  - Even, Odd, Mark or Space Parity Generation
  - Parity, Framing and Overrun Error Detection
  - Automatic Echo, Local Loopback and Remote Loopback Channel Modes
  - Interrupt Generation
  - Support for Two PDC Channels with Connection to Receiver and Transmitter

36.3 Block Diagram

Figure 36-1. UART Block Diagram

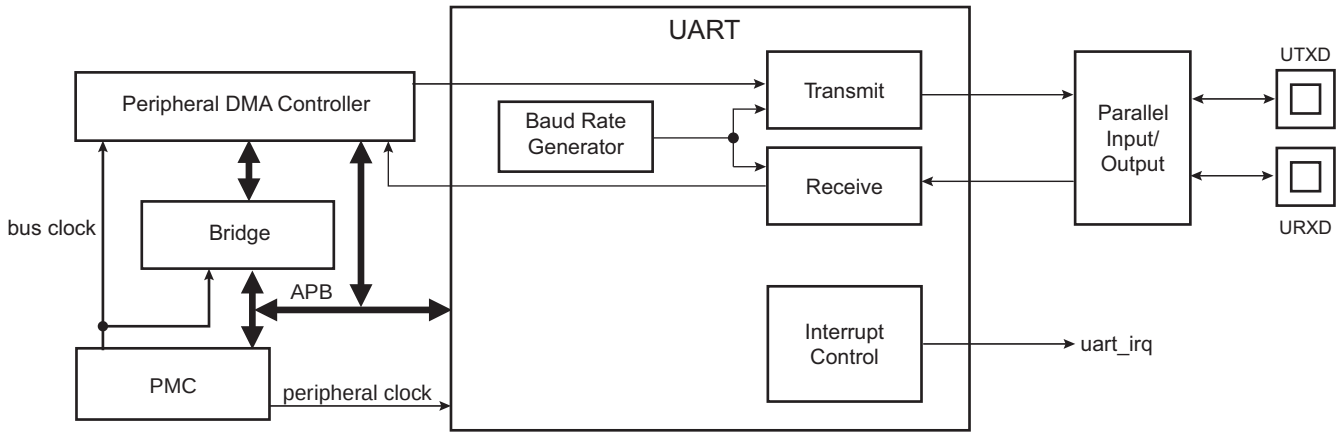


Table 36-1. UART Pin Description

Pin Name	Description	Type
URXD	UART Receive Data	Input
UTXD	UART Transmit Data	Output

36.4 Product Dependencies

36.4.1 I/O Lines

The UART pins are multiplexed with PIO lines. The user must first configure the corresponding PIO Controller to enable I/O line operations of the UART.

Table 36-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
UART0	URXD0	PA9	A
UART0	UTXD0	PA10	A
UART1	URXD1	PA5	C
UART1	UTXD1	PA6	C

36.4.2 Power Management

The UART clock can be controlled through the Power Management Controller (PMC). In this case, the user must first configure the PMC to enable the UART clock. Usually, the peripheral identifier used for this purpose is 1.

36.4.3 Interrupt Sources

The UART interrupt line is connected to one of the interrupt sources of the Interrupt Controller. Interrupt handling requires programming of the Interrupt Controller before configuring the UART.

Table 36-3. Peripheral IDs

Instance	ID
UART0	7
UART1	45

36.5 Functional Description

The UART operates in Asynchronous mode only and supports only 8-bit character handling (with parity). It has no clock pin.

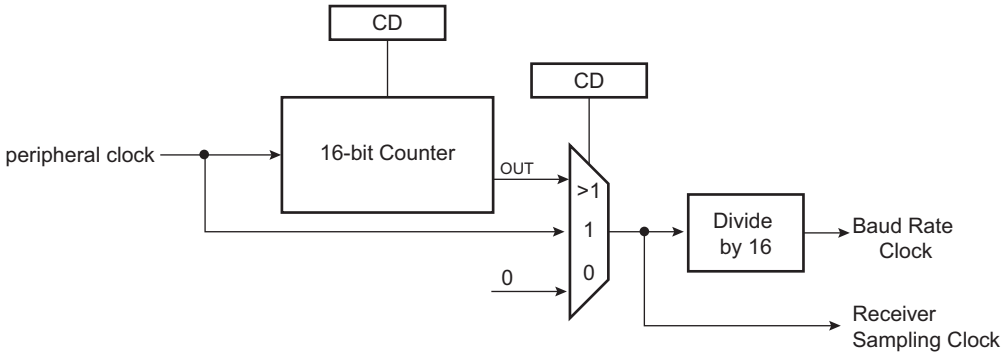
The UART is made up of a receiver and a transmitter that operate independently, and a common baud rate generator. Receiver timeout and transmitter time guard are not implemented. However, all the implemented features are compatible with those of a standard USART.

36.5.1 Baud Rate Generator

The baud rate generator provides the bit period clock named baud rate clock to both the receiver and the transmitter.

The baud rate clock is the peripheral clock divided by 16 times the clock divisor (CD) value written in the Baud Rate Generator register (UART_BRGR). If UART_BRGR is set to 0, the baud rate clock is disabled and the UART remains inactive. The maximum allowable baud rate is peripheral clock divided by 16. The minimum allowable baud rate is peripheral clock divided by (16 x 65536).

Figure 36-2. Baud Rate Generator



36.5.2 Receiver

36.5.2.1 Receiver Reset, Enable and Disable

After device reset, the UART receiver is disabled and must be enabled before being used. The receiver can be enabled by writing the Control Register (UART_CR) with the bit RXEN at 1. At this command, the receiver starts looking for a start bit.

The programmer can disable the receiver by writing UART_CR with the bit RXDIS at 1. If the receiver is waiting for a start bit, it is immediately stopped. However, if the receiver has already detected a start bit and is receiving the data, it waits for the stop bit before actually stopping its operation.

The receiver can be put in reset state by writing UART_CR with the bit RSTRX at 1. In this case, the receiver immediately stops its current operations and is disabled, whatever its current state. If RSTRX is applied when data is being processed, this data is lost.

36.5.2.2 Start Detection and Data Sampling

The UART only supports asynchronous operations, and this affects only its receiver. The UART receiver detects the start of a received character by sampling the URXD signal until it detects a valid start bit. A low level (space) on URXD is interpreted as a valid start bit if it is detected for more than seven 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 detecting the falling edge of the start bit.

Each subsequent bit is sampled 16 cycles (1-bit period) after the previous one.

Figure 36-3. Start Bit Detection

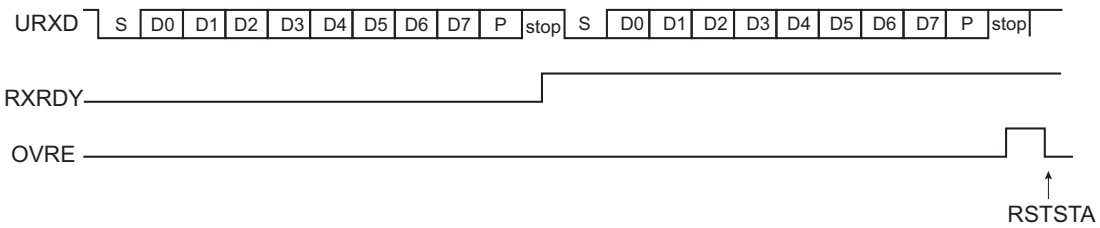
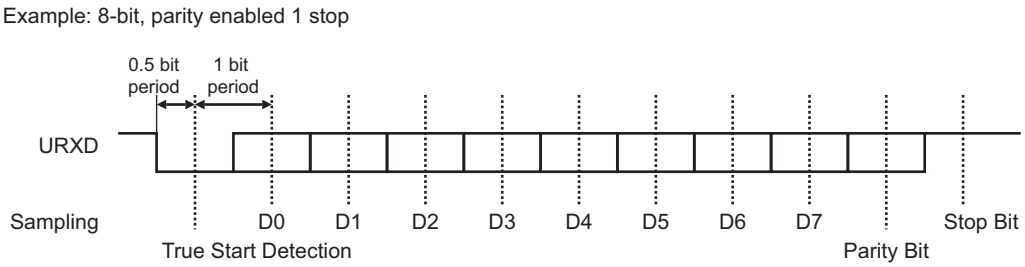


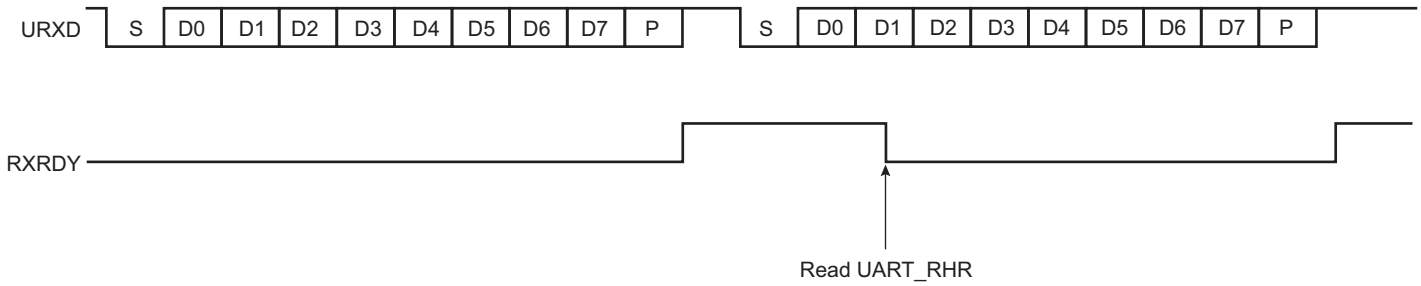
Figure 36-4. Character Reception



36.5.2.3 Receiver Ready

When a complete character is received, it is transferred to the Receive Holding Register (UART_RHR) and the RXRDY status bit in the Status Register (UART_SR) is set. The bit RXRDY is automatically cleared when UART_RHR is read.

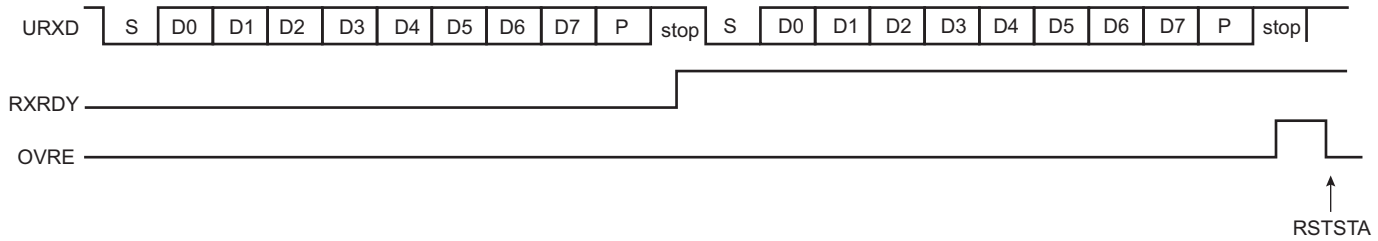
Figure 36-5. Receiver Ready



36.5.2.4 Receiver Overrun

The OVRE status bit in UART_SR is set if UART_RHR has not been read by the software (or the PDC) since the last transfer, the RXRDY bit is still set and a new character is received. OVRE is cleared when the software writes a 1 to the bit RSTSTA (Reset Status) in UART_CR.

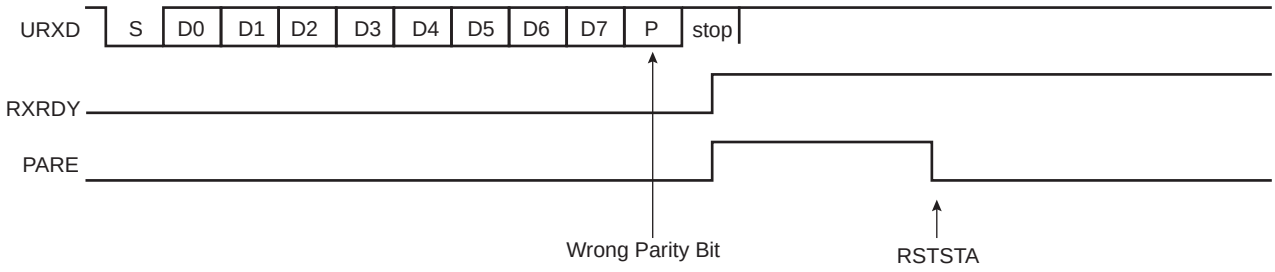
Figure 36-6. Receiver Overrun



36.5.2.5 Parity Error

Each time a character is received, the receiver calculates the parity of the received data bits, in accordance with the field PAR in the Mode Register (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 RXRDY is set. The parity bit is cleared when UART_CR is written with the bit RSTSTA (Reset Status) at 1. If a new character is received before the reset status command is written, the PARE bit remains at 1.

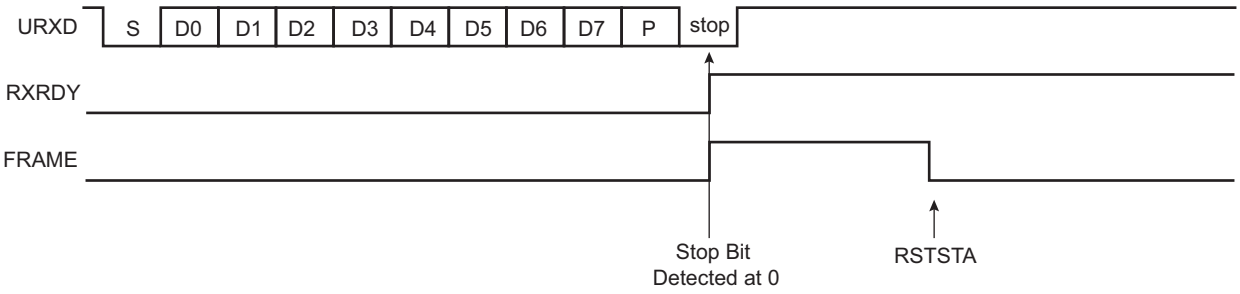
Figure 36-7. Parity Error



36.5.2.6 Receiver Framing Error

When a start bit is detected, it generates a character reception when all the data bits have been sampled. The stop bit is also sampled and when it is detected at 0, the FRAME (Framing Error) bit in UART_SR is set at the same time the RXRDY bit is set. The FRAME bit remains high until the Control Register (UART_CR) is written with the bit RSTSTA at 1.

Figure 36-8. Receiver Framing Error



36.5.3 Transmitter

36.5.3.1 Transmitter Reset, Enable and Disable

After device reset, the UART transmitter is disabled and must be enabled before being used. The transmitter is enabled by writing 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.

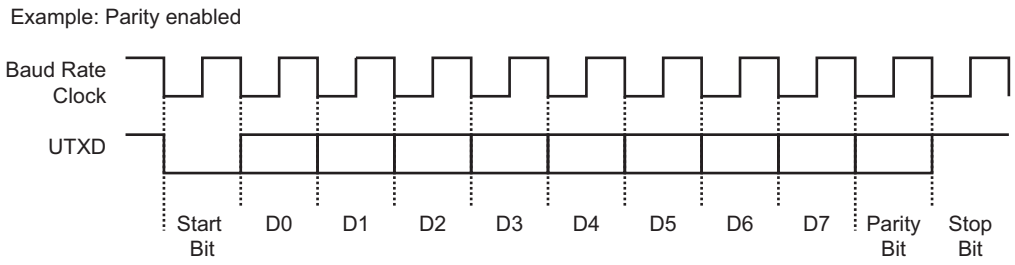
The programmer can disable the transmitter by writing UART_CR with the bit TXDIS at 1. If the transmitter is not operating, it is immediately stopped. However, if a character is being processed into the internal shift register and/or a character has been written in the UART_THR, the characters are completed before the transmitter is actually stopped.

The programmer can also put the transmitter in its reset state by writing the UART_CR with the bit RSTTX at 1. This immediately stops the transmitter, whether or not it is processing characters.

36.5.3.2 Transmit Format

The UART transmitter drives the pin UTXD at the baud rate clock speed. The line is driven depending on the format defined in UART_MR and the data stored in the internal 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 UART_MR defines whether or not a parity bit is shifted out. When a parity bit is enabled, it can be selected between an odd parity, an even parity, or a fixed space or mark bit.

Figure 36-9. Character Transmission

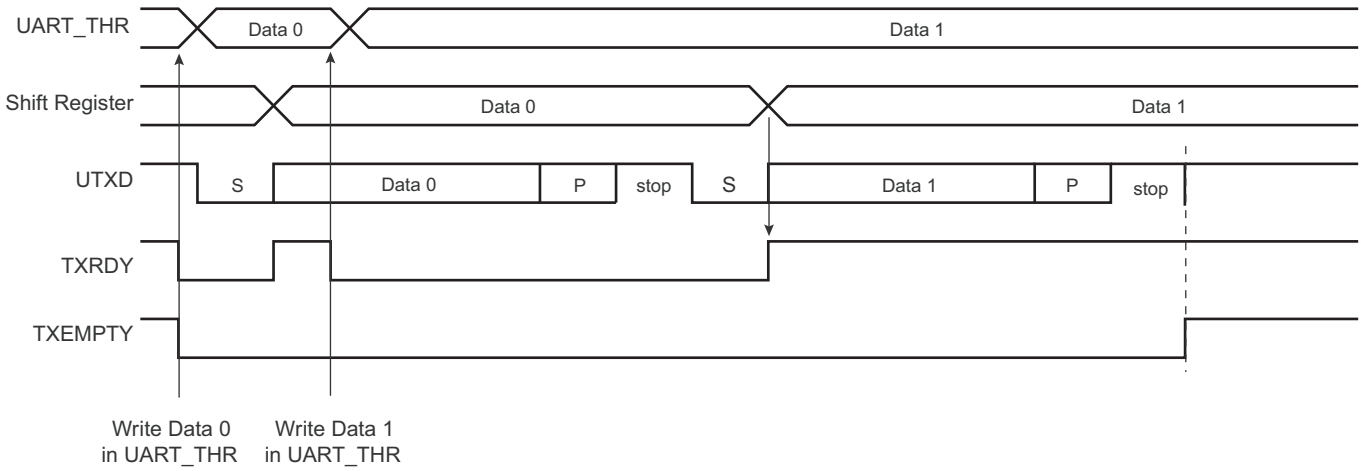


36.5.3.3 Transmitter Control

When the transmitter is enabled, the bit TXRDY (Transmitter Ready) is set in UART_SR. The transmission starts when the programmer writes in the UART_THR, and after the written character is transferred from UART_THR to the internal 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 internal shift register and TXRDY rises again, showing that the holding register is empty.

When both the internal shift register and UART_THR are empty, i.e., all the characters written in UART_THR have been processed, the TXEMPTY bit rises after the last stop bit has been completed.

Figure 36-10. Transmitter Control



36.5.4 Peripheral DMA Controller (PDC)

Both the receiver and the transmitter of the UART are connected to a PDC.

The PDC channels are programmed via registers that are mapped within the UART user interface from the offset 0x100. The status bits are reported in UART_SR and generate an interrupt.

The RXRDY bit triggers the PDC channel data transfer of the receiver. This results in a read of the data in UART_RHR. The TXRDY bit triggers the PDC channel data transfer of the transmitter. This results in a write of data in UART_THR.

36.5.5 Test Modes

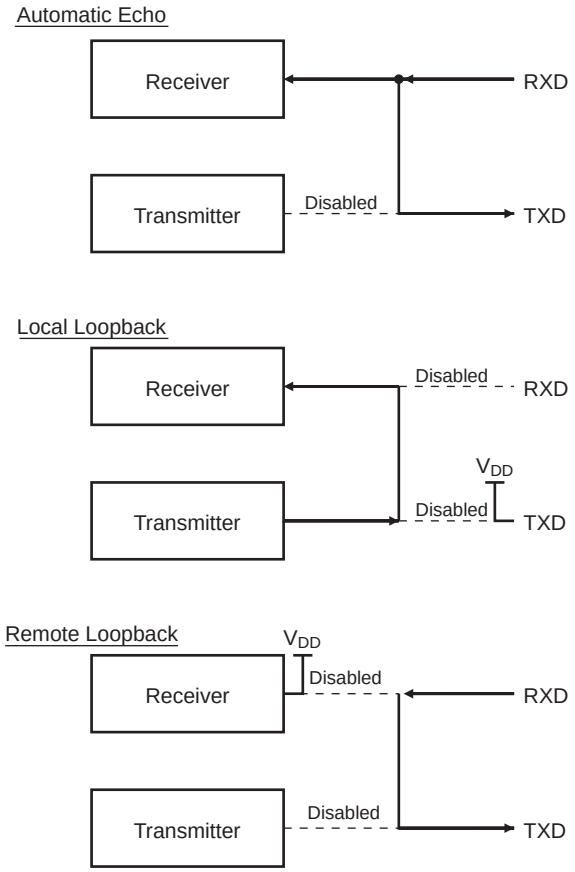
The UART supports three test modes. These modes of operation are programmed by using the CHMODE field in UART_MR.

The Automatic Echo mode allows a bit-by-bit retransmission. When a bit is received on the URXD line, it is sent to the UTXD line. The transmitter operates normally, but has no effect on the UTXD line.

The Local Loopback mode allows the transmitted characters to be received. UTXD and URXD pins are not used and the output of the transmitter is internally connected to the input of the receiver. The URXD pin level has no effect and the UTXD line is held high, as in idle state.

The Remote Loopback mode directly connects the URXD pin to the UTXD line. The transmitter and the receiver are disabled and have no effect. This mode allows a bit-by-bit retransmission.

Figure 36-11. Test Modes



36.6 Universal Asynchronous Receiver Transmitter (UART) User Interface

Table 36-4. 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	Reserved	–	–	–
0x0028–0x003C	Reserved	–	–	–
0x0040–0x00E8	Reserved	–	–	–
0x00EC–0x00FC	Reserved	–	–	–
0x0100–0x0128	Reserved for PDC registers	–	–	–

36.6.1 UART Control Register

Name: UART_CR

Address: 0x400E0600 (0), 0x40060600 (1)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	RSTSTA
7	6	5	4	3	2	1	0
TXDIS	TXEN	RXDIS	RXEN	RSTTX	RSTRX	–	–

- **RSTRX: Reset Receiver**

0: No effect.

1: The receiver logic is reset and disabled. If a character is being received, the reception is aborted.

- **RSTTX: Reset Transmitter**

0: No effect.

1: The transmitter logic is reset and disabled. If a character is being transmitted, the transmission is aborted.

- **RXEN: Receiver Enable**

0: No effect.

1: The receiver is enabled if RXDIS is 0.

- **RXDIS: Receiver Disable**

0: No effect.

1: The receiver is disabled. If a character is being processed and RSTRX is not set, the character is completed before the receiver is stopped.

- **TXEN: Transmitter Enable**

0: No effect.

1: The transmitter is enabled if TXDIS is 0.

- **TXDIS: Transmitter Disable**

0: No effect.

1: The transmitter is disabled. If a character is being processed and a character has been written in the UART_THR and RSTTX is not set, both characters are completed before the transmitter is stopped.

- **RSTSTA: Reset Status**

0: No effect.

1: Resets the status bits PARE, FRAME and OVRE in the UART_SR.

36.6.2 UART Mode Register

Name: UART_MR

Address: 0x400E0604 (0), 0x40060604 (1)

Access: Read/Write

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

- **PAR: Parity Type**

Value	Name	Description
0	EVEN	Even Parity
1	ODD	Odd Parity
2	SPACE	Space: parity forced to 0
3	MARK	Mark: parity forced to 1
4	NO	No parity

- **CHMODE: Channel Mode**

Value	Name	Description
0	NORMAL	Normal mode
1	AUTOMATIC	Automatic echo
2	LOCAL_LOOPBACK	Local loopback
3	REMOTE_LOOPBACK	Remote loopback

36.6.3 UART Interrupt Enable Register

Name: UART_IER

Address: 0x400E0608 (0), 0x40060608 (1)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	–	TXEMPTY	–
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **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**

36.6.4 UART Interrupt Disable Register

Name: UART_IDR

Address: 0x400E060C (0), 0x4006060C (1)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	–	TXEMPTY	–
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Disables the corresponding interrupt.

- **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**

36.6.5 UART Interrupt Mask Register

Name: UART_IMR

Address: 0x400E0610 (0), 0x40060610 (1)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	–	TXEMPTY	–
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is disabled.

1: The corresponding interrupt is enabled.

- **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**

36.6.6 UART Status Register

Name: UART_SR

Address: 0x400E0614 (0), 0x40060614 (1)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	–	TXEMPTY	–
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

- **RXRDY: Receiver Ready**

0: No character has been received since the last read of the UART_RHR, or the receiver is disabled.

1: At least one complete character has been received, transferred to UART_RHR and not yet read.

- **TXRDY: Transmitter Ready**

0: A character has been written to UART_THR and not yet transferred to the internal shift register, or the transmitter is disabled.

1: There is no character written to UART_THR not yet transferred to the internal shift register.

- **ENDRX: End of Receiver Transfer**

0: The end of transfer signal from the receiver PDC channel is inactive.

1: The end of transfer signal from the receiver PDC channel is active.

- **ENDTX: End of Transmitter Transfer**

0: The end of transfer signal from the transmitter PDC channel is inactive.

1: The end of transfer signal from the transmitter 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 framing error has occurred since the last RSTSTA.

1: At least one framing error has occurred since the last RSTSTA.

- **PARE: Parity Error**

0: No parity error has occurred since the last RSTSTA.

1: At least one parity error has occurred since the last RSTSTA.

- **TXEMPTY: Transmitter Empty**

0: There are characters in UART_THR, or characters being processed by the transmitter, or the transmitter is disabled.

1: There are no characters in UART_THR and there are no characters being processed by the transmitter.

- **TXBUFE: Transmission Buffer Empty**

0: The buffer empty signal from the transmitter PDC channel is inactive.

1: The buffer empty signal from the transmitter PDC channel is active.

- **RXBUFF: Receive Buffer Full**

0: The buffer full signal from the receiver PDC channel is inactive.

1: The buffer full signal from the receiver PDC channel is active.

36.6.7 UART Receiver Holding Register

Name: UART_RHR

Address: 0x400E0618 (0), 0x40060618 (1)

Access: Read-only

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

- **RXCHR: Received Character**

Last received character if RXRDY is set.

36.6.8 UART Transmit Holding Register

Name: UART_THR

Address: 0x400E061C (0), 0x4006061C (1)

Access: Write-only

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

- **TXCHR: Character to be Transmitted**

Next character to be transmitted after the current character if TXRDY is not set.

36.6.9 UART Baud Rate Generator Register

Name: UART_BRGR

Address: 0x400E0620 (0), 0x40060620 (1)

Access: Read/Write

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

- **CD: Clock Divisor**

0: Baud rate clock is disabled

1 to 65,535:

$$CD = \frac{f_{peripheral\ clock}}{16 \times Baud\ Rate}$$

37. Universal Synchronous Asynchronous Receiver Transmitter (USART)

37.1 Description

The Universal Synchronous Asynchronous Receiver Transceiver (USART) provides one full duplex universal synchronous asynchronous serial link. Data frame format is widely programmable (data length, parity, number of stop bits) to support a maximum of standards. The receiver implements parity error, framing error and overrun error detection. The receiver time-out enables handling variable-length frames and the transmitter timeguard facilitates communications with slow remote devices. Multidrop communications are also supported through address bit handling in reception and transmission.

The USART features three test modes: Remote Loopback, Local Loopback and Automatic Echo.

The USART supports specific operating modes providing interfaces on RS485, and SPI buses, with ISO7816 T = 0 or T = 1 smart card slots, infrared transceivers and connection to modem ports. The hardware handshaking feature enables an out-of-band flow control by automatic management of the pins RTS and CTS.

The USART supports the connection to the DMA Controller and the Peripheral DMA Controller, which enables data transfers to the transmitter and from the receiver. The PDC and DMAC provide chained buffer management without any intervention of the processor.

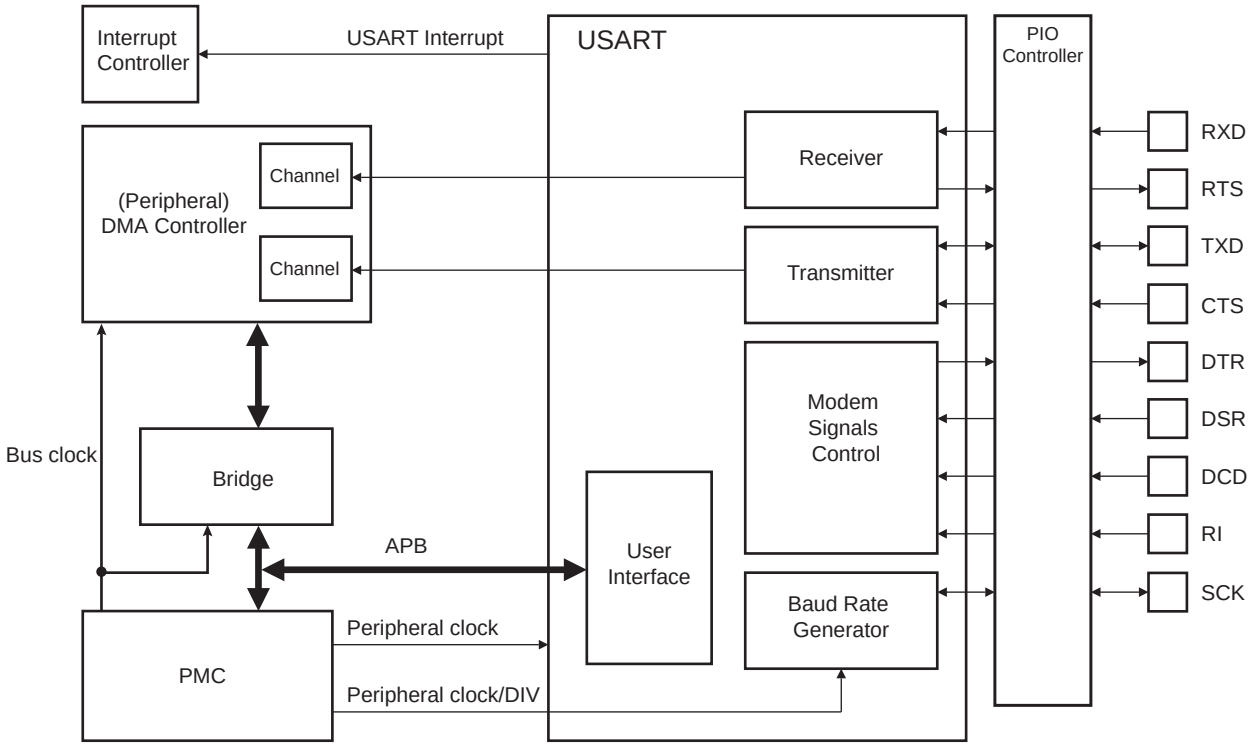
37.2 Embedded Characteristics

- Programmable Baud Rate Generator
- 5- to 9-bit Full-duplex Synchronous or Asynchronous Serial Communications
  - 1, 1.5 or 2 Stop Bits in Asynchronous Mode or 1 or 2 Stop Bits in Synchronous Mode
  - Parity Generation and Error Detection
  - Framing Error Detection, Overrun Error Detection
  - Digital Filter on Receive Line
  - MSB- or LSB-first
  - Optional Break Generation and Detection
  - By 8 or by 16 Oversampling Receiver Frequency
  - Optional Hardware Handshaking RTS-CTS
  - Optional Modem Signal Management DTR-DSR-DCD-RI
  - Receiver Time-out and Transmitter Timeguard
  - Optional Multidrop Mode with Address Generation and Detection
- RS485 with Driver Control Signal
- ISO7816, T = 0 or T = 1 Protocols for Interfacing with Smart Cards
  - NACK Handling, Error Counter with Repetition and Iteration Limit
- IrDA Modulation and Demodulation
  - Communication at up to 115.2 kbit/s
- SPI Mode
  - Master or Slave
  - Serial Clock Programmable Phase and Polarity
  - SPI Serial Clock (SCK) Frequency up to $f_{peripheral\ clock}/6$
- Test Modes
  - Remote Loopback, Local Loopback, Automatic Echo
- Supports Connection of:
  - Two DMA Controller Channels (DMAC) and Two Peripheral DMA Controller Channels (PDC)

- Offers Buffer Transfer without Processor Intervention
- Register Write Protection

37.3 Block Diagram

Figure 37-1. USART Block Diagram



37.4 I/O Lines Description

Table 37-1. I/O Line Description

Name	Description	Type	Active Level
SCK	Serial Clock	I/O	—
TXD	Transmit Serial Data or Master Out Slave In (MOSI) in SPI Master mode or Master In Slave Out (MISO) in SPI Slave mode	I/O	—
RXD	Receive Serial Data or Master In Slave Out (MISO) in SPI Master mode or Master Out Slave In (MOSI) in SPI Slave mode	Input	—
RI	Ring Indicator	Input	Low
DSR	Data Set Ready	Input	Low
DCD	Data Carrier Detect	Input	Low
DTR	Data Terminal Ready	Output	Low
CTS	Clear to Send or Slave Select (NSS) in SPI Slave mode	Input	Low
RTS	Request to Send or Slave Select (NSS) in SPI Master mode	Output	Low

37.5 Product Dependencies

37.5.1 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.

All the pins of the modems may or may not be implemented on the USART. Only USART1 is fully equipped with all the modem signals. On USARTs not equipped with the corresponding pin, the associated control bits and statuses have no effect on the behavior of the USART.

Table 37-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
USART0	CTS0	PB2	C
USART0	RTS0	PB3	C
USART0	RXD0	PB0	C
USART0	SCK0	PB13	C
USART0	TXD0	PB1	C
USART1	CTS1	PA25	A
USART1	DCD1	PA26	A
USART1	DSR1	PA28	A
USART1	DTR1	PA27	A
USART1	RI1	PA29	A
USART1	RTS1	PA24	A
USART1	RXD1	PA21	A
USART1	SCK1	PA23	A
USART1	TXD1	PA22	A

37.5.2 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.

37.5.3 Interrupt Sources

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.

Table 37-3. Peripheral IDs

Instance	ID
USART0	14
USART1	15

37.6 Functional Description

37.6.1 Baud Rate Generator

The baud rate generator provides the bit period clock, also named the baud rate clock, to both the receiver and the transmitter.

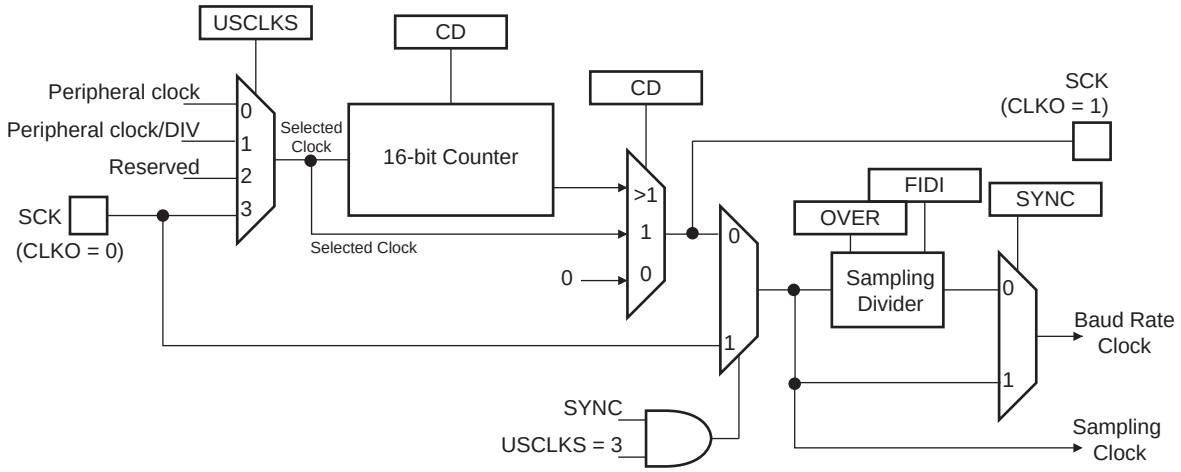
The baud rate generator clock source is selected by configuring the USCLKS field in the USART Mode Register (US_MR) to one of the following:

- The peripheral clock
- A division of the peripheral clock, where the divider is 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 a 0 is written to CD, the baud rate generator does not generate any clock. If a 1 is written to CD, 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 peripheral clock period. The frequency of the signal provided on SCK must be at least 3 times lower than the frequency provided on the peripheral clock in USART mode (field USART_MODE differs from 0xE or 0xF), or 6 times lower in SPI mode (field USART_MODE equals 0xE or 0xF).

Figure 37-2. Baud Rate Generator



37.6.1.1 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 US_BRGR. The resulting clock is provided to the receiver as a sampling clock and then divided by 16 or 8, depending on how the OVER bit in the US_MR is programmed.

If OVER is set, the receiver sampling is eight times higher than the baud rate clock. If OVER is cleared, the sampling is performed at 16 times the baud rate clock.

The baud rate is calculated as per the following formula:

$$\text{Baud Rate} = \frac{\text{Selected Clock}}{(8(2-OVER)CD)}$$

This gives a maximum baud rate of peripheral clock divided by 8, assuming that the peripheral clock is the highest possible clock and that the OVER bit is set.

Baud Rate Calculation Example

Table 37-4 shows calculations of CD to obtain a baud rate at 38,400 bit/s for different source clock frequencies. This table also shows the actual resulting baud rate and the error.

Table 37-4. Baud Rate Example (OVER = 0)

Source Clock (MHz)	Expected Baud Rate (bit/s)	Calculation Result	CD	Actual Baud Rate (bit/s)	Error
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%

In this example, the baud rate is calculated with the following formula:

$$\text{Baud Rate} = \text{Selected Clock} / CD \times 16$$

The baud rate error is calculated with the following formula. It is not recommended to work with an error higher than 5%.

$$Error = 1 - \left(\frac{\text{Expected Baud Rate}}{\text{Actual Baud Rate}} \right)$$

37.6.1.2 Fractional Baud Rate in Asynchronous Mode

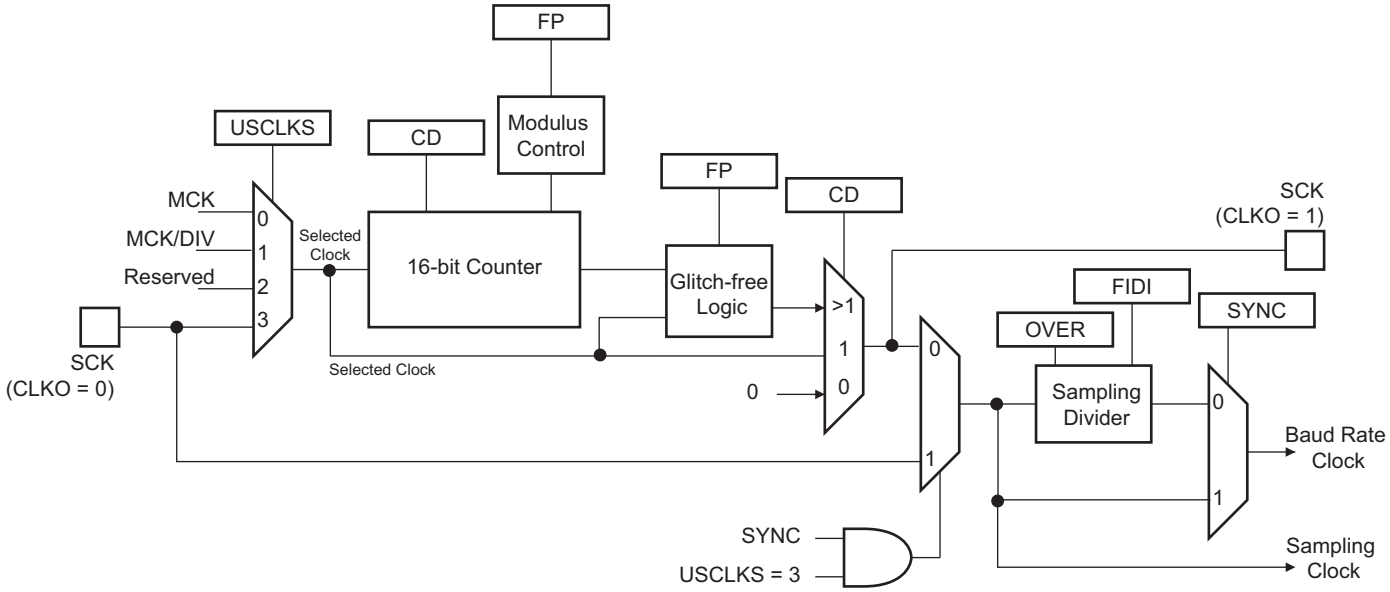
The baud rate generator is subject to the following limitation: the output frequency changes only by integer multiples of the reference frequency. An approach to this problem is to integrate a fractional N clock generator that has a high resolution. The generator architecture is modified to obtain baud rate changes by a fraction of the reference source clock. This fractional part is programmed with the FP field in the 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:

$$\text{Baud Rate} = \frac{\text{Selected Clock}}{\left(8(2-OVER)\left(CD+\frac{FP}{8}\right)\right)}$$

The modified architecture is presented in the following Figure 37-3.

Figure 37-3. Fractional Baud Rate Generator



Warning: When the value of field FP is greater than 0, the SCK (oversampling clock) generates non-constant duty cycles. The SCK high duration is increased by “selected clock” period from time to time. The duty cycle depends on the value of the CD field.

37.6.1.3 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 the US_BRGR.

$$\text{Baud Rate} = \frac{\text{Selected Clock}}{CD}$$

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 Master mode, Synchronous mode (USCLKS = 0 or 1, CLKO set to 1), the receive part limits the SCK maximum frequency to Selected Clock/3 in USART mode, or Selected Clock/6 in SPI mode.

When either the external clock SCK or the internal clock divided (peripheral clock/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. When the peripheral clock is selected, the baud rate generator ensures a 50:50 duty cycle on the SCK pin, even if the value programmed in CD is odd.

37.6.1.4 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 37-5.

Table 37-5. 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 37-6.

Table 37-6. 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 37-7 shows the resulting Fi/Di ratio, which is the ratio between the ISO7816 clock and the baud rate clock.

Table 37-7. Possible Values for the Fi/Di Ratio

Fi/Di	372	558	744	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

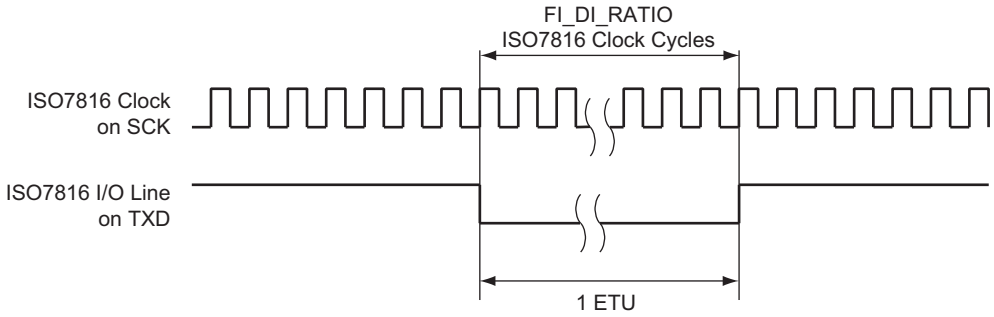
If the USART is configured in ISO7816 mode, the clock selected by the USCLKS field in US_MR is first divided by the value programmed in the field CD in the US_BRGR. The resulting clock can be provided to the SCK pin to feed the smart card clock inputs. This means that the CLKO bit can be set in US_MR.

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 37-4 shows the relation between the Elementary Time Unit, corresponding to a bit time, and the ISO 7816 clock.

Figure 37-4. Elementary Time Unit (ETU)



37.6.2 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 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 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 the US_CR. If the receiver is disabled during a character reception, the USART waits until the end of reception of the current character, then the reception is stopped. If the transmitter is disabled while it is operating, the USART waits the end of transmission of both the current character and character being stored in the Transmit Holding register (US_THR). If a timeguard is programmed, it is handled normally.

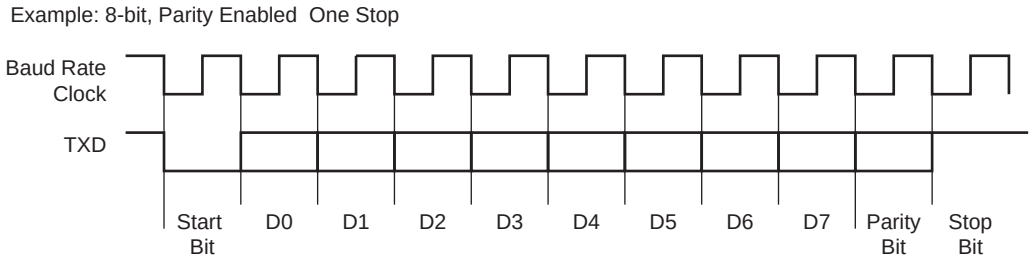
37.6.3 Synchronous and Asynchronous Modes

37.6.3.1 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 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 the 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 the US_MR. The 1.5 stop bit is supported in Asynchronous mode only.

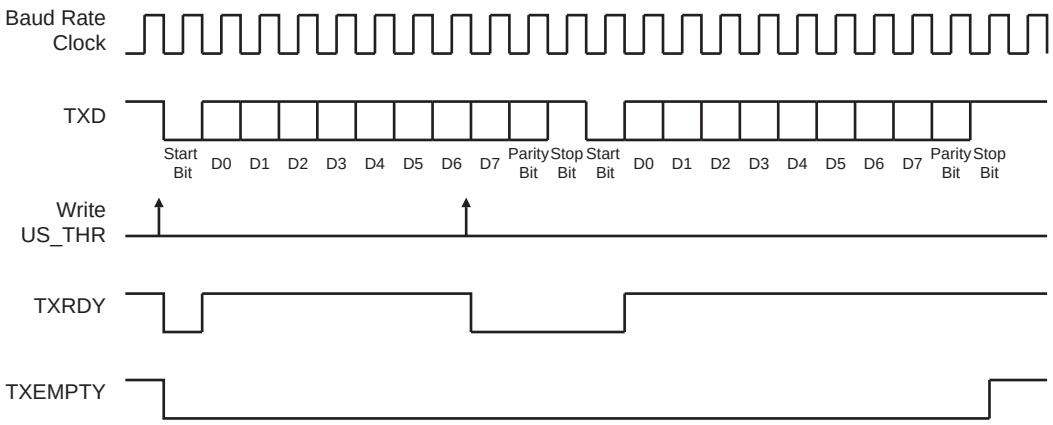
Figure 37-5. Character Transmit



The characters are sent by writing in the Transmit Holding register (US_THR). The transmitter reports two status bits in the Channel Status register (US_CSR): TXRDY (Transmitter Ready), which indicates that US_THR is empty and TXEMPTY, which indicates that all the characters written in US_THR have been processed. When the current character processing is completed, the last character written in US_THR is transferred into the Shift register of the transmitter and US_THR becomes empty, thus TXRDY rises.

Both TXRDY and TXEMPTY bits are low when the transmitter is disabled. Writing a character in US_THR while TXRDY is low has no effect and the written character is lost.

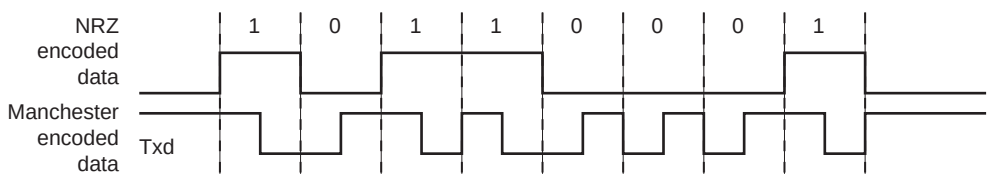
Figure 37-6. Transmitter Status



37.6.3.2 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 bit in the US_MR to 1. Depending on polarity configuration, a logic level (zero or one), is transmitted as a coded signal one-to-zero or zero-to-one. Thus, a transition always occurs at the midpoint of each bit time. It consumes more bandwidth than the original NRZ signal (2x) but the receiver has more error control since the expected input must show a change at the center of a bit cell. An example of Manchester encoded sequence is: the byte 0xB1 or 10110001 encodes to 10 01 10 10 01 01 01 10, assuming the default polarity of the encoder. Figure 37-7 illustrates this coding scheme.

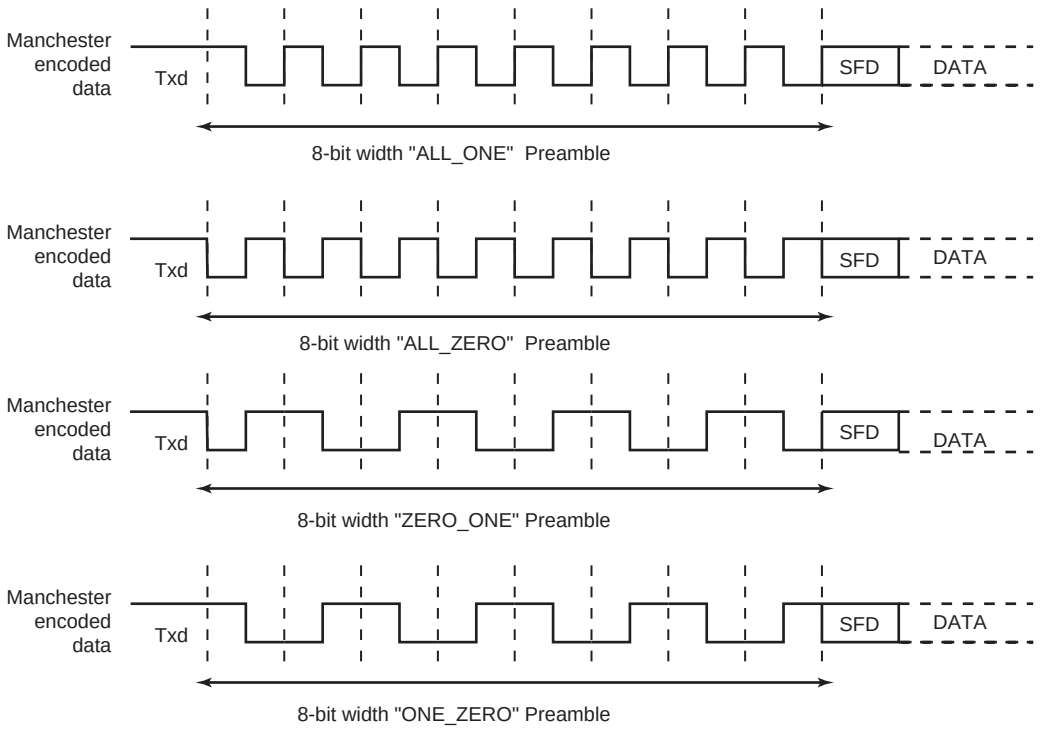
Figure 37-7. NRZ to Manchester Encoding



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

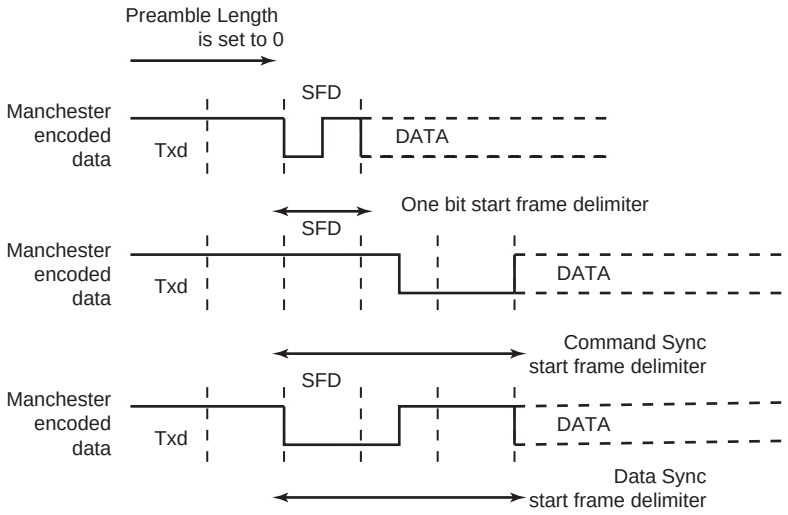
predefined pattern with a programmable length from 1 to 15 bit times. If the preamble length is set to 0, the preamble waveform is not generated prior to any character. The preamble pattern is chosen among the following sequences: ALL_ONE, ALL_ZERO, ONE_ZERO or ZERO_ONE, writing the field TX_PP in the US_MAN register, the field TX_PL is used to configure the preamble length. Figure 37-8 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 1, a logic one is encoded with a one-to-zero transition and a logic zero is encoded with a zero-to-one transition.

Figure 37-8. Preamble Patterns, Default Polarity Assumed



A start frame delimiter is to be configured using the ONEBIT bit in the US_MR. It consists of a user-defined pattern that indicates the beginning of a valid data. Figure 37-9 illustrates these patterns. If the start frame delimiter, also known as the start bit, is one bit, (ONEBIT = 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 three 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 bit in the US_MR 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 bit in US_MR must be set to 1. In this case, the MODSYNC bit in the US_MR is bypassed and the sync configuration is held in the TXSYNH in the US_THR. The USART character format is modified and includes sync information.

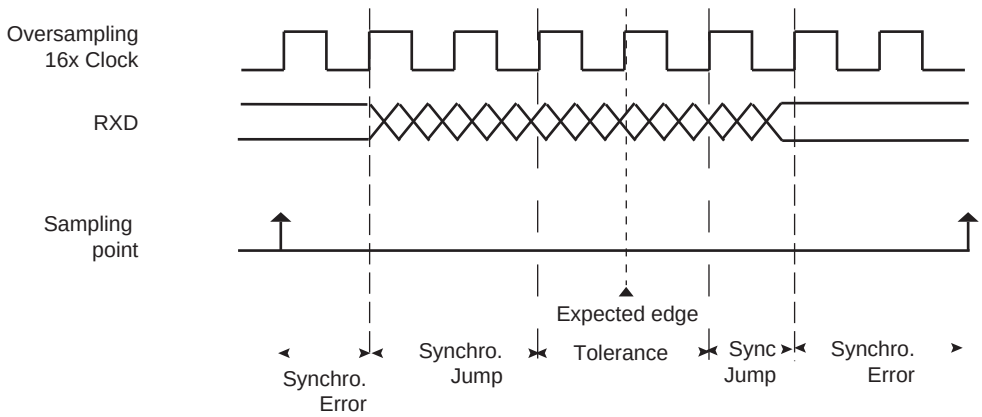
Figure 37-9. Start Frame Delimiter



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 37-10. Bit Resynchronization



37.6.3.3 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 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 = 0), a start is detected at the eighth sample to 0. Data bits, parity bit and stop bit are assumed to have a duration corresponding to 16 oversampling clock cycles. If the oversampling is 8 (OVER = 1), a start bit is detected at the fourth sample to 0. Data bits, parity bit and stop bit are assumed to have a duration corresponding to 8 oversampling clock cycles.

The number of data bits, first bit sent and Parity mode are selected by the same fields and bits as the transmitter, i.e., respectively CHRL, MODE9, MSBF and PAR. For the synchronization mechanism **only**, the number of stop bits has no effect on the receiver as it considers only one stop bit, regardless of the field NBSTOP, so that resynchronization between the receiver and the transmitter can occur. Moreover, as soon as the stop bit is sampled, the receiver starts looking for a new start bit so that resynchronization can also be accomplished when the transmitter is operating with one stop bit.

Figure 37-11 and Figure 37-12 illustrate start detection and character reception when USART operates in Asynchronous mode.

Figure 37-11. Asynchronous Start Detection

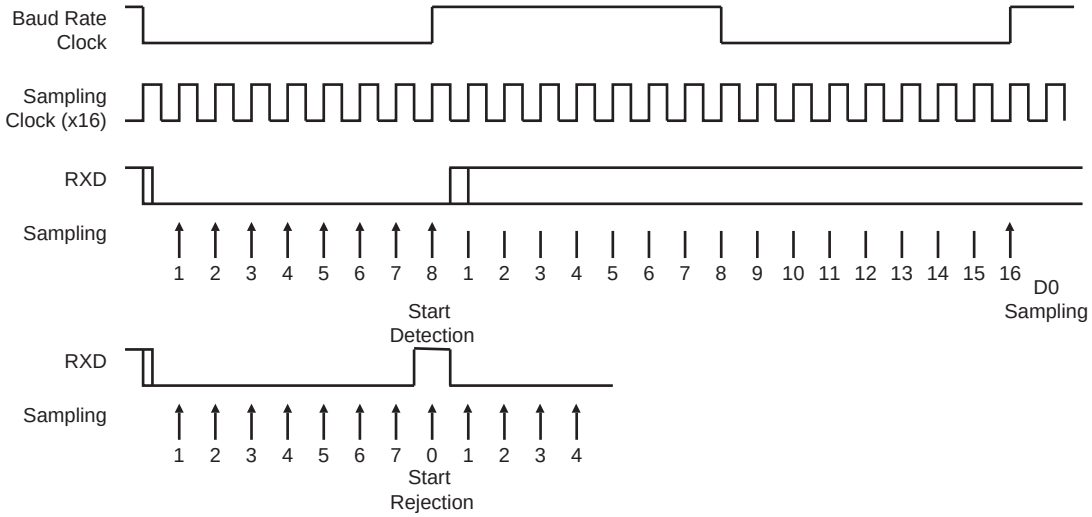
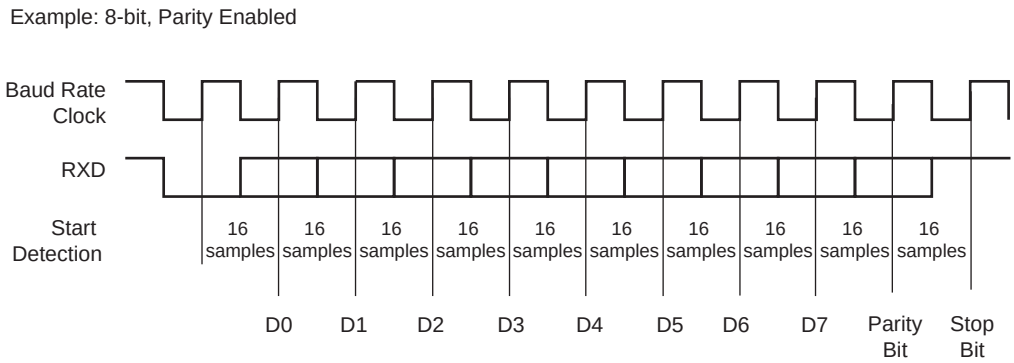


Figure 37-12. Asynchronous Character Reception



37.6.3.4 Manchester Decoder

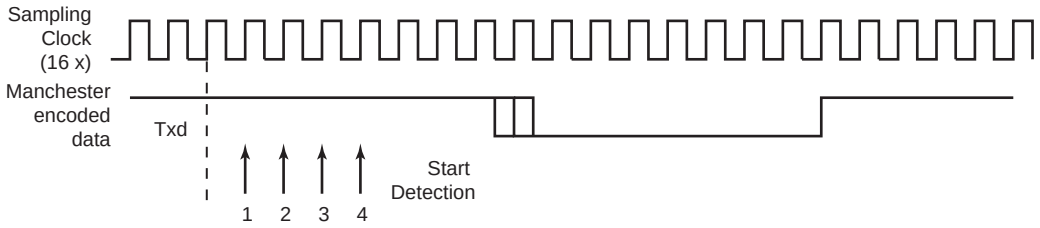
When the MAN bit in the US_MR is set to 1, the Manchester decoder is enabled. The decoder performs both preamble and start frame delimiter detection. One input line is dedicated to Manchester encoded input data.

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 bit in US_MAN register. Depending on the desired application the preamble pattern matching is to be defined via the RX_PP field in US_MAN. See Figure 37-8 for available preamble patterns.

Unlike preamble, the start frame delimiter is shared between Manchester Encoder and Decoder. So, if ONEBIT field is set to 1, only a zero encoded Manchester can be detected as a valid start frame delimiter. If ONEBIT is set

to 0, only a sync pattern is detected as a valid start frame delimiter. Decoder operates by detecting transition on incoming stream. If RXD is sampled during one quarter of a bit time to zero, a start bit is detected. See Figure 37-13. The sample pulse rejection mechanism applies.

Figure 37-13. Asynchronous Start Bit Detection



The receiver is activated and starts preamble and frame delimiter detection, sampling the data at one quarter and then three quarters. If a valid preamble pattern or start frame delimiter is detected, the receiver continues decoding with the same synchronization. If the stream does not match a valid pattern or a valid start frame delimiter, the receiver resynchronizes on the next valid edge.The minimum time threshold to estimate the bit value is three quarters of a bit time.

If a valid preamble (if used) followed with a valid start frame delimiter is detected, the incoming stream is decoded into NRZ data and passed to USART for processing. Figure 37-14 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, the MANERR flag in the US_CSR is raised. It is cleared by writing a 1 to the RSTSTA in the US_CR. See Figure 37-15 for an example of Manchester error detection during data phase.

Figure 37-14. Preamble Pattern Mismatch

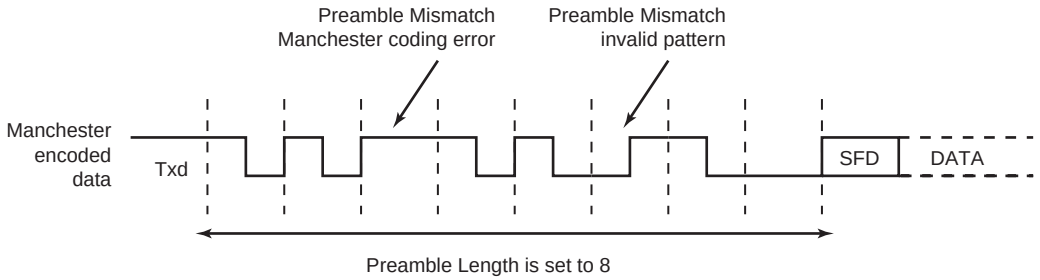
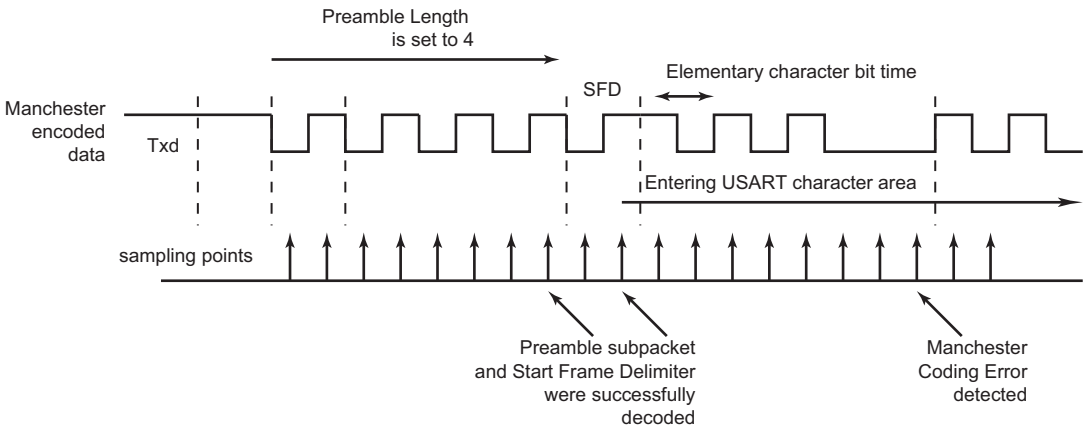


Figure 37-15. Manchester Error Flag



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 and the RXSYNH is updated. RXCHR is set to 1 when the received character is a command, and it is set to 0 if the received character is a data. This mechanism alleviates and simplifies the direct memory access as the character contains its own sync field in the same register.

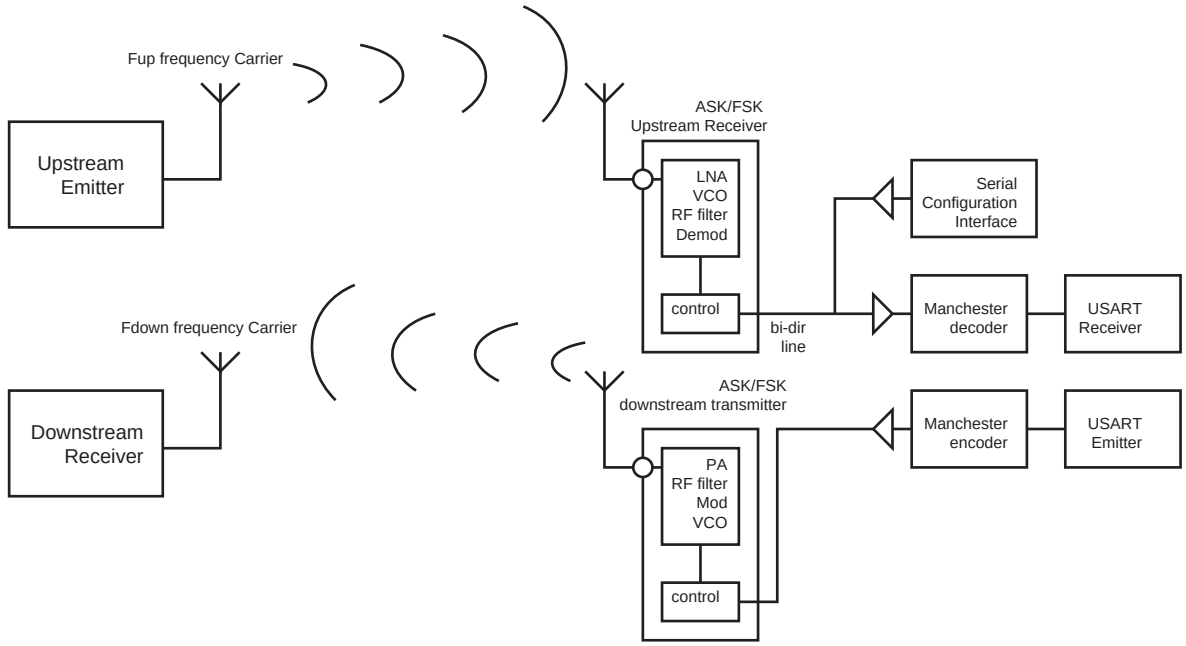
As the decoder is setup to be used in Unipolar mode, the first bit of the frame has to be a zero-to-one transition.

37.6.3.5 Radio Interface: Manchester Encoded USART Application

This section describes low data rate RF transmission systems and their integration with a Manchester encoded USART. These systems are based on transmitter and receiver ICs that support ASK and FSK modulation schemes.

The goal is to perform full duplex radio transmission of characters using two different frequency carriers. See the configuration in Figure 37-16.

Figure 37-16. Manchester Encoded Characters RF Transmission



The USART peripheral is configured as a Manchester encoder/decoder. Looking at the downstream communication channel, Manchester encoded characters are serially sent to the RF emitter. This may also include a user defined preamble and a start frame delimiter. Mostly, preamble is used in the RF receiver to distinguish between a valid data from a transmitter and signals due to noise. The Manchester stream is then modulated. See Figure 37-17 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 37-18.

From the receiver side, another carrier frequency is used. The RF receiver performs a bit check operation examining demodulated data stream. If a valid pattern is detected, the receiver switches to Receiving mode. The demodulated stream is sent to the Manchester decoder. Because of bit checking inside RF IC, the data transferred to the microcontroller is reduced by a user-defined number of bits. The Manchester preamble length is to be defined in accordance with the RF IC configuration.

Figure 37-17. ASK Modulator Output

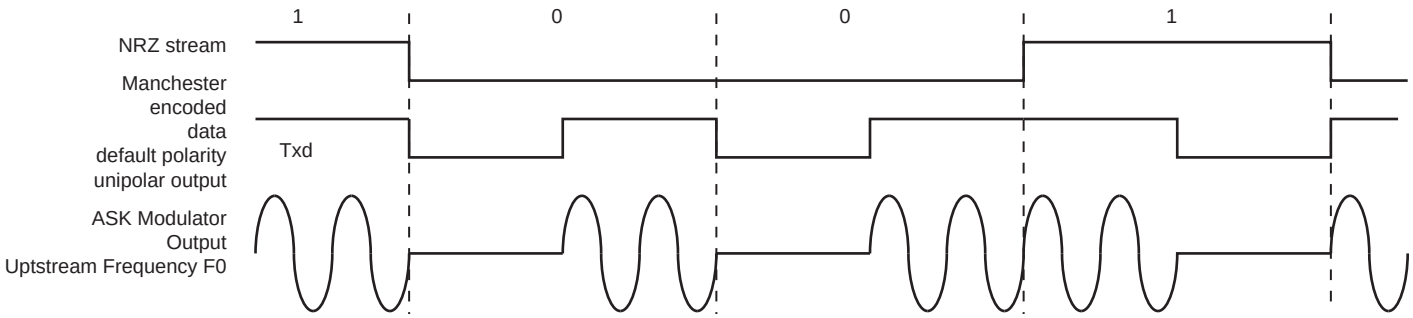
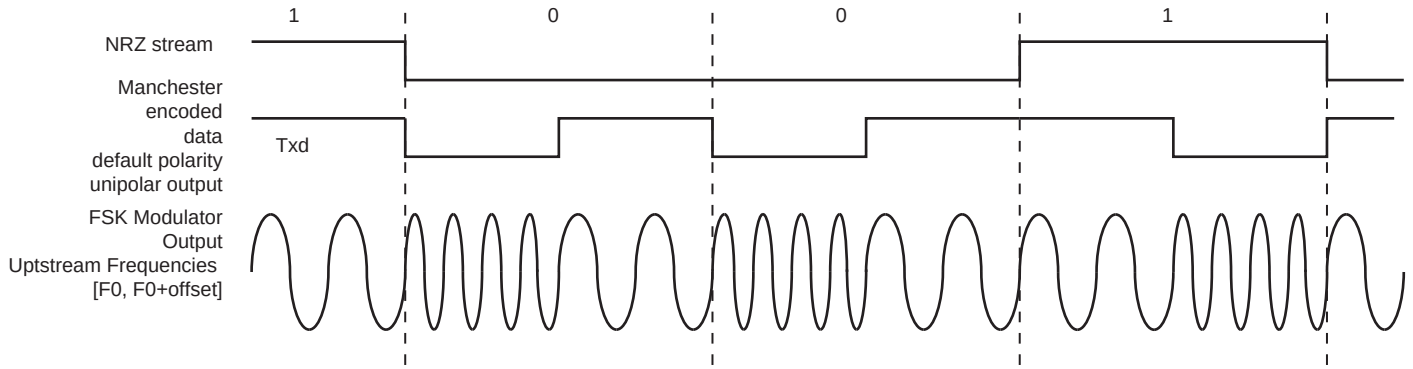


Figure 37-18. FSK Modulator Output



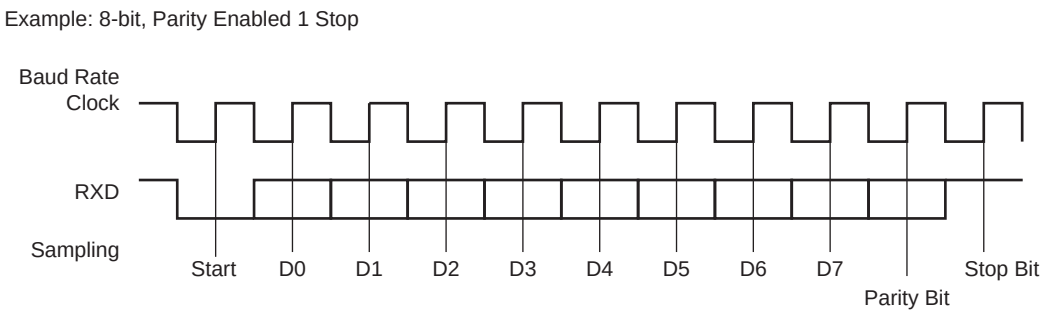
37.6.3.6 Synchronous Receiver

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 37-19 illustrates a character reception in Synchronous mode.

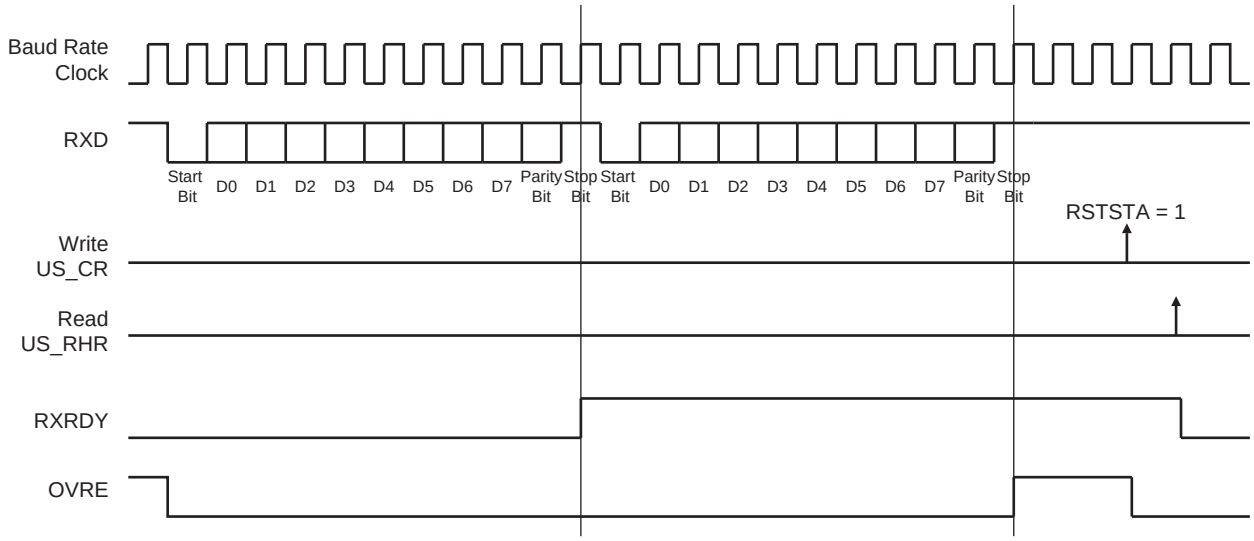
Figure 37-19. Synchronous Mode Character Reception



37.6.3.7 Receiver Operations

When a character reception is completed, it is transferred to the Receive Holding register (US_RHR) and the RXRDY bit in 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 a 1 to the RSTSTA (Reset Status) bit in the US_CR.

Figure 37-20. Receiver Status



37.6.3.8 Parity

The USART supports five Parity modes that are selected by writing to the PAR field in the US_MR. The PAR field also enables the Multidrop mode, see Section 37.6.3.9 ”Multidrop Mode”. Even and odd parity bit generation and error detection are supported.

If even parity is selected, the parity generator of the transmitter drives the parity bit to 0 if a number of 1s in the character data bit is even, and to 1 if the number of 1s is odd. Accordingly, the receiver parity checker counts the number of received 1s and reports a parity error if the sampled parity bit does not correspond. If odd parity is selected, the parity generator of the transmitter drives the parity bit to 1 if a number of 1s in the character data bit is even, and to 0 if the number of 1s is odd. Accordingly, the receiver parity checker counts the number of received 1s and reports a parity error if the sampled parity bit does not correspond. If the mark parity is used, the parity generator of the transmitter drives the parity bit to 1 for all characters. The receiver parity checker reports an error if the parity bit is sampled to 0. If the space parity is used, the parity generator of the transmitter drives the parity bit to 0 for all characters. The receiver parity checker reports an error if the parity bit is sampled to 1. If parity is disabled, the transmitter does not generate any parity bit and the receiver does not report any parity error.

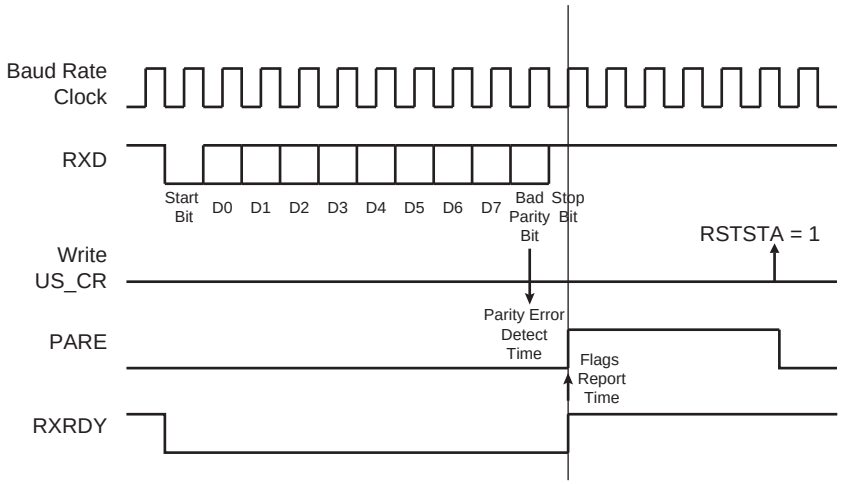
Table 37-8 shows an example of the parity bit for the character 0x41 (character ASCII “A”) depending on the configuration of the USART. Because there are two bits set to 1 in the character value, the parity bit is set to 1 when the parity is odd, or configured to 0 when the parity is even.

Table 37-8. Parity Bit Examples

Character	Hexadecimal	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 US_CSR. The PARE bit can be cleared by writing a 1 to the RSTSTA bit the US_CR. Figure 37-21 illustrates the parity bit status setting and clearing.

Figure 37-21. Parity Error



37.6.3.9 Multidrop Mode

If the value 0x6 or 0x07 is written to the PAR field in the US_MR, the USART runs in Multidrop mode. This mode differentiates the data characters and the address characters. Data is transmitted with the parity bit at 0 and addresses are transmitted with the parity bit at 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 a 1 is written to the SENTA bit in the US_CR.

To handle parity error, the PARE bit is cleared when a 1 is written to the RSTSTA bit in the US_CR.

The transmitter sends an address byte (parity bit set) when SENDA is written to in the US_CR. In this case, the next byte written to the US_THR is transmitted as an address. Any character written in the US_THR without having written the command SENDA is transmitted normally with the parity at 0.

37.6.3.10 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 written to zero no timeguard is generated. Otherwise, the transmitter holds a high level on TXD after each transmitted byte during the number of bit periods programmed in TG in addition to the number of stop bits.

As illustrated in Figure 37-22, the behavior of TXRDY and TXEMPTY status bits is modified by the programming of a timeguard. TXRDY rises only when the start bit of the next character is sent, and thus remains to 0 during the timeguard transmission if a character has been written in US_THR. TXEMPTY remains low until the timeguard transmission is completed as the timeguard is part of the current character being transmitted.

Figure 37-22. Timeguard Operations

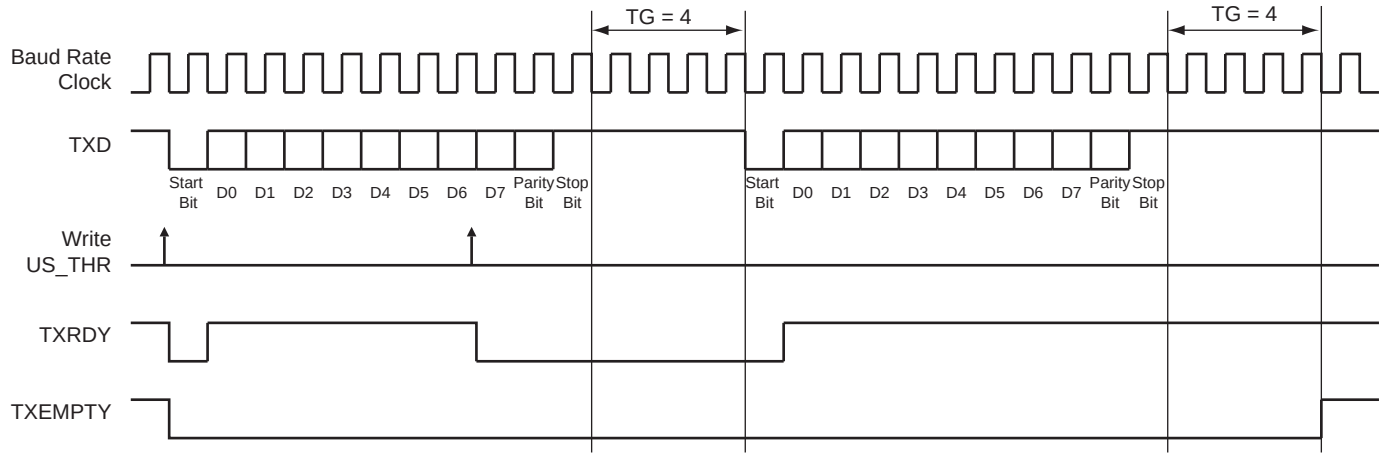


Table 37-9 indicates the maximum length of a timeguard period that the transmitter can handle depending on the baud rate.

Table 37-9. Maximum Timeguard Length Depending on Baud Rate

Baud Rate (bit/s)	Bit Time (µs)	Timeguard (ms)
1,200	833	212.50
9,600	104	26.56
14,400	69.4	17.71
19,200	52.1	13.28
28,800	34.7	8.85
38,400	26	6.63
56,000	17.9	4.55
57,600	17.4	4.43
115,200	8.7	2.21

37.6.3.11 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 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 written to 0, the Receiver Time-out is disabled and no time-out is detected. The TIMEOUT bit in the US_CSR remains at 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 US_CSR rises. Then, the user can either:

- Stop the counter clock until a new character is received. This is performed by writing a 1 to the STTTO (Start Time-out) bit in the US_CR. 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 a 1 to the RETTO (Reload and Start Time-out) bit in the US_CR. If RETTO is performed, the counter starts counting down immediately from the value TO. This enables generation of a periodic interrupt so that a user time-out can be handled, for example when no key is pressed on a keyboard.

If STTTO is performed, the counter clock is stopped until a first character is received. The idle state on RXD before the start of the frame does not provide a time-out. This prevents having to obtain a periodic interrupt and enables a wait of the end of frame when the idle state on RXD is detected.

If RETTO is performed, the counter starts counting down immediately from the value TO. This enables generation of a periodic interrupt so that a user time-out can be handled, for example when no key is pressed on a keyboard.

Figure 37-23 shows the block diagram of the Receiver Time-out feature.

Figure 37-23. Receiver Time-out Block Diagram

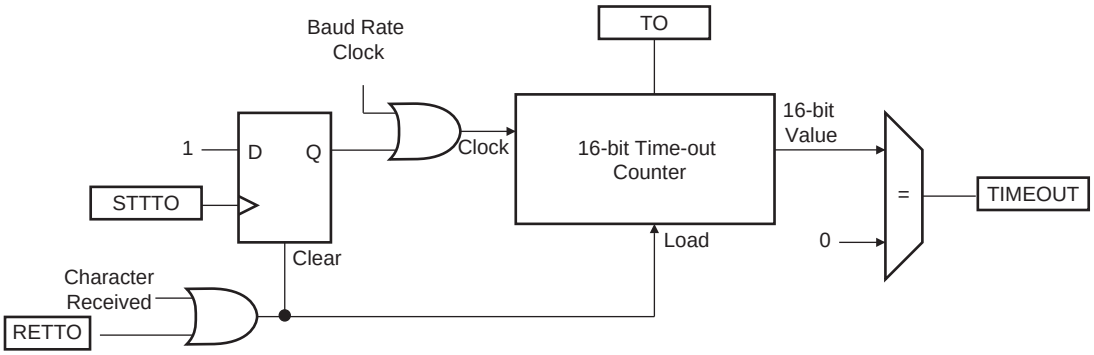


Table 37-10 gives the maximum time-out period for some standard baud rates.

Table 37-10. Maximum Time-out Period

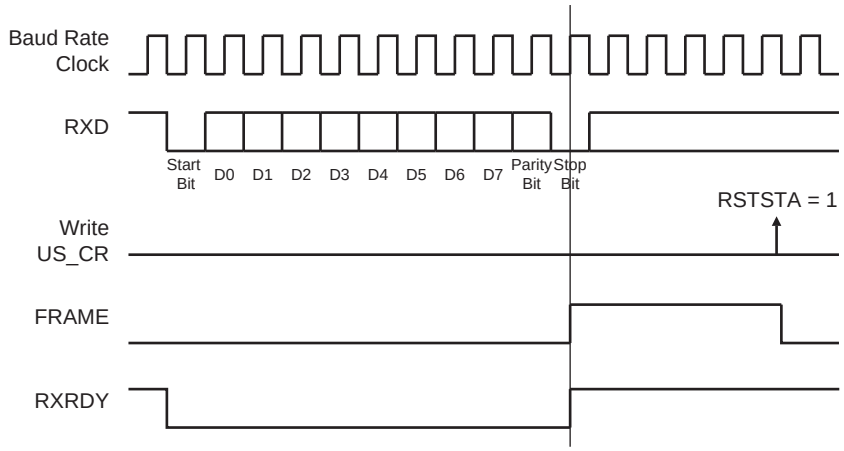
Baud Rate (bit/s)	Bit Time (µs)	Time-out (ms)
600	1,667	109,225
1,200	833	54,613
2,400	417	27,306
4,800	208	13,653
9,600	104	6,827
14,400	69	4,551
19,200	52	3,413
28,800	35	2,276
38,400	26	1,704
56,000	18	1,170
57,600	17	1,138
200,000	5	328

37.6.3.12 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 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 a 1 to the RSTSTA bit in the US_CR.

Figure 37-24. Framing Error Status



37.6.3.13 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 at 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 a 1 to the STTBRK bit in the US_CR. 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 a 1 to the STPBRK bit in the US_CR. 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 processed only if the TXRDY bit in US_CSR is to 1 and the start of the break condition clears the TXRDY and TXEMPTY bits as if a character is processed.

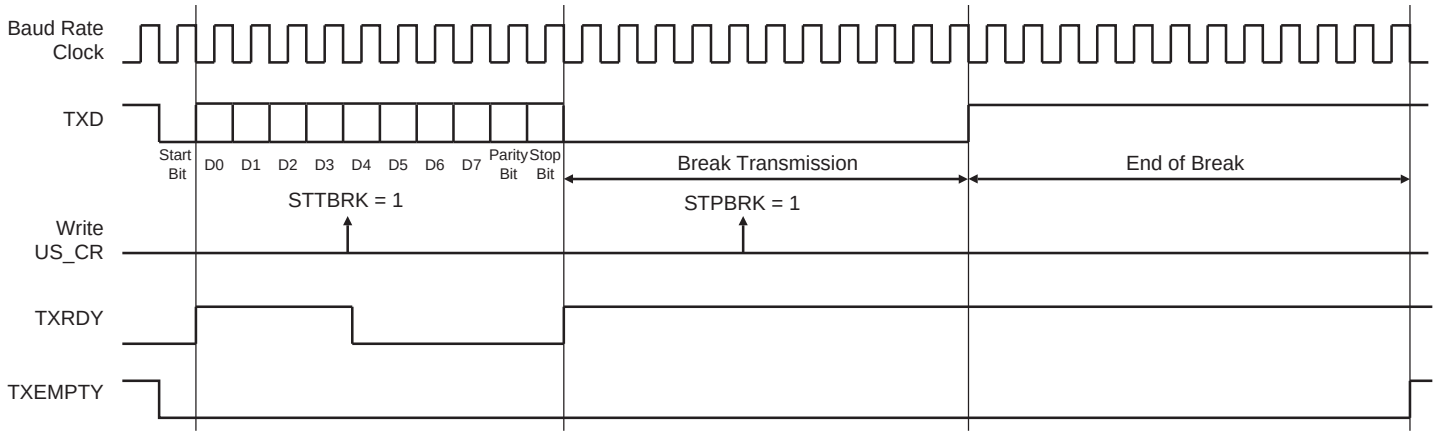
Writing US_CR with both STTBRK and STPBRK bits to 1 can lead to an unpredictable result. All STPBRK commands requested without a previous STTBRK command are ignored. A byte written into the Transmit Holding register while a break is pending, but not started, is ignored.

After the break condition, the transmitter returns the TXD line to 1 for a minimum of 12 bit times. Thus, the transmitter ensures that the remote receiver detects correctly the end of break and the start of the next character. If the timeguard is programmed with a value higher than 12, the TXD line is held high for the timeguard period.

After holding the TXD line for this period, the transmitter resumes normal operations.

Figure 37-25 illustrates the effect of both the Start Break (STTBRK) and Stop Break (STPBRK) commands on the TXD line.

Figure 37-25. Break Transmission



37.6.3.14 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.

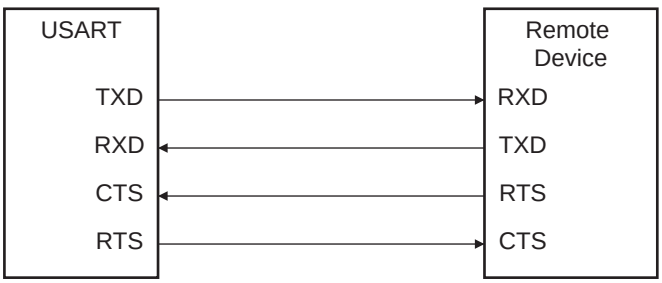
When the low stop bit is detected, the receiver asserts the RXBRK bit in US_CSR. This bit may be cleared by writing a 1 to the RSTSTA bit in the US_CR.

An end of receive break is detected by a high level for at least 2/16 of a bit period in Asynchronous operating mode or one sample at high level in Synchronous operating mode. The end of break detection also asserts the RXBRK bit.

37.6.3.15 Hardware Handshaking

The USART features a hardware handshaking out-of-band flow control. The RTS and CTS pins are used to connect with the remote device, as shown in Figure 37-26.

Figure 37-26. 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 US_MR to the value 0x2.

When hardware handshaking is enabled, the USART displays similar behavior as in standard Synchronous or Asynchronous modes, with the difference that the receiver drives the RTS pin and the level on the CTS pin modifies the behavior of the transmitter, as shown in the figures below. Using this mode requires using the PDC channel for reception. The transmitter can handle hardware handshaking in any case.

Figure 37-27 shows how the receiver operates if hardware handshaking is enabled. The RTS pin is driven high if the receiver is disabled or 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 in the PDC clears the status bit RXBUFF and, as a result, asserts the pin RTS low.

Figure 37-27. Receiver Behavior when Operating with Hardware Handshaking

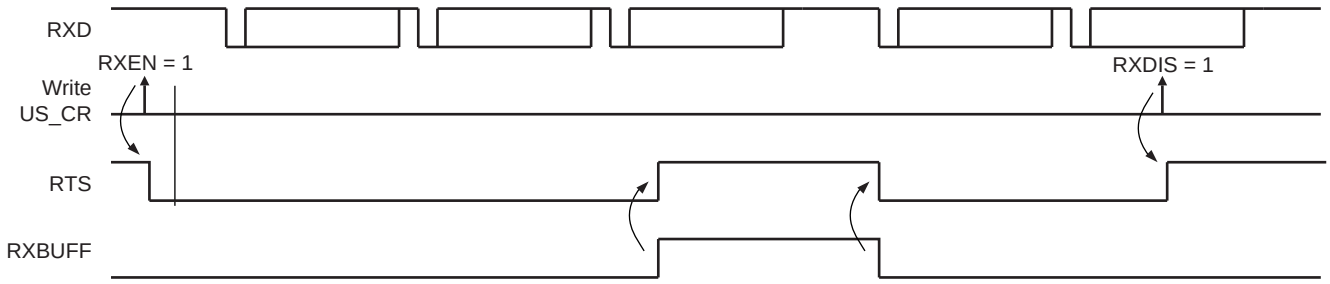
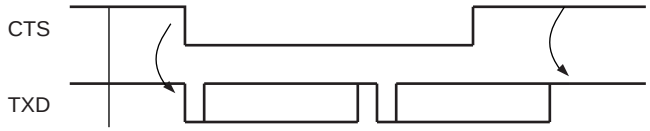


Figure 37-28 shows how the transmitter operates if hardware handshaking is enabled. The CTS pin disables the transmitter. If a character is being processed, the transmitter is disabled only after the completion of the current character and transmission of the next character happens as soon as the pin CTS falls.

Figure 37-28. Transmitter Behavior when Operating with Hardware Handshaking



37.6.4 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.

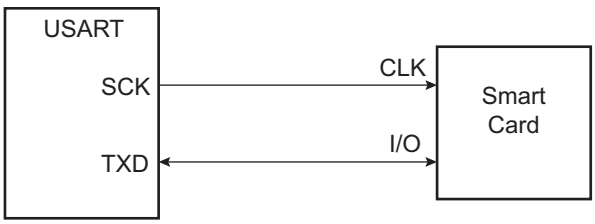
Setting the USART in ISO7816 mode is performed by writing the USART_MODE field in US_MR to the value 0x4 for protocol T = 0 and to the value 0x5 for protocol T = 1.

37.6.4.1 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 Section 37-2 ”Baud Rate Generator”).

The USART connects to a smart card as shown in Figure 37-29. The TXD line becomes bidirectional and the baud rate generator feeds the ISO7816 clock on the SCK pin. As the TXD pin becomes bidirectional, its output remains driven by the output of the transmitter but only when the transmitter is active while its input is directed to the input of the receiver. The USART is considered as the master of the communication as it generates the clock.

Figure 37-29. 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 Section 37.7.3 ”USART Mode Register” and “PAR: Parity Type” .

The USART cannot operate concurrently in both Receiver and Transmitter modes as the communication is unidirectional at a time. It has to be configured according to the required mode by enabling or disabling either the receiver or the transmitter as desired. Enabling both the receiver and the transmitter at the same time in ISO7816 mode may lead to unpredictable results.

The ISO7816 specification defines an inverse transmission format. Data bits of the character must be transmitted on the I/O line at their negative value.

37.6.4.2 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 at 1 during the guard time and the transmitter can continue with the transmission of the next character, as shown in Figure 37-30.

If a parity error is detected by the receiver, it drives the I/O line to 0 during the guard time, as shown in Figure 37-31. This error bit is also named NACK, for Non Acknowledge. In this case, the character lasts 1 bit time more, as the guard time length is the same and is added to the error bit time which lasts 1 bit time.

When the USART is the receiver and it detects an error, it does not load the erroneous character in the Receive Holding register (US_RHR). It appropriately sets the PARE bit in the Status register (US_SR) so that the software can handle the error.

Figure 37-30. T = 0 Protocol without Parity Error

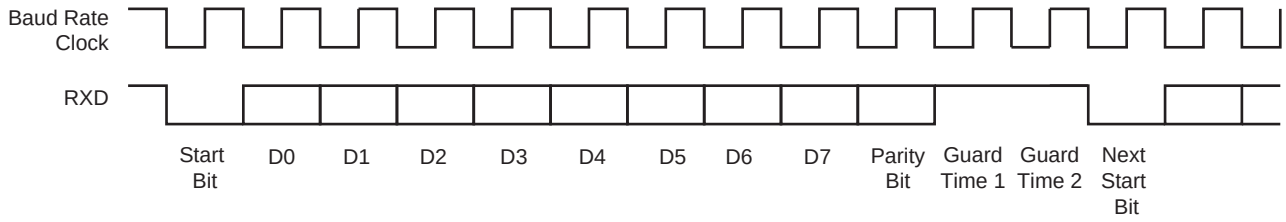
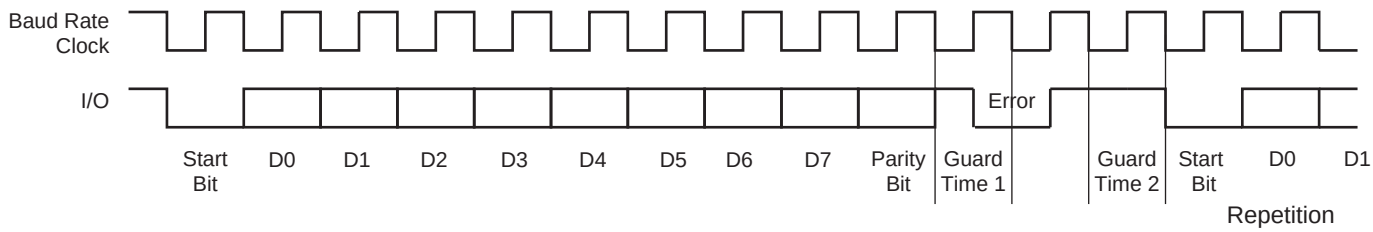


Figure 37-31. T = 0 Protocol with Parity Error



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 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 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 and the last repeated character is not acknowledged, the ITER bit is set in US_CSR. If the repetition of the character is acknowledged by the receiver, the repetitions are stopped and the iteration counter is cleared.

The ITER bit in US_CSR can be cleared by writing a 1 to the RSTIT bit in the US_CR.

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 US_MR. The maximum number of NACKs transmitted is programmed in the MAX_ITERATION field. As soon as MAX_ITERATION is reached, no error signal is driven on the I/O line and the ITER bit in the US_CSR is set.

37.6.4.3 Protocol T = 1

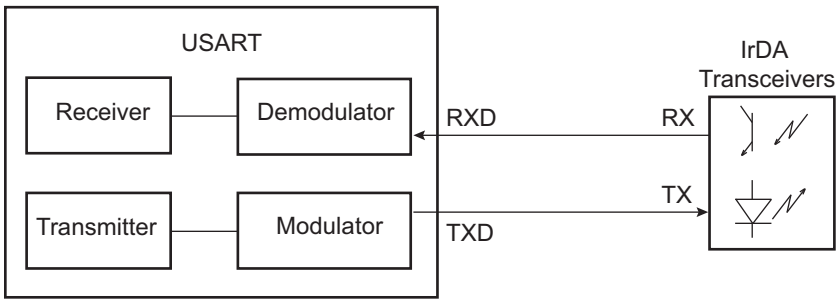
When operating in ISO7816 protocol T = 1, the transmission is similar to an asynchronous format with only one stop bit. The parity is generated when transmitting and checked when receiving. Parity error detection sets the PARE bit in the US_CSR.

37.6.5 IrDA Mode

The USART features an IrDA mode supplying half-duplex point-to-point wireless communication. It embeds the modulator and demodulator which allows a glueless connection to the infrared transceivers, as shown in Figure 37-32. The modulator and demodulator are compliant with the IrDA specification version 1.1 and support data transfer speeds ranging from 2.4 kbit/s to 115.2 kbit/s.

The IrDA mode is enabled by setting the USART_MODE field in US_MR to the value 0x8. The IrDA Filter register (US_IF) is used to configure the demodulator filter. The USART transmitter and receiver operate in a normal Asynchronous mode and all parameters are accessible. Note that the modulator and the demodulator are activated.

Figure 37-32. Connection to IrDA Transceivers



The receiver and the transmitter must be enabled or disabled depending on the direction of the transmission to be managed.

To receive IrDA signals, the following needs to be done:

- Disable TX and Enable RX
- Configure the TXD pin as PIO and set it as an output to 0 (to avoid LED emission). Disable the internal pull-up (better for power consumption).

- Receive data

37.6.5.1 IrDA Modulation

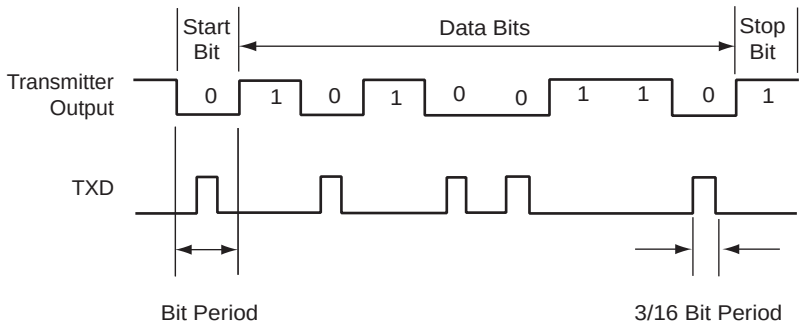
For baud rates up to and including 115.2 kbit/s, the RZI modulation scheme is used. “0” is represented by a light pulse of 3/16th of a bit time. Some examples of signal pulse duration are shown in Table 37-11.

Table 37-11. IrDA Pulse Duration

Baud Rate	Pulse Duration (3/16)
2.4 kbit/s	78.13 µs
9.6 kbit/s	19.53 µs
19.2 kbit/s	9.77 µs
38.4 kbit/s	4.88 µs
57.6 kbit/s	3.26 µs
115.2 kbit/s	1.63 µs

Figure 37-33 shows an example of character transmission.

Figure 37-33. IrDA Modulation



37.6.5.2 IrDA Baud Rate

Table 37-12 gives some examples of CD values, baud rate error and pulse duration. Note that the requirement on the maximum acceptable error of ±1.87% must be met.

Table 37-12. IrDA Baud Rate Error

Peripheral Clock	Baud Rate (bit/s)	CD	Baud Rate Error	Pulse Time (µs)
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

Table 37-12. IrDA Baud Rate Error (Continued)

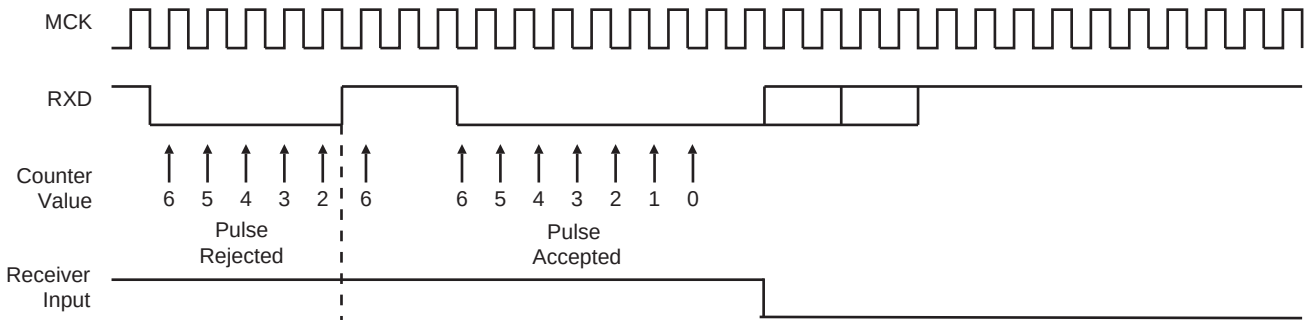
Peripheral Clock	Baud Rate (bit/s)	CD	Baud Rate Error	Pulse Time (µs)
3,686,400	19,200	12	0.00%	9.77
20,000,000	19,200	65	0.16%	9.77
32,768,000	19,200	107	0.31%	9.77
40,000,000	19,200	130	0.16%	9.77
3,686,400	9,600	24	0.00%	19.53
20,000,000	9,600	130	0.16%	19.53
32,768,000	9,600	213	0.16%	19.53
40,000,000	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

37.6.5.3 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 peripheral clock speed. If a rising edge is detected on the RXD pin, the counter stops and is reloaded with US_IF. If no rising edge is detected when the counter reaches 0, the input of the receiver is driven low during one bit time.

Figure 37-34 illustrates the operations of the IrDA demodulator.

Figure 37-34. IrDA Demodulator Operations



The programmed value in the US_IF register must always meet the following criteria:

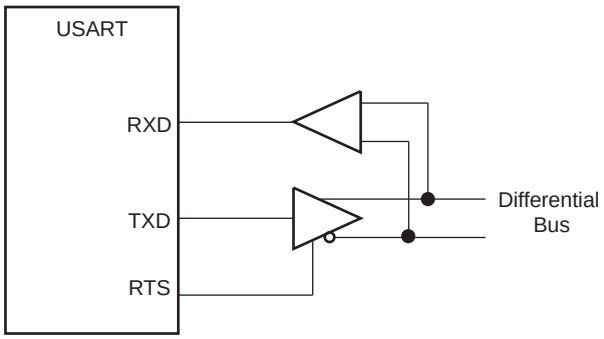
$$t_{peripheral\ clock} \times (IRDA\_FILTER + 3) < 1.41\ \mu s$$

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 make sure IrDA communications operate correctly.

37.6.6 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 an RS485 bus is shown in Figure 37-35.

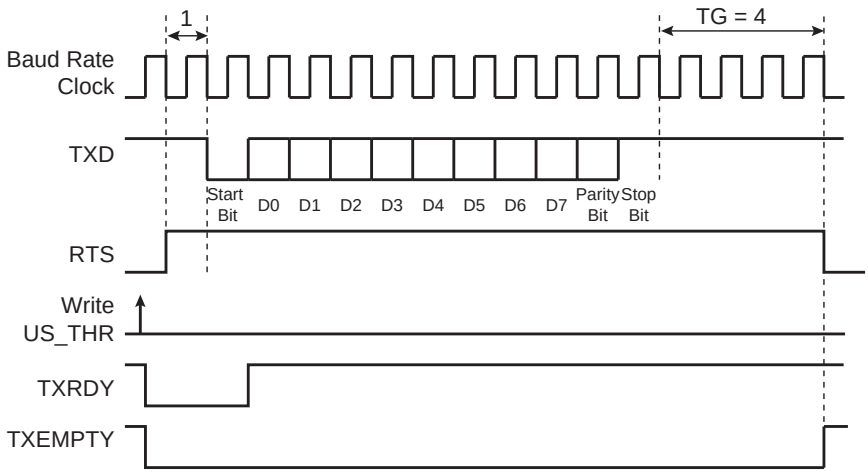
Figure 37-35. Typical Connection to a RS485 Bus



The USART is set in RS485 mode by writing the value 0x1 to the USART_MODE field in US_MR.

The RTS pin is at a level inverse to the TXEMPTY bit. Significantly, the RTS pin remains high when a timeguard is programmed so that the line can remain driven after the last character completion. Figure 37-36 gives an example of the RTS waveform during a character transmission when the timeguard is enabled.

Figure 37-36. Example of RTS Drive with Timeguard



37.6.7 Modem Mode

The USART features Modem mode, which enables control of the signals: DTR (Data Terminal Ready), DSR (Data Set Ready), RTS (Request to Send), CTS (Clear to Send), DCD (Data Carrier Detect) and RI (Ring Indicator). While operating in Modem mode, the USART behaves as a DTE (Data Terminal Equipment) as it drives DTR and RTS and can detect level change on DSR, DCD, CTS and RI.

Setting the USART in Modem mode is performed by writing the USART_MODE field in US_MR to the value 0x3. While operating in Modem mode, the USART behaves as though in Asynchronous mode and all the parameter configurations are available.

Table 37-13 gives the correspondence of the USART signals with modem connection standards.

Table 37-13. Circuit References

USART Pin	V24	CCITT	Direction
TXD	2	103	From terminal to modem
RTS	4	105	From terminal to modem
DTR	20	108.2	From terminal to modem

Table 37-13. Circuit References

USART Pin	V24	CCITT	Direction
RXD	3	104	From modem to terminal
CTS	5	106	From terminal to modem
DSR	6	107	From terminal to modem
DCD	8	109	From terminal to modem
RI	22	125	From terminal to modem

The control of the DTR output pin is performed by writing a 1 to the DTRDIS and DTREN bits respectively in US_CR. The disable command forces the corresponding pin to its inactive level, i.e., high. The enable command forces the corresponding pin to its active level, i.e., low. The RTS output pin is automatically controlled in this mode.

The level changes are detected on the RI, DSR, DCD and CTS pins. If an input change is detected, the RIIC, DSRIC, DCDIC and CTSIC bits in US_CSR are set respectively and can trigger an interrupt. The status is automatically cleared when US_CSR is read. Furthermore, the CTS automatically disables the transmitter when it is detected at its inactive state. If a character is being transmitted when the CTS rises, the character transmission is completed before the transmitter is actually disabled.

37.6.8 SPI Mode

The Serial Peripheral Interface (SPI) mode is a synchronous serial data link that provides communication with external devices in Master or Slave mode. It also enables communication between processors if an external processor is connected to the system.

The Serial Peripheral Interface is essentially a shift register that serially transmits data bits to other SPIs. During a data transfer, one SPI system acts as the “master” which controls the data flow, while the other devices act as “slaves" which have data shifted into and out by the master. Different CPUs can take turns being masters and one master may simultaneously shift data into multiple slaves. (Multiple master protocol is the opposite of single master protocol, where one CPU is always the master while all of the others are always slaves.) However, only one slave may drive its output to write data back to the master at any given time.

A slave device is selected when its NSS signal is asserted by the master. The USART in SPI Master mode can address only one SPI slave because it can generate only one NSS signal.

The SPI system consists of two data lines and two control lines:

- Master Out Slave In (MOSI): This data line supplies the output data from the master shifted into the input of the slave.
- Master In Slave Out (MISO): This data line supplies the output data from a slave to the input of the master.
- Serial Clock (SCK): This control line is driven by the master and regulates the flow of the data bits. The master may transmit data at a variety of baud rates. The SCK line cycles once for each bit that is transmitted.
- Slave Select (NSS): This control line allows the master to select or deselect the slave.

37.6.8.1 Modes of Operation

The USART can operate in SPI Master mode or in SPI Slave mode.

Operation in SPI Master mode is programmed by writing 0xE to the USART_MODE field in US_MR. 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 US_MR. 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 unpredictable behavior, any change of the SPI mode must be followed by a software reset of the transmitter and of the receiver (except the initial configuration after a hardware reset). (See Section 37.6.8.4).

37.6.8.2 Baud Rate

In SPI mode, the baud rate generator operates in the same way as in USART Synchronous mode. See Section 37.6.1.3 "Baud Rate in Synchronous Mode or SPI Mode". 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 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 divided peripheral clock 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 peripheral clock is selected.

In SPI Slave mode:

- The external clock (SCK) selection is forced regardless of the value of the USCLKS field in the US_MR. Likewise, the value written in US_BRGR has no effect, because the clock is provided directly by the signal on the USART SCK pin.
- To obtain correct behavior of the receiver and the transmitter, the external clock (SCK) frequency must be at least 6 times lower than the system clock.

37.6.8.3 Data Transfer

Up to nine 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 US_MR. The nine 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 US_MR. The clock phase is programmed with the CPHA bit. These two parameters determine the edges of the clock signal upon which data is driven and sampled. Each of the two parameters has two possible states, resulting in four possible combinations that are incompatible with one another. Thus, a master/slave pair must use the same parameter pair values to communicate. If multiple slaves are used and fixed in different configurations, the master must reconfigure itself each time it needs to communicate with a different slave.

Table 37-14. SPI Bus Protocol Mode

SPI Bus Protocol Mode	CPOL	CPHA
0	0	1
1	0	0
2	1	1
3	1	0

Figure 37-37. SPI Transfer Format (CPHA = 1, 8 bits per transfer)

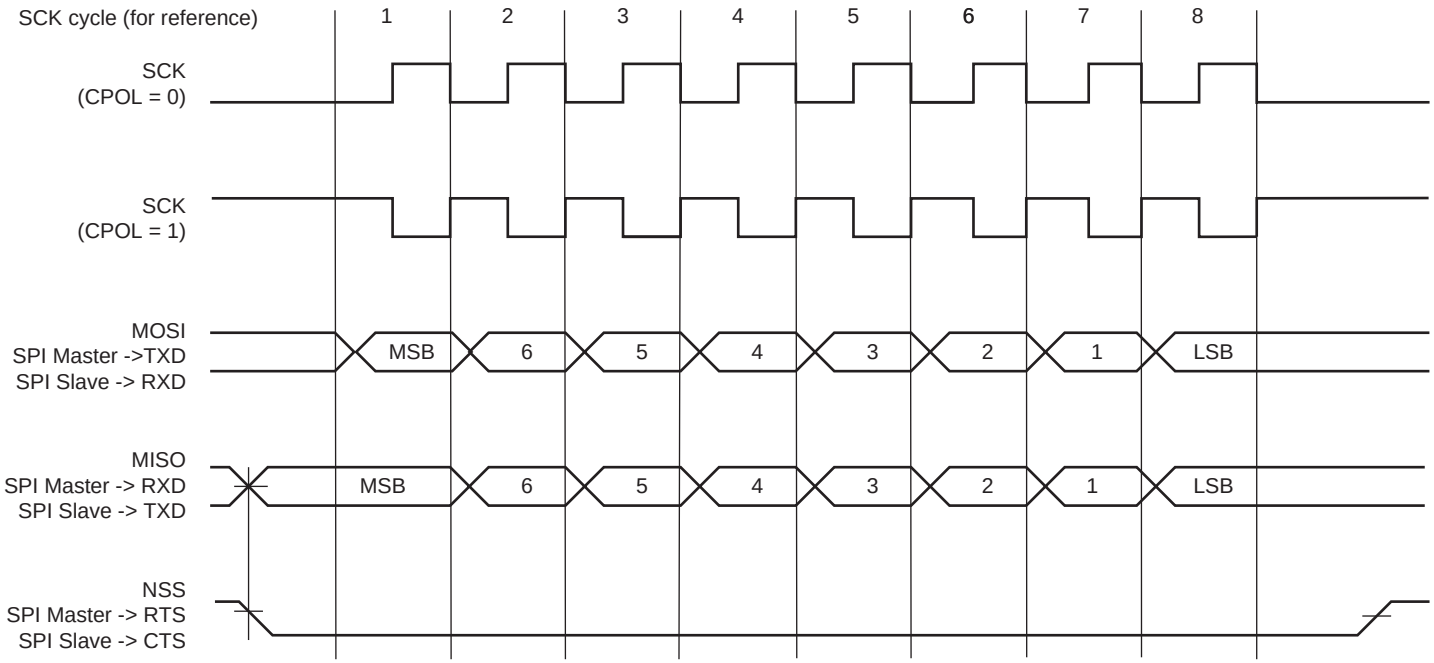
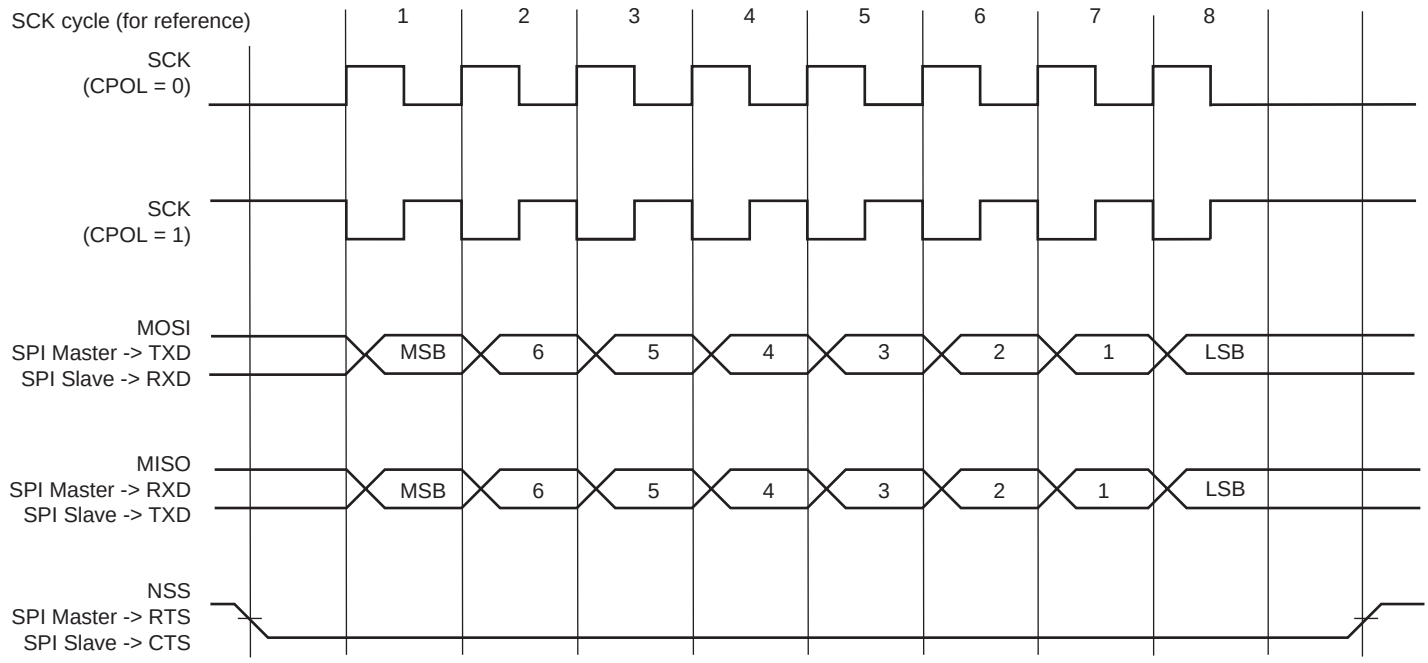


Figure 37-38. SPI Transfer Format (CPHA = 0, 8 bits per transfer)



37.6.8.4 Receiver and Transmitter Control

See Section 37.6.2 "Receiver and Transmitter Control"

37.6.8.5 Character Transmission

The characters are sent by writing in the Transmit Holding register (US_THR). An additional condition for transmitting a character can be added when the USART is configured in SPI Master mode. In the USART Mode Register (SPI_MODE) (USART_MR), the value configured on the bit WRDBT can prevent any character

transmission (even if US_THR has been written) while the receiver side is not ready (character not read). When WRDBT equals 0, the character is transmitted whatever the receiver status. If WRDBT is set to 1, the transmitter waits for the Receive Holding register (US_RHR) to be read before transmitting the character (RXRDY flag cleared), thus preventing any overflow (character loss) on the receiver side.

The chip select line is de-asserted for a period equivalent to three bits between the transmission of two data.

The transmitter reports two status bits in 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 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 a 1 to the RSTSTA (Reset Status) bit in US_CR.

In SPI Master mode, the slave select line (NSS) is asserted at low level one t_{bit} (t_{bit} being the nominal time required to transmit a bit) before the transmission of the MSB bit and released at high level one t_{bit} 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 three t_{bit} 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 a 1 to the RCS bit in the US_CR. The slave select line (NSS) can be released at high level only by writing a 1 to the FCS bit in the US_CR (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 one t_{bit} before the first serial clock cycle corresponding to the MSB bit.

37.6.8.6 Character Reception

When a character reception is completed, it is transferred to the Receive Holding register (US_RHR) and the RXRDY bit in the Status register (US_CSR) rises. If a character is completed while RXRDY is set, the OVRE (Overrun Error) bit is set. The last character is transferred into US_RHR and overwrites the previous one. The OVRE bit is cleared by writing a 1 to the RSTSTA (Reset Status) bit in the US_CR.

To ensure correct behavior of the receiver in SPI Slave mode, the master device sending the frame must ensure a minimum delay of one t_{bit} 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 one t_{bit} before the first serial clock cycle corresponding to the MSB bit.

37.6.8.7 Receiver Timeout

Because the receiver baud rate 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 US_RTOR.

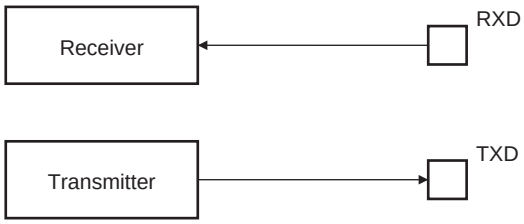
37.6.9 Test Modes

The USART can be programmed to operate in three different test modes. The internal loopback capability allows on-board diagnostics. In Loopback mode, the USART interface pins are disconnected or not and reconfigured for loopback internally or externally.

37.6.9.1 Normal Mode

Normal mode connects the RXD pin on the receiver input and the transmitter output on the TXD pin.

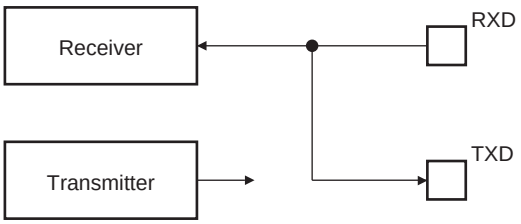
Figure 37-39. Normal Mode Configuration



37.6.9.2 Automatic Echo Mode

Automatic Echo mode allows bit-by-bit retransmission. When a bit is received on the RXD pin, it is sent to the TXD pin, as shown in Figure 37-40. Programming the transmitter has no effect on the TXD pin. The RXD pin is still connected to the receiver input, thus the receiver remains active.

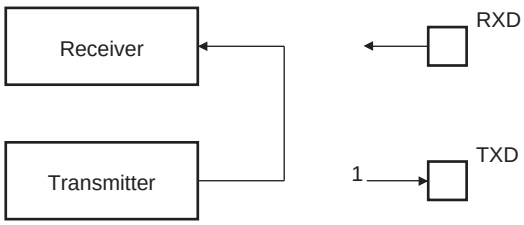
Figure 37-40. Automatic Echo Mode Configuration



37.6.9.3 Local Loopback Mode

Local Loopback mode connects the output of the transmitter directly to the input of the receiver, as shown in Figure 37-41. The TXD and RXD pins are not used. The RXD pin has no effect on the receiver and the TXD pin is continuously driven high, as in idle state.

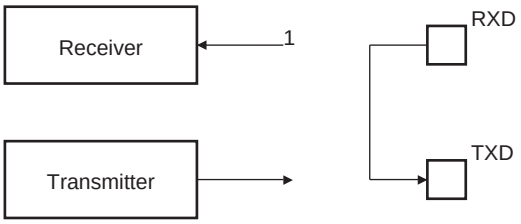
Figure 37-41. Local Loopback Mode Configuration



37.6.9.4 Remote Loopback Mode

Remote Loopback mode directly connects the RXD pin to the TXD pin, as shown in Figure 37-42. The transmitter and the receiver are disabled and have no effect. This mode allows bit-by-bit retransmission.

Figure 37-42. Remote Loopback Mode Configuration



37.6.10 Register Write Protection

To prevent any single software error from corrupting USART behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the USART Write Protection Mode Register (US_WPMR).

If a write access to a write-protected register is detected, the WPVS flag in the USART Write Protection Status Register (US_WPSR) is set and the field WPVSRC indicates the register in which the write access has been attempted.

The WPVS bit is automatically cleared after reading the US_WPSR.

The following registers can be write-protected:

- USART Mode Register
- USART Baud Rate Generator Register
- USART Receiver Time-out Register
- USART Transmitter Timeguard Register
- USART FI DI RATIO Register
- USART IrDA Filter Register
- USART Manchester Configuration Register

37.7 Universal Synchronous Asynchronous Receiver Transmitter (USART) User Interface

Table 37-15. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	Control Register	US_CR	Write-only	–
0x0004	Mode Register	US_MR	Read/Write	0x0
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	0x0
0x0018	Receive Holding Register	US_RHR	Read-only	0x0
0x001C	Transmit 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
0x002C–0x003C	Reserved	–	–	–
0x0040	FI DI Ratio Register	US_FIDI	Read/Write	0x174
0x0044	Number of Errors Register	US_NER	Read-only	0x0
0x0048	Reserved	–	–	–
0x004C	IrDA Filter Register	US_IF	Read/Write	0x0
0x0050	Manchester Configuration Register	US_MAN	Read/Write	0x30011004
0x0054–0x005C	Reserved	–	–	–
0x0060–0x00E0	Reserved	–	–	–
0x00E4	Write Protection Mode Register	US_WPMR	Read/Write	0x0
0x00E8	Write Protection Status Register	US_WPSR	Read-only	0x0
0x00EC–0x00FC	Reserved	–	–	–
0x0100–0x0128	Reserved for PDC Registers	–	–	–

37.7.1 USART Control Register

Name: US_CR

Address: 0x400A0000 (0), 0x400A4000 (1)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	RTSDIS	RTSEN	DTRDIS	DTREN
15	14	13	12	11	10	9	8
RETTO	RSTNACK	RSTIT	SENDA	STTTO	STPBRK	STTBRK	RSTSTA
7	6	5	4	3	2	1	0
TXDIS	TXEN	RXDIS	RXEN	RSTTX	RSTRX	–	–

For SPI control, see Section 37.7.2 ”USART Control Register (SPI_MODE)”.

- **RSTRX: Reset Receiver**

0: No effect.

1: Resets the receiver.

- **RSTTX: Reset Transmitter**

0: No effect.

1: Resets the transmitter.

- **RXEN: Receiver Enable**

0: No effect.

1: Enables the receiver, if RXDIS is 0.

- **RXDIS: Receiver Disable**

0: No effect.

1: Disables the receiver.

- **TXEN: Transmitter Enable**

0: No effect.

1: Enables the transmitter if TXDIS is 0.

- **TXDIS: Transmitter Disable**

0: No effect.

1: Disables the transmitter.

- **RSTSTA: Reset Status Bits**

0: No effect.

1: Resets the status bits PARE, FRAME, OVRE, MANERR and RXBRK in US_CSR.

- **STTBRK: Start Break**

0: No effect.

1: Starts transmission of a break after the characters present in US_THR and the Transmit Shift Register have been transmitted. No effect if a break is already being transmitted.

- **STPBRK: Stop Break**

0: No effect.

1: Stops transmission of the break after a minimum of one character length and transmits a high level during 12-bit periods. No effect if no break is being transmitted.

- **STTTO: Clear TIMEOUT Flag and Start Time-out After Next Character Received**

0: No effect.

1: Starts waiting for a character before enabling the time-out counter. Immediately disables a time-out period in progress. 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 ITER 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: Start Time-out Immediately**

0: No effect

1: Immediately restarts time-out period.

- **DTREN: Data Terminal Ready Enable**

0: No effect.

1: Drives the pin DTR to 0.

- **DTRDIS: Data Terminal Ready Disable**

0: No effect.

1: Drives the pin DTR to 1.

- **RTSEN: Request to Send Pin Control**

0: No effect.

1: Drives RTS pin to 0 if US_MR.USART_MODE field = 0.

- **RTSDIS: Request to Send Pin Control**

0: No effect.

1: Drives RTS pin to 1 if US_MR.USART_MODE field = 0.

37.7.2 USART Control Register (SPI_MODE)

Name: US_CR (SPI_MODE)

Address: 0x400A0000 (0), 0x400A4000 (1)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	RCS	FCS	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	RSTSTA
7	6	5	4	3	2	1	0
TXDIS	TXEN	RXDIS	RXEN	RSTTX	RSTRX	–	–

This configuration is relevant only if USART_MODE = 0xE or 0xF in the USART Mode Register.

- **RSTRX: Reset Receiver**

0: No effect.

1: Resets the receiver.

- **RSTTX: Reset Transmitter**

0: No effect.

1: Resets the transmitter.

- **RXEN: Receiver Enable**

0: No effect.

1: Enables the receiver, if RXDIS is 0.

- **RXDIS: Receiver Disable**

0: No effect.

1: Disables the receiver.

- **TXEN: Transmitter Enable**

0: No effect.

1: Enables the transmitter if TXDIS is 0.

- **TXDIS: Transmitter Disable**

0: No effect.

1: Disables the transmitter.

- **RSTSTA: Reset Status Bits**

0: No effect.

1: Resets the status bits OVRE, UNRE in US_CSR.

- **FCS: Force SPI Chip Select**

Applicable if USART operates in SPI Master mode (USART_MODE = 0xE):

0: No effect.

1: Forces the Slave Select Line NSS (RTS pin) to 0, even if USART is not transmitting, in order to address SPI slave devices supporting the CSAAT mode (Chip Select Active After Transfer).

- **RCS: Release SPI Chip Select**

Applicable if USART operates in SPI Master mode (USART_MODE = 0xE):

0: No effect.

1: Releases the Slave Select Line NSS (RTS pin).

37.7.3 USART Mode Register

Name: US_MR

Address: 0x400A0004 (0), 0x400A4004 (1)

Access: Read/Write

31	30	29	28	27	26	25	24
ONEBIT	MODSYNC	MAN	FILTER	–	MAX_ITERATION		
23	22	21	20	19	18	17	16
INVDATA	VAR_SYNC	DSNACK	INACK	OVER	CLKO	MODE9	MSBF
15	14	13	12	11	10	9	8
CHMODE		NBSTOP		PAR			SYNC
7	6	5	4	3	2	1	0
CHRL		USCLKS		USART_MODE			

This register can only be written if the WPEN bit is cleared in the USART Write Protection Mode Register.

For SPI configuration, see Section 37.7.4 ”USART Mode Register (SPI_MODE)”.

- **USART_MODE: USART Mode of Operation**

Value	Name	Description
0x0	NORMAL	Normal mode
0x1	RS485	RS485
0x2	HW_HANDSHAKING	Hardware Handshaking
0x3	MODEM	Modem
0x4	IS07816_T_0	IS07816 Protocol: T = 0
0x6	IS07816_T_1	IS07816 Protocol: T = 1
0x8	IRDA	IrDA
0xE	SPI_MASTER	SPI master mode (CLKO must be written to 1 and USCLKS = 0, 1 or 2)
0xF	SPI_SLAVE	SPI Slave mode

The PDC transfers are supported in all USART modes of operation.

- **USCLKS: Clock Selection**

Value	Name	Description
0	MCK	Peripheral clock is selected
1	DIV	Peripheral clock divided (DIV=8) is selected
2	—	Reserved
3	SCK	Serial clock (SCK) is selected

- **CHRL: Character Length**

Value	Name	Description
0	5_BIT	Character length is 5 bits
1	6_BIT	Character length is 6 bits
2	7_BIT	Character length is 7 bits
3	8_BIT	Character length is 8 bits

- **SYNC: Synchronous Mode Select**

0: USART operates in Asynchronous mode.

1: USART operates in Synchronous mode.

- **PAR: Parity Type**

Value	Name	Description
0	EVEN	Even parity
1	ODD	Odd parity
2	SPACE	Parity forced to 0 (Space)
3	MARK	Parity forced to 1 (Mark)
4	NO	No parity
6	MULTIDROP	Multidrop mode

- **NBSTOP: Number of Stop Bits**

Value	Name	Description
0	1_BIT	1 stop bit
1	1_5_BIT	1.5 stop bit (SYNC = 0) or reserved (SYNC = 1)
2	2_BIT	2 stop bits

- **CHMODE: Channel Mode**

Value	Name	Description
0	NORMAL	Normal mode
1	AUTOMATIC	Automatic Echo. Receiver input is connected to the TXD pin.
2	LOCAL_LOOPBACK	Local Loopback. Transmitter output is connected to the Receiver Input.
3	REMOTE_LOOPBACK	Remote Loopback. RXD pin is internally connected to the TXD pin.

- **MSBF: Bit Order**

0: Least significant bit is sent/received first.

1: Most significant bit is sent/received first.

- **MODE9: 9-bit Character Length**

0: CHRL defines character length

1: 9-bit character length

- **CLKO: Clock Output Select**

0: The USART does not drive the SCK pin.

1: The USART drives the SCK pin if USCLKS does not select the external clock SCK.

- **OVER: Oversampling Mode**

0: 16 × Oversampling

1: 8 × Oversampling

- **INACK: Inhibit Non Acknowledge**

0: The NACK is generated.

1: The NACK is not generated.

- **DSNACK: Disable Successive NACK**

0: NACK is sent on the ISO line as soon as a parity error occurs in the received character (unless INACK is set).

1: Successive parity errors are counted up to the value specified in the MAX_ITERATION field. These parity errors generate a NACK on the ISO line. As soon as this value is reached, no additional NACK is sent on the ISO line. The flag ITER is asserted.

Note: MAX_ITERATION field must be set to 0 if DSNACK is cleared.

- **INVDATA: Inverted Data**

0: The data field transmitted on TXD line is the same as the one written in US_THR 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 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.

- **MAX_ITERATION: Maximum Number of Automatic Iteration**

0–7: Defines the maximum number of iterations in mode ISO7816, protocol T = 0.

- **FILTER: Receive Line Filter**

0: The USART does not filter the receive line.

1: The USART filters the receive line using a three-sample filter (1/16-bit clock) (2 over 3 majority).

- **MAN: Manchester Encoder/Decoder Enable**

0: Manchester encoder/decoder are disabled.

1: Manchester encoder/decoder are enabled.

- **MODSYNC: Manchester Synchronization Mode**

0:The Manchester start bit is a 0 to 1 transition

1: The Manchester start bit is a 1 to 0 transition.

- **ONEBIT: Start Frame Delimiter Selector**

0: Start frame delimiter is COMMAND or DATA SYNC.

1: Start frame delimiter is one bit.

37.7.4 USART Mode Register (SPI_MODE)

Name: US_MR (SPI_MODE)

Address: 0x400A0004 (0), 0x400A4004 (1)

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	WRDBT	–	CLKO	–	CPOL
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	CPHA
7	6	5	4	3	2	1	0
CHRL		USCLKS		USART_MODE			

This configuration is relevant only if USART_MODE = 0xE or 0xF in the USART Mode Register.
This register can only be written if the WPEN bit is cleared in the USART Write Protection Mode Register.

- **USART_MODE: USART Mode of Operation**

Value	Name	Description
0xE	SPI_MASTER	SPI master
0xF	SPI_SLAVE	SPI Slave

- **USCLKS: Clock Selection**

Value	Name	Description
0	MCK	Peripheral clock is selected
1	DIV	Peripheral clock divided (DIV=8) is selected
3	SCK	Serial Clock SLK is selected

- **CHRL: Character Length**

Value	Name	Description
3	8_BIT	Character length is 8 bits

- **CPHA: SPI Clock Phase**
  – Applicable if USART operates in SPI mode (USART_MODE = 0xE or 0xF):

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.

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.

- **CPOL: SPI Clock Polarity**

Applicable if USART operates in SPI mode (Slave or Master, USART_MODE = 0xE or 0xF):

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 CPHA to produce the required clock/data relationship between master and slave devices.

- **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.

- **WRDBT: Wait Read Data Before Transfer**

0: The character transmission starts as soon as a character is written into US_THR (assuming TXRDY was set).

1: The character transmission starts when a character is written and only if RXRDY flag is cleared (Receive Holding Register has been read).

37.7.5 USART Interrupt Enable Register

Name: US_IER

Address: 0x400A0008 (0), 0x400A4008 (1)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	MANE
23	22	21	20	19	18	17	16
–	–	–	–	CTSIC	DCDIC	DSRIC	RIIC
15	14	13	12	11	10	9	8
–	–	NACK	RXBUFF	TXBUFE	ITER	TXEMPTY	TIMEOUT
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY

For SPI specific configuration, see Section 37.7.6 "USART Interrupt Enable Register (SPI_MODE)".

The following configuration values are valid for all listed bit names of this register:

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 Buffer Interrupt Enable (available in all USART modes of operation)**
- **ENDTX: End of Transmit Buffer Interrupt Enable (available in all USART modes of operation)**
- **OVRE: Overrun Error Interrupt Enable**
- **FRAME: Framing Error Interrupt Enable**
- **PARE: Parity Error Interrupt Enable**
- **TIMEOUT: Time-out Interrupt Enable**
- **TXEMPTY: TXEMPTY Interrupt Enable**
- **ITER: Max number of Repetitions Reached Interrupt Enable**
- **TXBUFE: Transmit Buffer Empty Interrupt Enable (available in all USART modes of operation)**
- **RXBUFF: Receive Buffer Full Interrupt Enable (available in all USART modes of operation)**
- **NACK: Non Acknowledge Interrupt Enable**
- **RIIC: Ring Indicator Input Change Enable**

- **DSRIC: Data Set Ready Input Change Enable**
- **DCDIC: Data Carrier Detect Input Change Interrupt Enable**
- **CTSIC: Clear to Send Input Change Interrupt Enable**
- **MANE: Manchester Error Interrupt Enable**

37.7.6 USART Interrupt Enable Register (SPI_MODE)

Name: US_IER (SPI_MODE)

Address: 0x400A0008 (0), 0x400A4008 (1)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	NSSE	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	UNRE	TXEMPTY	–
7	6	5	4	3	2	1	0
–	–	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

This configuration is relevant only if USART_MODE = 0xE or 0xF in the USART Mode Register.

The following configuration values are valid for all listed bit names of this register:

0: No effect

1: Enables the corresponding interrupt.

- **RXRDY: RXRDY Interrupt Enable**
- **TXRDY: TXRDY Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable**
- **ENDTX: End of Transmit Buffer Interrupt Enable**
- **OVRE: Overrun Error Interrupt Enable**
- **TXEMPTY: TXEMPTY Interrupt Enable**
- **UNRE: SPI Underrun Error Interrupt Enable**
- **TXBUFE: Transmit Buffer Empty Interrupt Enable**
- **RXBUFF: Receive Buffer Full Interrupt Enable**
- **NSSE: NSS Line (Driving CTS Pin) Rising or Falling Edge Event Interrupt Enable**

37.7.7 USART Interrupt Disable Register

Name: US_IDR

Address: 0x400A000C (0), 0x400A400C (1)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	MANE
23	22	21	20	19	18	17	16
–	–	–	–	CTSIC	DCDIC	DSRIC	RIIC
15	14	13	12	11	10	9	8
–	–	NACK	RXBUFF	TXBUFE	ITER	TXEMPTY	TIMEOUT
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY

For SPI specific configuration, see Section 37.7.8 ”USART Interrupt Disable Register (SPI_MODE)”.

The following configuration values are valid for all listed bit names of this register:

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 Buffer Transfer Interrupt Disable (available in all USART modes of operation)**
- **ENDTX: End of Transmit Buffer Interrupt Disable (available in all USART modes of operation)**
- **OVRE: Overrun Error Interrupt Enable**
- **FRAME: Framing Error Interrupt Disable**
- **PARE: Parity Error Interrupt Disable**
- **TIMEOUT: Time-out Interrupt Disable**
- **TXEMPTY: TXEMPTY Interrupt Disable**
- **ITER: Max Number of Repetitions Reached Interrupt Disable**
- **TXBUFE: Transmit Buffer Empty Interrupt Disable (available in all USART modes of operation)**
- **RXBUFF: Receive Buffer Full Interrupt Disable (available in all USART modes of operation)**
- **NACK: Non Acknowledge Interrupt Disable**
- **RIIC: Ring Indicator Input Change Disable**

- **DSRIC: Data Set Ready Input Change Disable**
- **DCDIC: Data Carrier Detect Input Change Interrupt Disable**
- **CTSIC: Clear to Send Input Change Interrupt Disable**
- **MANE: Manchester Error Interrupt Disable**

37.7.8 USART Interrupt Disable Register (SPI_MODE)

Name: US_IDR (SPI_MODE)

Address: 0x400A000C (0), 0x400A400C (1)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	NSSE	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	UNRE	TXEMPTY	–
7	6	5	4	3	2	1	0
–	–	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

This configuration is relevant only if USART_MODE = 0xE or 0xF in the USART Mode Register.

The following configuration values are valid for all listed bit names of this register:

0: No effect

1: Disables the corresponding interrupt.

- **RXRDY: RXRDY Interrupt Disable**
- **TXRDY: TXRDY Interrupt Disable**
- **ENDRX: End of Receive Buffer Transfer Interrupt Disable**
- **ENDTX: End of Transmit Buffer Interrupt Disable**
- **OVRE: Overrun Error Interrupt Disable**
- **TXEMPTY: TXEMPTY Interrupt Disable**
- **UNRE: SPI Underrun Error Interrupt Disable**
- **TXBUFE: Transmit Buffer Empty Interrupt Disable**
- **RXBUFF: Receive Buffer Full Interrupt Disable**
- **NSSE: NSS Line (Driving CTS Pin) Rising or Falling Edge Event Interrupt Disable**

37.7.9 USART Interrupt Mask Register

Name: US_IMR

Address: 0x400A0010 (0), 0x400A4010 (1)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	MANE
23	22	21	20	19	18	17	16
–	–	–	–	CTSIC	DCDIC	DSRIC	RIIC
15	14	13	12	11	10	9	8
–	–	NACK	RXBUFF	TXBUFE	ITER	TXEMPTY	TIMEOUT
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY

For SPI specific configuration, see Section 37.7.10 ”USART Interrupt Mask Register (SPI_MODE)”.

The following configuration values are valid for all listed bit names of this register:

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 Buffer Interrupt Mask (available in all USART modes of operation)**
- **ENDTX: End of Transmit Buffer Interrupt Mask (available in all USART modes of operation)**
- **OVRE: Overrun Error Interrupt Mask**
- **FRAME: Framing Error Interrupt Mask**
- **PARE: Parity Error Interrupt Mask**
- **TIMEOUT: Time-out Interrupt Mask**
- **TXEMPTY: TXEMPTY Interrupt Mask**
- **ITER: Max Number of Repetitions Reached Interrupt Mask**
- **TXBUFE: Transmit Buffer Empty Interrupt Mask (available in all USART modes of operation)**
- **RXBUFF: Receive Buffer Full Interrupt Mask (available in all USART modes of operation)**
- **NACK: Non Acknowledge Interrupt Mask**
- **RIIC: Ring Indicator Input Change Mask**

- **DSRIC: Data Set Ready Input Change Mask**
- **DCDIC: Data Carrier Detect Input Change Interrupt Mask**
- **CTSIC: Clear to Send Input Change Interrupt Mask**
- **MANE: Manchester Error Interrupt Mask**

37.7.10 USART Interrupt Mask Register (SPI_MODE)

Name: US_IMR (SPI_MODE)

Address: 0x400A0010 (0), 0x400A4010 (1)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	NSSE	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	UNRE	TXEMPTY	–
7	6	5	4	3	2	1	0
–	–	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

This configuration is relevant only if USART_MODE = 0xE or 0xF in the USART Mode Register.

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

- **RXRDY: RXRDY Interrupt Mask**
- **TXRDY: TXRDY Interrupt Mask**
- **ENDRX: End of Receive Buffer Interrupt Mask**
- **ENDTX: End of Transmit Buffer Interrupt Mask**
- **OVRE: Overrun Error Interrupt Mask**
- **TXEMPTY: TXEMPTY Interrupt Mask**
- **UNRE: SPI Underrun Error Interrupt Mask**
- **TXBUFE: Transmit Buffer Empty Interrupt Mask**
- **RXBUFF: Receive Buffer Full Interrupt Mask**
- **NSSE: NSS Line (Driving CTS Pin) Rising or Falling Edge Event Interrupt Mask**

37.7.11 USART Channel Status Register

Name: US_CSR

Address: 0x400A0014 (0), 0x400A4014 (1)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	MANERR
23	22	21	20	19	18	17	16
CTS	DCD	DSR	RI	CTSIC	DCDIC	DSRIC	RIIC
15	14	13	12	11	10	9	8
–	–	NACK	RXBUFF	TXBUFE	ITER	TXEMPTY	TIMEOUT
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY

For SPI specific configuration, see Section 37.7.12 ”USART Channel Status Register (SPI_MODE)”.

- **RXRDY: Receiver Ready (cleared by reading US_RHR)**

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 (cleared by writing US_THR)**

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 (cleared by writing a one to bit US_CR.RSTSTA)**

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 RX Buffer (cleared by writing US_RCR or US_RNCR)**

0: The Receive Counter Register has not reached 0 since the last write in US_RCR or US_RNCR(1).

1: The Receive Counter Register has reached 0 since the last write in US_RCR or US_RNCR(1).

- **ENDTX: End of TX Buffer (cleared by writing US_TCR or US_TNCR)**

0: The Transmit Counter Register has not reached 0 since the last write in US_TCR or US_TNCR(1).

1: The Transmit Counter Register has reached 0 since the last write in US_TCR or US_TNCR(1).

- **OVRE: Overrun Error (cleared by writing a one to bit US_CR.RSTSTA)**

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 (cleared by writing a one to bit US_CR.RSTSTA)**

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 (cleared by writing a one to bit US_CR.RSTSTA)**

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 (cleared by writing a one to bit US_CR.STTTO)**

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 (cleared by writing US_THR)**

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 (cleared by writing a one to bit US_CR.RSTIT)**

0: Maximum number of repetitions has not been reached since the last RSTIT.

1: Maximum number of repetitions has been reached since the last RSTIT.

- **TXBUFE: TX Buffer Empty (cleared by writing US_TCR or US_TNCR)**

0: US_TCR or US_TNCR have a value other than 0[(1)].

1: Both US_TCR and US_TNCR have a value of 0[(1)].

- **RXBUFF: RX Buffer Full (cleared by writing US_RCR or US_RNCR)**

0: US_RCR or US_RNCR have a value other than 0[(1)].

1: Both US_RCR and US_RNCR have a value of 0[(1)].

Note: 1. US_RCR, US_RNCR, US_TCR and US_TNCR are PDC registers.

- **NACK: Non Acknowledge Interrupt (cleared by writing a one to bit US_CR.RSTNACK)**

0: Non acknowledge has not been detected since the last RSTNACK.

1: At least one non acknowledge has been detected since the last RSTNACK.

- **RIIC: Ring Indicator Input Change Flag (cleared on read)**

0: No input change has been detected on the RI pin since the last read of US_CSR.

1: At least one input change has been detected on the RI pin since the last read of US_CSR.

- **DSRIC: Data Set Ready Input Change Flag (cleared on read)**

0: No input change has been detected on the DSR pin since the last read of US_CSR.

1: At least one input change has been detected on the DSR pin since the last read of US_CSR.

- **DCDIC: Data Carrier Detect Input Change Flag (cleared on read)**

0: No input change has been detected on the DCD pin since the last read of US_CSR.

1: At least one input change has been detected on the DCD pin since the last read of US_CSR.

- **CTSIC: Clear to Send Input Change Flag (cleared on read)**

0: No input change has been detected on the CTS pin since the last read of US_CSR.

1: At least one input change has been detected on the CTS pin since the last read of US_CSR.

- **RI: Image of RI Input**

0: RI input is driven low.

1: RI input is driven high.

- **DSR: Image of DSR Input**

0: DSR input is driven low.

1: DSR input is driven high.

- **DCD: Image of DCD Input**

0: DCD input is driven low.

1: DCD input is driven high.

- **CTS: Image of CTS Input**

0: CTS input is driven low.

1: CTS input is driven high.

- **MANERR: Manchester Error (cleared by writing a one to the bit US_CR.RSTSTA)**

0: No Manchester error has been detected since the last RSTSTA.

1: At least one Manchester error has been detected since the last RSTSTA.

37.7.12 USART Channel Status Register (SPI_MODE)

Name: US_CSR (SPI_MODE)

Address: 0x400A0014 (0), 0x400A4014 (1)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
NSS	–	–	–	NSSE	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	UNRE	TXEMPTY	–
7	6	5	4	3	2	1	0
–	–	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

This configuration is relevant only if USART_MODE = 0xE or 0xF in the USART Mode Register.

- **RXRDY: Receiver Ready (cleared by reading US_RHR)**

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 (cleared by writing US_THR)**

0: A character is in the US_THR waiting to be transferred to the Transmit Shift Register or the transmitter is disabled. As soon as the transmitter is enabled, TXRDY becomes 1.

1: There is no character in the US_THR.

- **ENDRX: End of RX Buffer (cleared by writing US_RCR or US_RNCR)**

0: The Receive Counter Register has not reached 0 since the last write in US_RCR or US_RNCR(1).

1: The Receive Counter Register has reached 0 since the last write in US_RCR or US_RNCR(1).

- **ENDTX: End of TX Buffer (cleared by writing US_TCR or US_TNCR)**

0: The Transmit Counter Register has not reached 0 since the last write in US_TCR or US_TNCR(1).

1: The Transmit Counter Register has reached 0 since the last write in US_TCR or US_TNCR(1).

- **OVRE: Overrun Error (cleared by writing a one to bit US_CR.RSTSTA)**

0: No overrun error has occurred since the last RSTSTA.

1: At least one overrun error has occurred since the last RSTSTA.

- **TXEMPTY: Transmitter Empty (cleared by writing US_THR)**

0: There are characters in either US_THR or the Transmit Shift Register, or the transmitter is disabled.

1: There are no characters in US_THR, nor in the Transmit Shift Register.

- **UNRE: Underrun Error (cleared by writing a one to bit US_CR.RSTSTA)**

0: No SPI underrun error has occurred since the last RSTSTA.

1: At least one SPI underrun error has occurred since the last RSTSTA.

- **TXBUFE: TX Buffer Empty (cleared by writing US_TCR or US_TNCR)**

0: US_TCR or US_TNCR have a value other than 0[(1)].

1: Both US_TCR and US_TNCR have a value of 0[(1)].

- **RXBUFF: RX Buffer Full (cleared by writing US_RCR or US_RNCR)**

0: US_RCR or US_RNCR have a value other than 0[(1)].

1: Both US_RCR and US_RNCR have a value of 0[(1)].

Note: 1. US_RCR, US_RNCR, US_TCR and US_TNCR are PDC registers.

- **NSSE: NSS Line (Driving CTS Pin) Rising or Falling Edge Event (cleared on read)**

0: No NSS line event has been detected since the last read of US_CSR.

1: A rising or falling edge event has been detected on NSS line since the last read of US_CSR .

- **NSS: Image of NSS Line**

0: NSS line is driven low (if NSSE = 1, falling edge occurred on NSS line).

1: NSS line is driven high (if NSSE = 1, rising edge occurred on NSS line).

37.7.13 USART Receive Holding Register

Name: US_RHR

Address: 0x400A0018 (0), 0x400A4018 (1)

Access: Read-only

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

- **RXCHR: Received Character**

Last character received if RXRDY is set.

- **RXSYNH: Received Sync**

0: Last character received is a data.

1: Last character received is a command.

37.7.14 USART Transmit Holding Register

Name: US_THR

Address: 0x400A001C (0), 0x400A401C (1)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
TXSYNH	–	–	–	–	–	–	TXCHR

7	6	5	4	3	2	1	0
TXCHR							

- **TXCHR: Character to be Transmitted**

Next character to be transmitted after the current character if TXRDY is not set.

- **TXSYNH: Sync Field to be Transmitted**

0: The next character sent is encoded as a data. Start frame delimiter is DATA SYNC.

1: The next character sent is encoded as a command. Start frame delimiter is COMMAND SYNC.

37.7.15 USART Baud Rate Generator Register

Name: US_BRGR

Address: 0x400A0020 (0), 0x400A4020 (1)

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the USART Write Protection Mode Register.

- **CD: Clock Divider**

	USART_MODE ≠ ISO7816			
	SYNC = 0		**SYNC = 1 or USART_MODE = SPI (Master or Slave)**	
CD	**OVER = 0**	**OVER = 1**		**USART_MODE = ISO7816**
0	Baud Rate Clock Disabled			
1 to 65535	CD = Selected Clock / (16 × Baud Rate)	CD = Selected Clock / (8 × Baud Rate)	CD = Selected Clock / Baud Rate	CD = Selected Clock / (FI_DI_RATIO × Baud Rate)

- **FP: Fractional Part**

0: Fractional divider is disabled.

1–7: Baud rate resolution, defined by FP × 1/8.

Warning: When the value of field FP is greater than 0, the SCK (oversampling clock) generates non-constant duty cycles. The SCK high duration is increased by “selected clock” period from time to time. The duty cycle depends on the value of the CD field.

37.7.16 USART Receiver Time-out Register

Name: US_RTOR

Address: 0x400A0024 (0), 0x400A4024 (1)

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
TO							

7	6	5	4	3	2	1	0
TO							

This register can only be written if the WPEN bit is cleared in the USART Write Protection Mode Register.

- **TO: Time-out Value**

0: The receiver time-out is disabled.

1–65535: The receiver time-out is enabled and TO is Time-out Delay / Bit Period.

37.7.17 USART Transmitter Timeguard Register

Name: US_TTGR

Address: 0x400A0028 (0), 0x400A4028 (1)

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the USART Write Protection Mode Register.

- **TG: Timeguard Value**

0: The transmitter timeguard is disabled.

1–255: The transmitter timeguard is enabled and TG is Timeguard Delay / Bit Period.

37.7.18 USART FI DI RATIO Register

Name: US_FIDI

Address: 0x400A0040 (0), 0x400A4040 (1)

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	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 the USART Write Protection Mode Register.

- **FI_DI_RATIO: FI Over DI Ratio Value**

0: If ISO7816 mode is selected, the baud rate generator generates no signal.

1–2: Do not use.

3–2047: If ISO7816 mode is selected, the baud rate is the clock provided on SCK divided by FI_DI_RATIO.

37.7.19 USART Number of Errors Register

Name: US_NER

Address: 0x400A0044 (0), 0x400A4044 (1)

Access: Read-only

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

This register is relevant only if USART_MODE = 0x4 or 0x6 in the USART Mode Register.

- **NB_ERRORS: Number of Errors**

Total number of errors that occurred during an ISO7816 transfer. This register automatically clears when read.

37.7.20 USART IrDA Filter Register

Name: US_IF

Address: 0x400A004C (0), 0x400A404C (1)

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
IRDA_FILTER							

This register is relevant only if USART_MODE = 0x8 in the USART Mode Register.
This register can only be written if the WPEN bit is cleared in the USART Write Protection Mode Register.

- **IRDA_FILTER: IrDA Filter**

The IRDA_FILTER value must be defined to meet the following criteria:

$t_{peripheral\ clock} \times (IRDA\_FILTER + 3) < 1.41\ \mu s$

37.7.21 USART Manchester Configuration Register

Name: US_MAN

Address: 0x400A0050 (0), 0x400A4050 (1)

Access: Read/Write

31	30	29	28	27	26	25	24
–	DRIFT	ONE	RX_MPOL	–	–	RX_PP	

23	22	21	20	19	18	17	16
–	–	–	–	RX_PL			

15	14	13	12	11	10	9	8
–	–	–	TX_MPOL	–	–	TX_PP	

7	6	5	4	3	2	1	0
–	–	–	–	TX_PL			

This register can only be written if the WPEN bit is cleared in the USART Write Protection Mode Register.

- **TX_PL: Transmitter Preamble Length**

0: The transmitter preamble pattern generation is disabled

1–15: The preamble length is TX_PL × Bit Period

- **TX_PP: Transmitter Preamble Pattern**

The following values assume that TX_MPOL field is not set:

Value	Name	Description
0	ALL_ONE	The preamble is composed of ‘1’s
1	ALL_ZERO	The preamble is composed of ‘0’s
2	ZERO_ONE	The preamble is composed of ‘01’s
3	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 × Bit Period

- **RX_PP: Receiver Preamble Pattern detected**

The following values assume that RX_MPOL field is not set:

Value	Name	Description
00	ALL_ONE	The preamble is composed of ‘1’s
01	ALL_ZERO	The preamble is composed of ‘0’s
10	ZERO_ONE	The preamble is composed of ‘01’s
11	ONE_ZERO	The preamble is composed of ‘10’s

- **RX_MPOL: Receiver Manchester Polarity**

0: Logic zero is coded as a zero-to-one transition, Logic one is coded as a one-to-zero transition.

1: Logic zero is coded as a one-to-zero transition, Logic one is coded as a zero-to-one transition.

- **ONE: Must Be Set to 1**

Bit 29 must always be set to 1 when programming the US_MAN register.

- **DRIFT: Drift Compensation**

0: The USART cannot recover from an important clock drift

1: The USART can recover from clock drift. The 16X clock mode must be enabled.

37.7.22 USART Write Protection Mode Register

Name: US_WPMR

Address: 0x400A00E4 (0), 0x400A40E4 (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: Disables the write protection if WPKEY corresponds to 0x555341 (“USA” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x555341 (“USA” in ASCII).

See Section 37.6.10 ”Register Write Protection” for the list of registers that can be write-protected.

- **WPKEY: Write Protection Key**

Value	Name	Description
0x555341	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

37.7.23 USART Write Protection Status Register

Name: US_WPSR

Address: 0x400A00E8 (0), 0x400A40E8 (1)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
WPVSRC							

15	14	13	12	11	10	9	8
WPVSRC							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WPVS

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of the US_WPSR.

1: A write protection violation has occurred since the last read of the US_WPSR. 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**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

38. Timer Counter (TC)

38.1 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

38.2 Embedded Characteristics

- Total number of TC channels implemented on this device: nine
- 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
  - Trigger/capture events can be directly synchronized by PWM signals
- Internal interrupt signal
- Read of the Capture registers by the PDC
- Compare event fault generation for PWM
- Register Write Protection

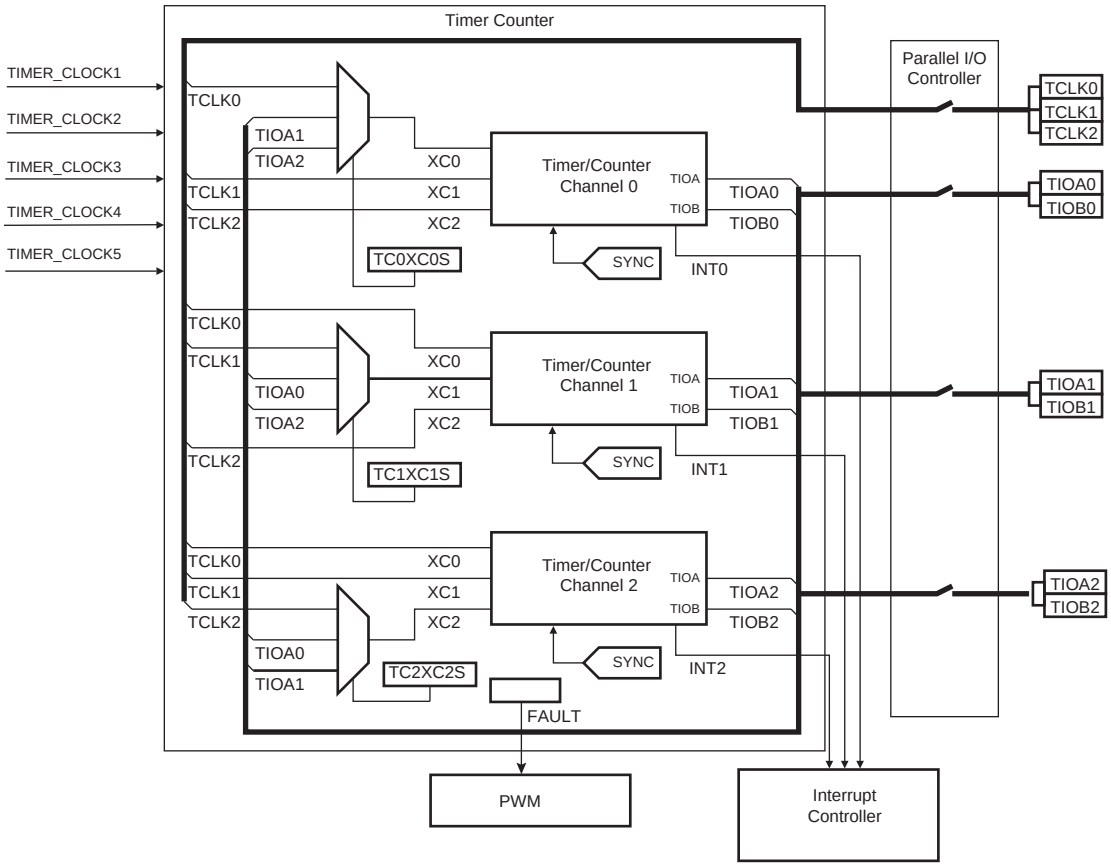
38.3 Block Diagram

Table 38-1. Timer Counter Clock Assignment

Name	Definition
TIMER_CLOCK1	MCK/2
TIMER_CLOCK2	MCK/8
TIMER_CLOCK3	MCK/32
TIMER_CLOCK4	MCK/128
TIMER_CLOCK5(1)	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 38-1. Timer Counter Block Diagram



Note: The QDEC connections are detailed in Figure 38-17.

Table 38-2. Channel Signal Description

Signal Name	Description
XC0, XC1, XC2	External Clock Inputs
TIOAx	Capture Mode: Timer Counter Input Waveform Mode: Timer Counter Output

Table 38-2. Channel Signal Description (Continued)

TIOBx	Capture Mode: Timer Counter Input Waveform Mode: Timer Counter Input/Output
INT	Interrupt Signal Output (internal signal)
SYNC	Synchronization Input Signal (from configuration register)

38.4 Pin List

Table 38-3. Pin List

Pin Name	Description	Type
TCLK0–TCLK2	External Clock Input	Input
TIOA0–TIOA2	I/O Line A	I/O
TIOB0–TIOB2	I/O Line B	I/O

38.5 Product Dependencies

38.5.1 I/O Lines

The pins used for interfacing the compliant external devices may be multiplexed with PIO lines. The programmer must first program the PIO controllers to assign the TC pins to their peripheral functions.

Table 38-4. I/O Lines

Instance	Signal	I/O Line	Peripheral
TC0	TCLK0	PA4	B
TC0	TCLK1	PA28	B
TC0	TCLK2	PA29	B
TC0	TIOA0	PA0	B
TC0	TIOA1	PA15	B
TC0	TIOA2	PA26	B
TC0	TIOB0	PA1	B
TC0	TIOB1	PA16	B
TC0	TIOB2	PA27	B
TC1	TCLK3	PC25	B
TC1	TCLK4	PC28	B
TC1	TCLK5	PC31	B
TC1	TIOA3	PC23	B
TC1	TIOA4	PC26	B
TC1	TIOA5	PC29	B
TC1	TIOB3	PC24	B
TC1	TIOB4	PC27	B
TC1	TIOB5	PC30	B
TC2	TCLK6	PC7	B
TC2	TCLK7	PC10	B

Table 38-4. I/O Lines

TC2	TCLK8	PC14	B
TC2	TIOA6	PC5	B
TC2	TIOA7	PC8	B
TC2	TIOA8	PC11	B
TC2	TIOB6	PC6	B
TC2	TIOB7	PC9	B
TC2	TIOB8	PC12	B

38.5.2 Power Management

The TC is clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the Timer Counter clock of each channel.

38.5.3 Interrupt Sources

The TC has an interrupt line per channel connected to the interrupt controller. Handling the TC interrupt requires programming the interrupt controller before configuring the TC.

Table 38-5. Peripheral IDs

Instance	ID
TC0	21
TC1	22
TC2	23

38.5.4 Synchronization Inputs from PWM

The TC has trigger/capture inputs internally connected to the PWM. Refer to Section 38.6.14 “Synchronization with PWM” and to the implementation of the Pulse Width Modulation (PWM) in this product.

38.5.5 Fault Output

The TC has the FAULT output internally connected to the fault input of PWM. Refer to Section 38.6.18 “Fault Mode” and to the implementation of the Pulse Width Modulation (PWM) in this product.

38.6 Functional Description

38.6.1 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 38-6 “Register Mapping”.

38.6.2 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.

38.6.3 Clock Selection

At block level, input clock signals of each channel can be connected either to the external inputs TCLKx, or to the internal I/O signals TIOAx for chaining[1] by programming the TC Block Mode Register (TC_BMR). See Figure 38-2.

Each channel can independently select an internal or external clock source for its counter[2]:

- External clock signals: 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 38-3.

Notes:
1. In Waveform mode, to chain two timers, it is mandatory to initialize some parameters:
   - Configure TIOx outputs to 1 or 0 by writing the required value to TC_CMR.ASWTRG.
   - Bit TC_BCR.SYNC must be written to 1 to start the channels at the same time.
2. In all cases, if an external clock is used, the duration of each of its levels must be longer than the peripheral clock period, so the clock frequency will be at least 2.5 times lower than the peripheral clock.

Figure 38-2. Clock Chaining Selection

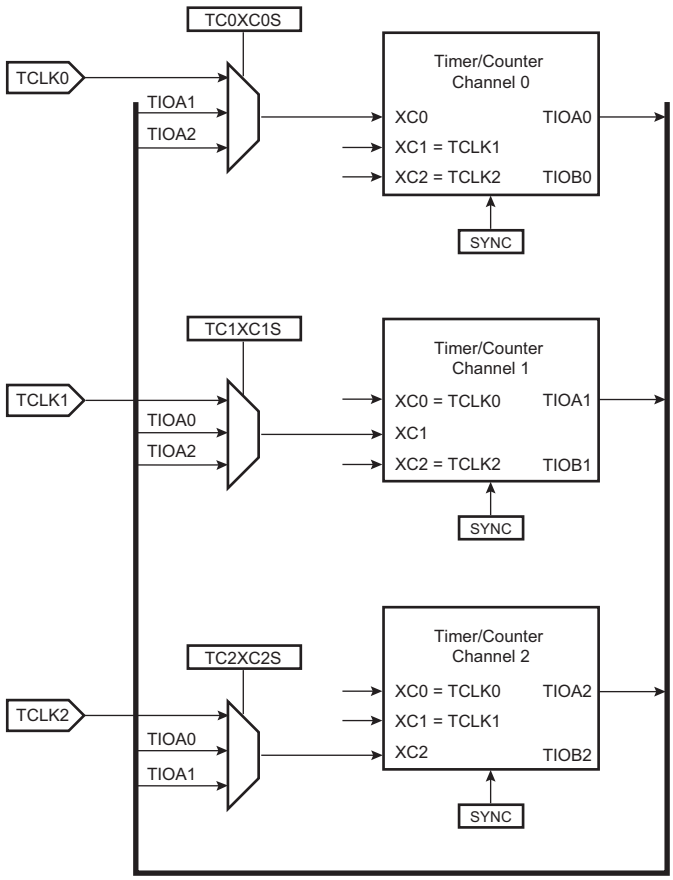
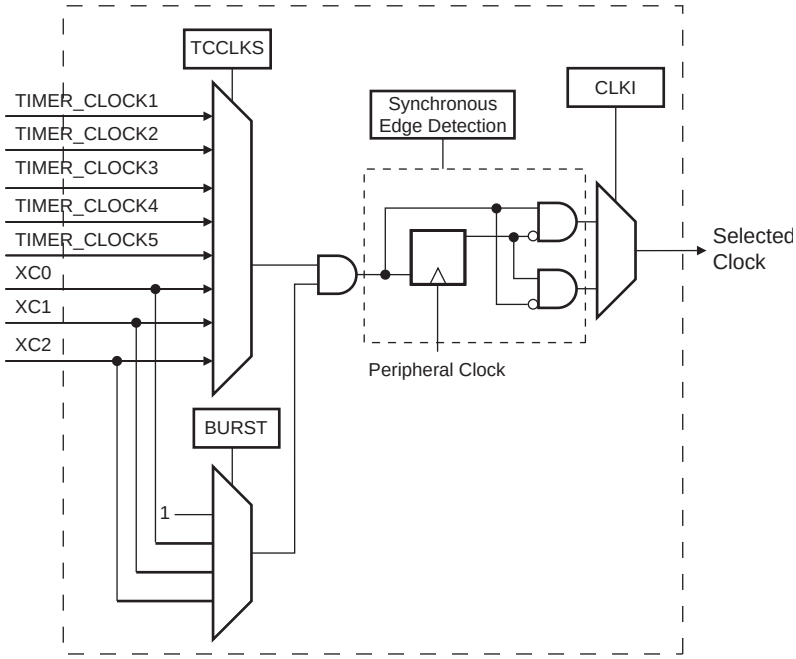


Figure 38-3. Clock Selection

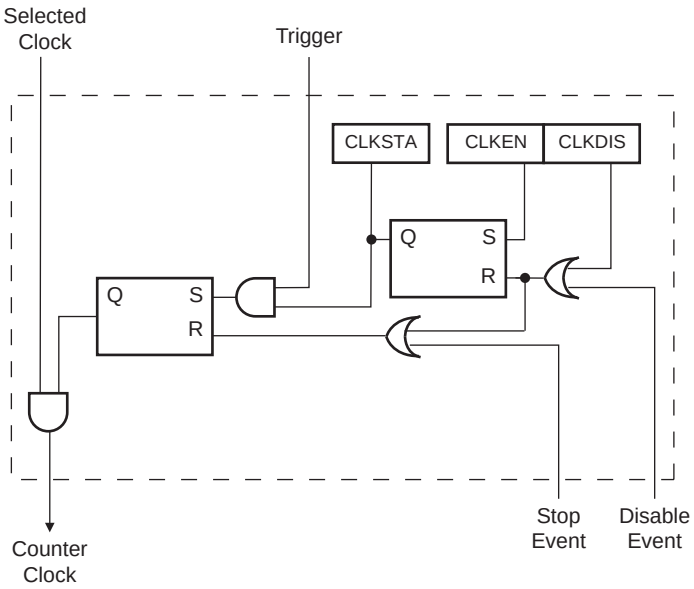


38.6.4 Clock Control

The clock of each counter can be controlled in two different ways: it can be enabled/disabled and started/stopped. See Figure 38-4.

- The clock can be enabled or disabled by the user with the CLKEN and the CLKDIS commands in the 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 38-4. Clock Control



38.6.5 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, TIOAx and TIOBx are configured as inputs.

In Waveform mode, TIOAx is always configured to be an output and TIOBx is an output if it is not selected to be the external trigger.

38.6.6 Trigger

A trigger resets the counter and starts the counter clock. Three types of triggers are common to both modes, and a fourth external trigger is available to each mode.

Regardless of the trigger used, it will be taken into account at the following active edge of the selected clock. This means that the counter value can be read differently from zero just after a trigger, especially when a low frequency signal is selected as the clock.

The following triggers are common to both modes:

- Software Trigger: Each channel has a software trigger, available by setting SWTRG in TC_CCR.
- SYNC: Each channel has a synchronization signal SYNC. When asserted, this signal has the same effect as a software trigger. The SYNC signals of all channels are asserted simultaneously by writing TC_BCR (Block Control) with SYNC set.
- Compare RC Trigger: RC is implemented in each channel and can provide a trigger when the counter value matches the RC value if CPCTRG is set in 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 TIOAx and TIOBx. In Waveform mode, an external event can be programmed on one of the following signals: TIOBx, 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.

38.6.7 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 TIOAx and TIOBx signals which are considered as inputs.

Figure 38-6 shows the configuration of the TC channel when programmed in Capture mode.

38.6.8 Capture Registers A and B

Registers A and B (RA and RB) are used as capture registers. They can be loaded with the counter value when a programmable event occurs on the signal TIOAx.

The LDRA field in the TC_CMR defines the TIOAx selected edge for the loading of register A, and the LDRB field defines the TIOAx selected edge for the loading of Register B.

The subsampling ratio defined by the SBSMPLR field in TC_CMR is applied to these selected edges, so that the loading of Register A and Register B occurs once every 1, 2, 4, 8 or 16 selected edges.

RA is loaded only if it has not been loaded since the last trigger or if RB has been loaded since the last loading of RA.

RB is loaded only if RA has been loaded since the last trigger or the last loading of RB.

Loading RA or RB before the read of the last value loaded sets the Overrun Error Flag (LOVRS bit) in the TC_SR. In this case, the old value is overwritten.

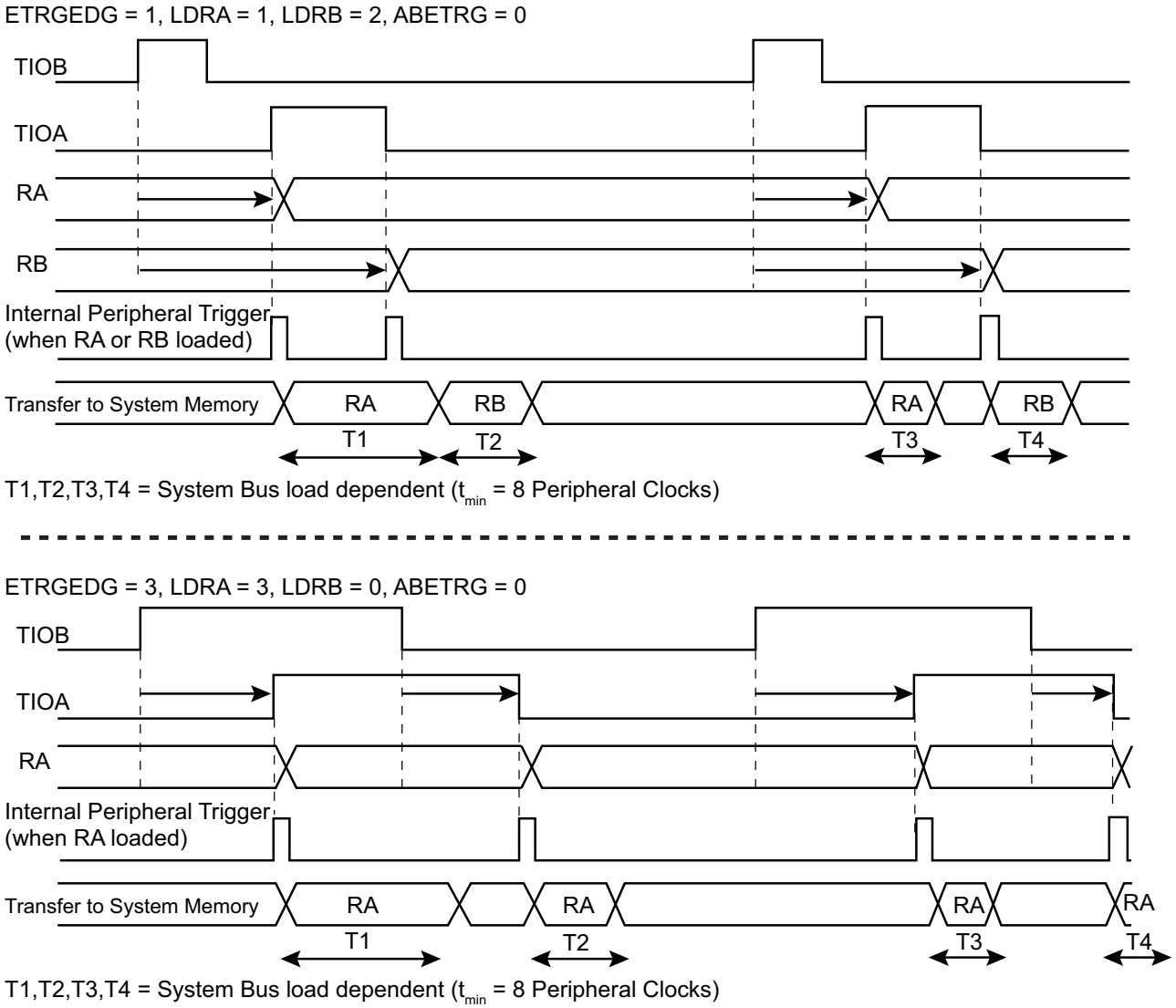
When DMA is used, the RAB register address must be configured as source address of the transfer. The RAB register provides the next unread value from Register A and Register B. It may be read by the DMA after a request has been triggered upon loading Register A or Register B.

38.6.9 Transfer with PDC in Capture Mode

The PDC can perform access from the TC to system memory in Capture mode only.

Figure 38-5 illustrates how TC_RA and TC_RB can be loaded in the system memory without CPU intervention.

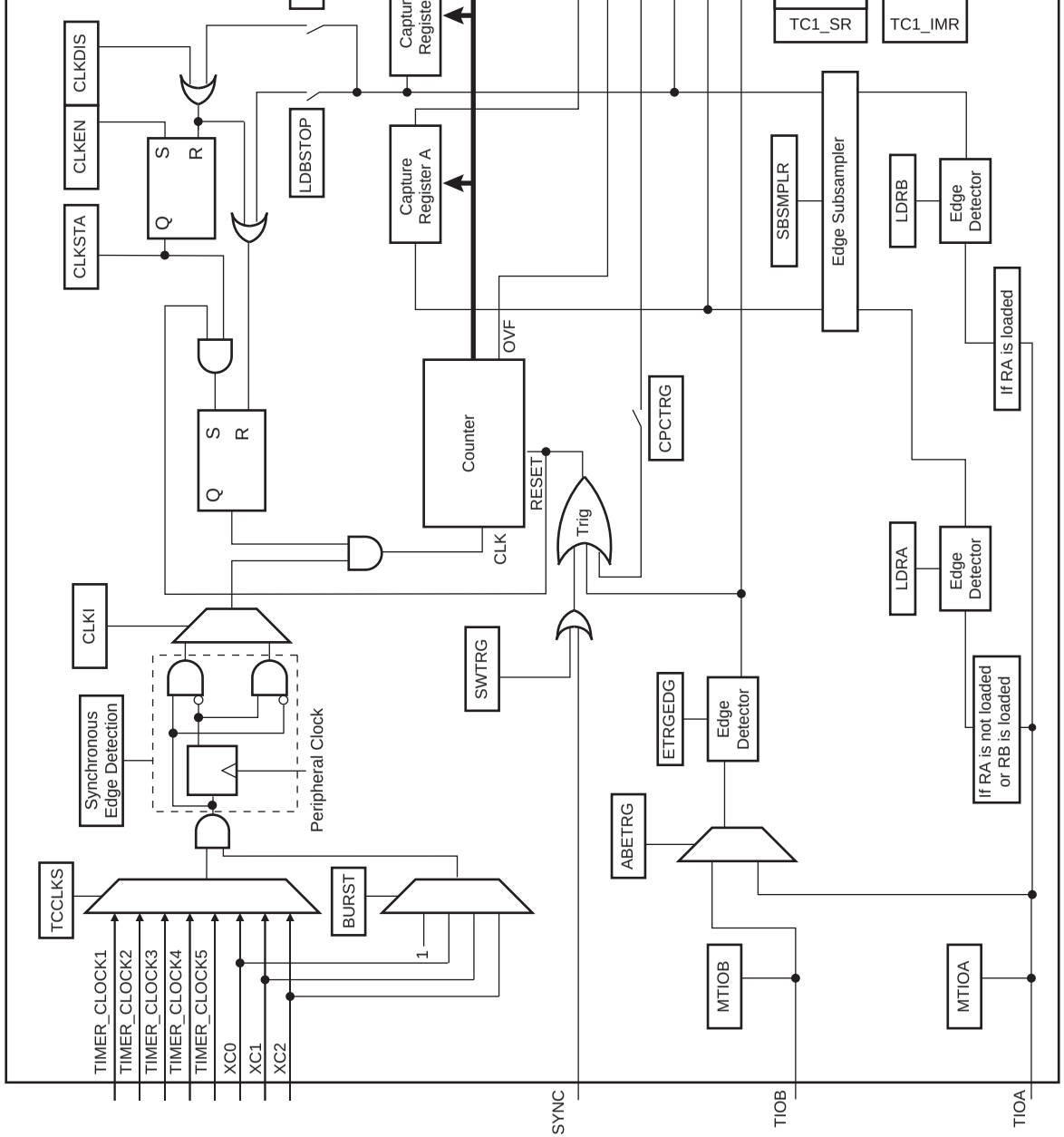
Figure 38-5. Example of Transfer with PDC in Capture Mode



38.6.10 Trigger Conditions

In addition to the SYNC signal, the software trigger and the RC compare trigger, an external trigger can be defined.

The ABETRG bit in the TC_CMR selects TIOAx or TIOBx input signal as an external trigger or the trigger signal from the output comparator of the PWM module. 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.


TC1_SR
TC1_IMR
CLKDIS
CLKEN
CLKSTA
S
R
Q
LDBSTOP
Capture Register A
SBSMPLR
Edge Subsampler
LDRB
Edge Detector
If RA is loaded
OVF
Counter
CPCTRG
RESET
Trig
CLK
CLKI
SWTRG
LDRA
Edge Detector
If RA is not loaded or RB is loaded
Synchronous Edge Detection
Peripheral Clock
ETRGEDG
Edge Detector
ABETRG
TCCLKS
BURST
1
MTIOB
MTIOA
TIMER_CLOCK1
TIMER_CLOCK2
TIMER_CLOCK3
TIMER_CLOCK4
TIMER_CLOCK5
XC0
XC1
XC2
SYNC
TIOB
TIOA

38.6.11 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, TIOAx is configured as an output and TIOBx is defined as an output if it is not used as an external event (EEVT parameter in TC_CMR).

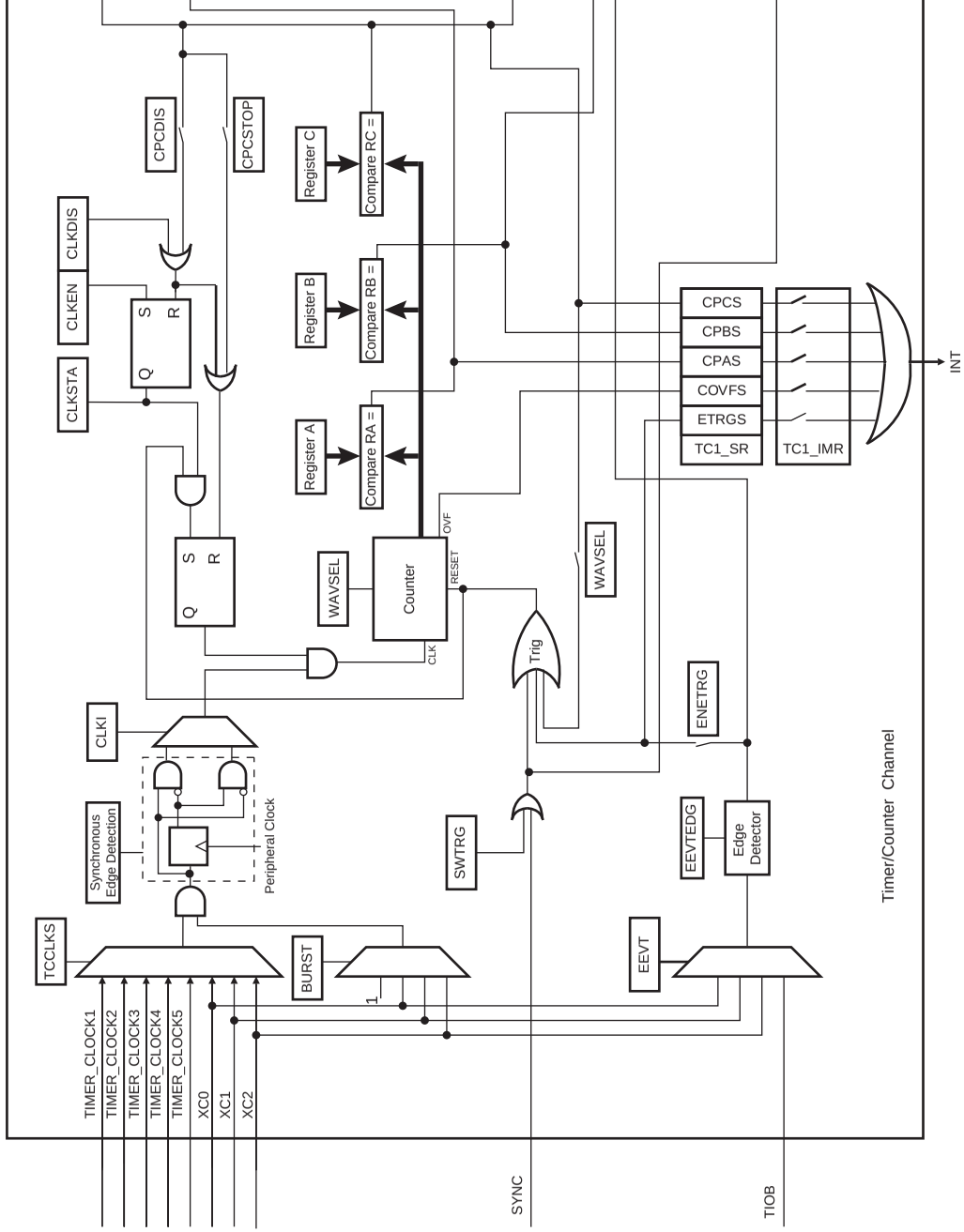
Figure 38-7 shows the configuration of the TC channel when programmed in Waveform operating mode.

38.6.12 Waveform Selection

Depending on the WAVSEL parameter in TC_CMR, 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 TIOAx output, RB Compare is used to control the TIOBx output (if correctly configured) and RC Compare is used to control TIOAx and/or TIOBx outputs.


Timer/Counter Channel
TIMER_CLOCK1
TIMER_CLOCK2
TIMER_CLOCK3
TIMER_CLOCK4
TIMER_CLOCK5
XC0
XC1
XC2
TCCLKS
BURST
1
CLKI
Synchronous Edge Detection
Peripheral Clock
CLKDIS
CLKEN
CLKSTA
S
R
Q
CPCDIS
CPCSTOP
Register C
Compare RC =
Register B
Compare RB =
Register A
Compare RA =
WAVSEL
Counter
OVF
RESET
CLK
Trig
SWTRG
ENETRG
EEVTEDG
Edge Detector
EEVT
CPCS
CPBS
CPAS
COVFS
ETRGS
TC1_SR
TC1_IMR
INT
SYNC
TIOB

38.6.12.1 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 38-8.

An external event trigger or a software trigger can reset the value of TC_CV. It is important to note that the trigger may occur at any time. See Figure 38-9.

RC Compare cannot be programmed to generate a trigger in this configuration. At the same time, RC Compare can stop the counter clock (CPCSTOP = 1 in TC_CMR) and/or disable the counter clock (CPCDIS = 1 in TC_CMR).

Figure 38-8. WAVSEL = 00 without Trigger

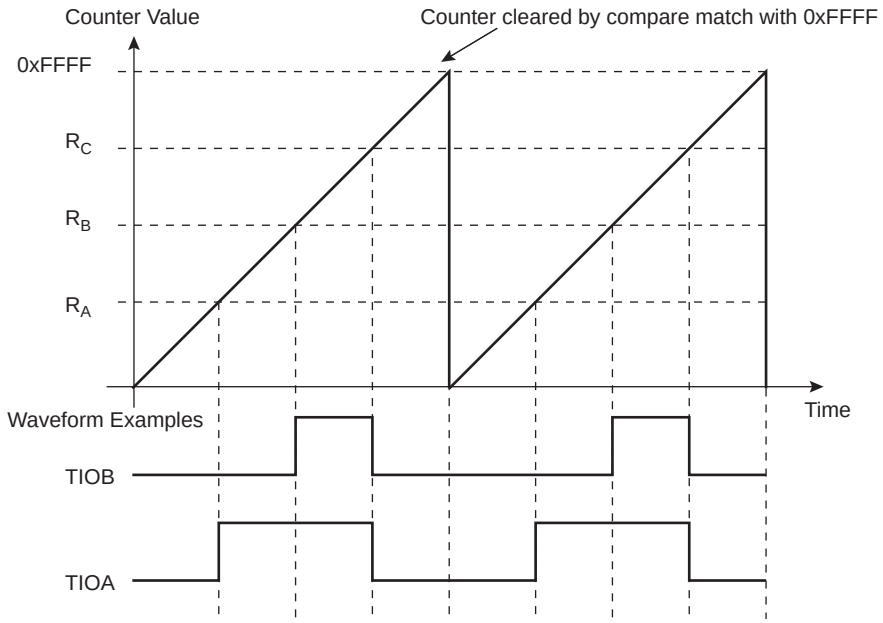
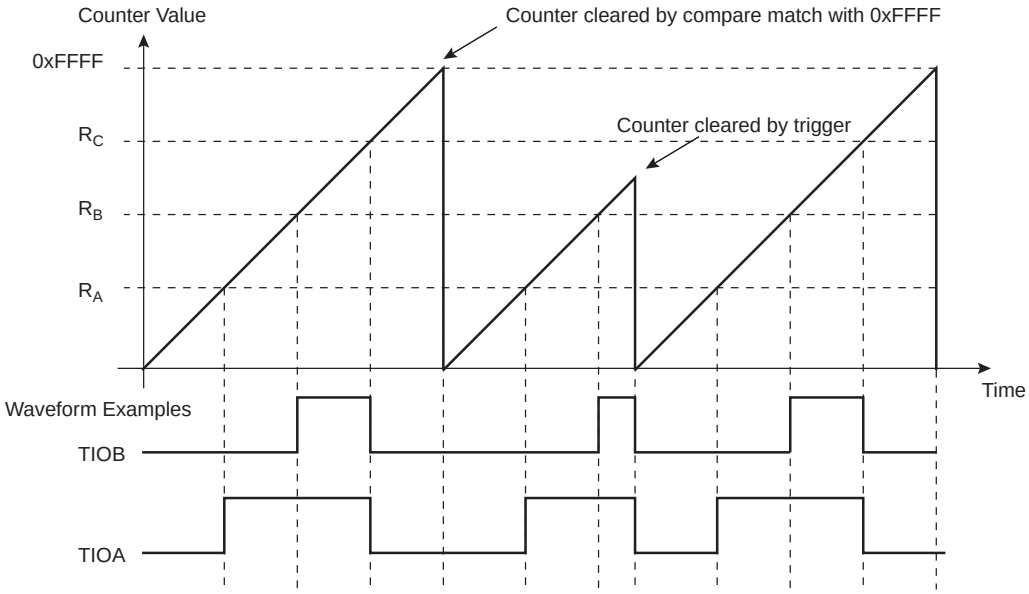


Figure 38-9. WAVSEL = 00 with Trigger



38.6.12.2 WAVSEL = 10

When WAVSEL = 10, the value of TC_CV is incremented from 0 to the value of RC, then automatically reset on a RC Compare. Once the value of TC_CV has been reset, it is then incremented and so on. See Figure 38-10.

It is important to note that TC_CV can be reset at any time by an external event or a software trigger if both are programmed correctly. See Figure 38-11.

In addition, RC Compare can stop the counter clock (CPCSTOP = 1 in TC_CMR) and/or disable the counter clock (CPCDIS = 1 in TC_CMR).

Figure 38-10. WAVSEL = 10 without Trigger

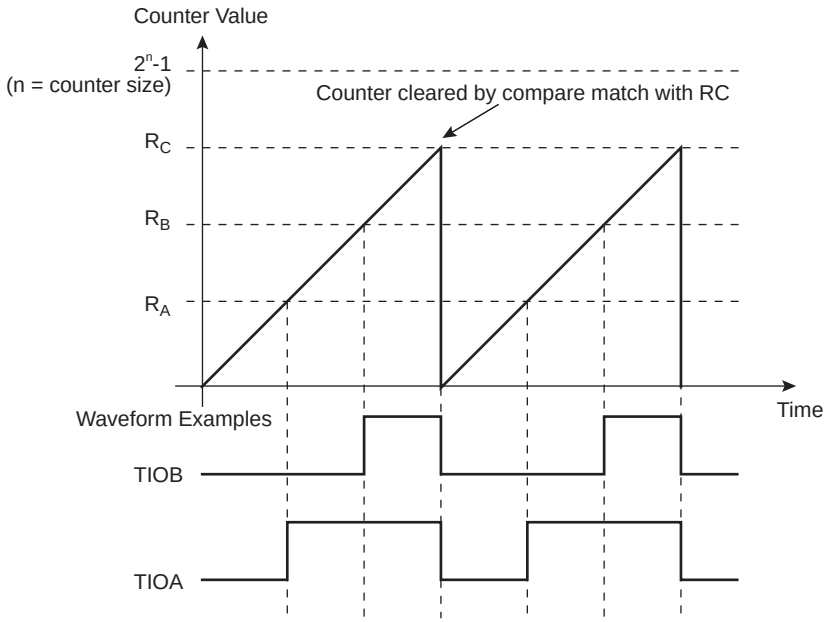
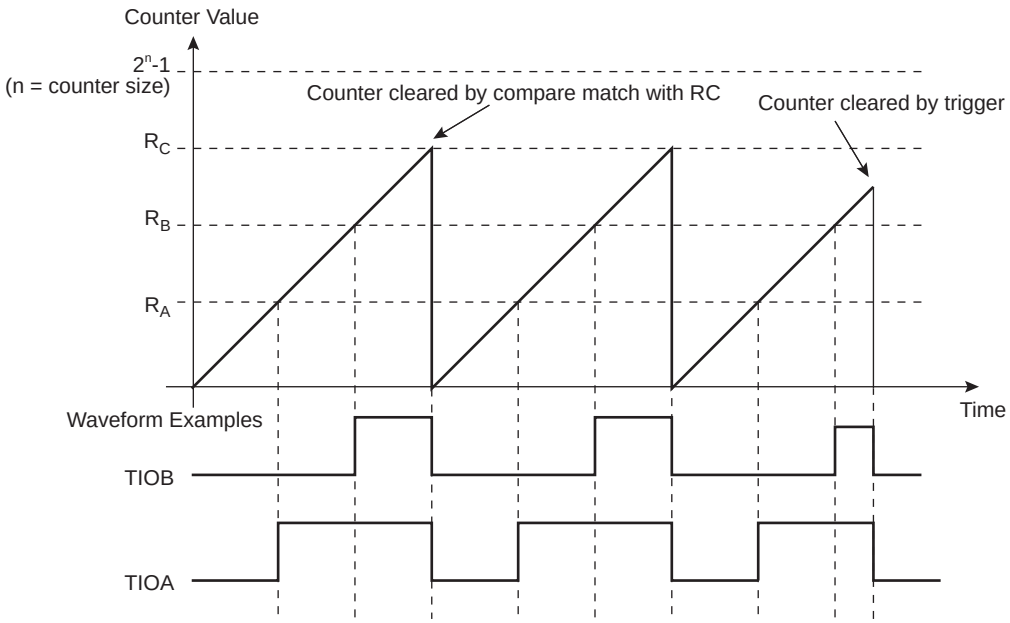


Figure 38-11. WAVSEL = 10 with Trigger



38.6.12.3 WAVSEL = 01

When WAVSEL = 01, the value of TC_CV is incremented from 0 to 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 38-12.

A trigger such as an external event or a software trigger can modify TC_CV at any time. If a trigger occurs while TC_CV is incrementing, TC_CV then decrements. If a trigger is received while TC_CV is decrementing, TC_CV then increments. See Figure 38-13.

RC Compare cannot be programmed to generate a trigger in this configuration.

At the same time, RC Compare can stop the counter clock (CPCSTOP = 1) and/or disable the counter clock (CPCDIS = 1).

Figure 38-12. WAVSEL = 01 without Trigger

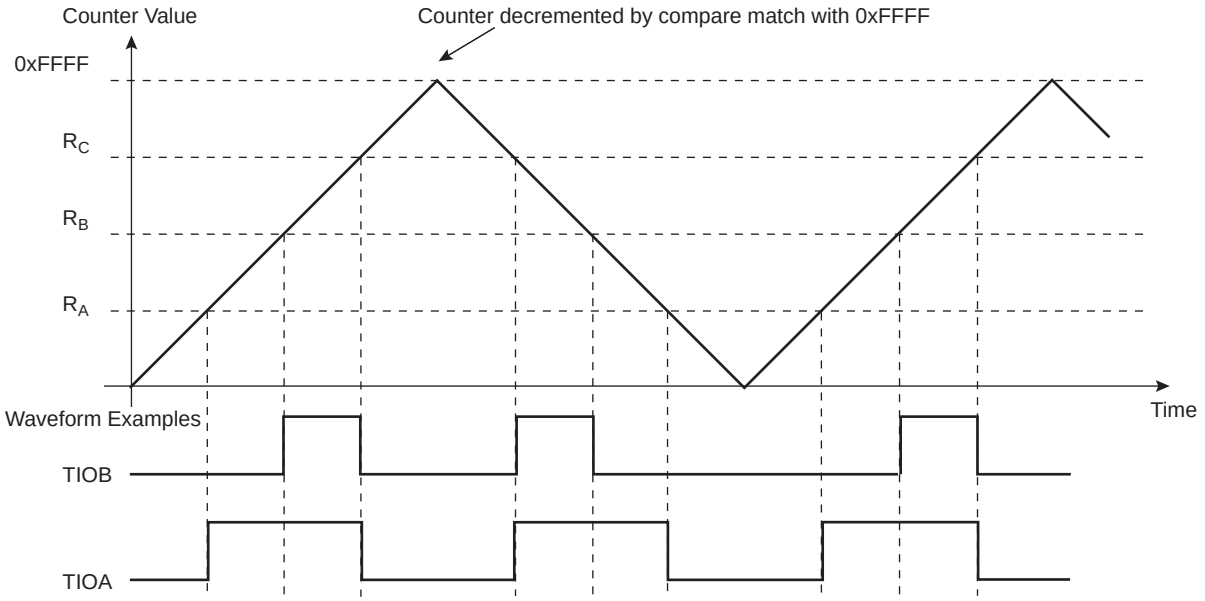
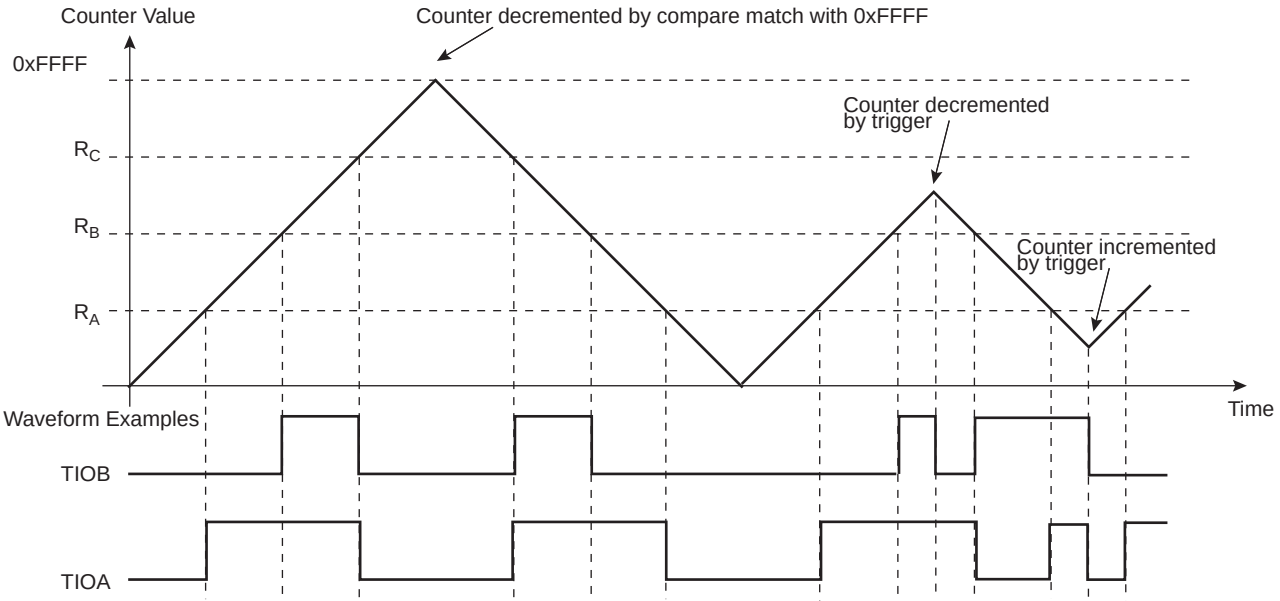


Figure 38-13. WAVSEL = 01 with Trigger



38.6.12.4 WAVSEL = 11

When WAVSEL = 11, the value of TC_CV is incremented from 0 to RC. Once RC is reached, the value of TC_CV is decremented to 0, then re-incremented to RC and so on. See Figure 38-14.

A trigger such as an external event or a software trigger can modify TC_CV at any time. If a trigger occurs while TC_CV is incrementing, TC_CV then decrements. If a trigger is received while TC_CV is decrementing, TC_CV then increments. See Figure 38-15.

RC Compare can stop the counter clock (CPCSTOP = 1) and/or disable the counter clock (CPCDIS = 1).

Figure 38-14. WAVSEL = 11 without Trigger

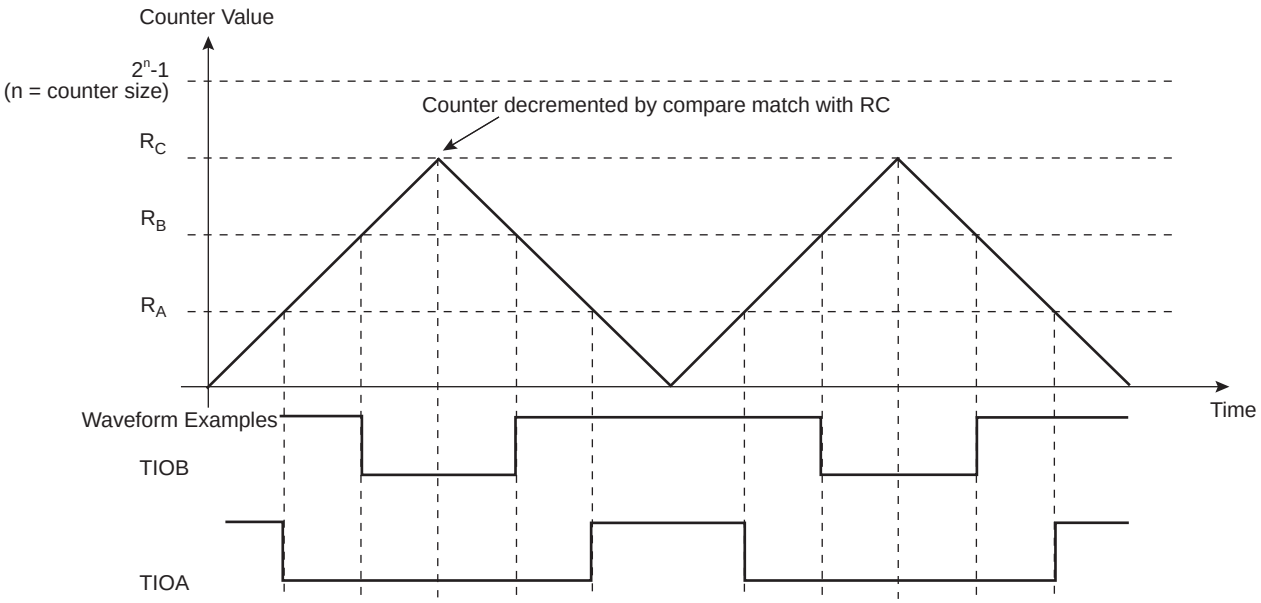
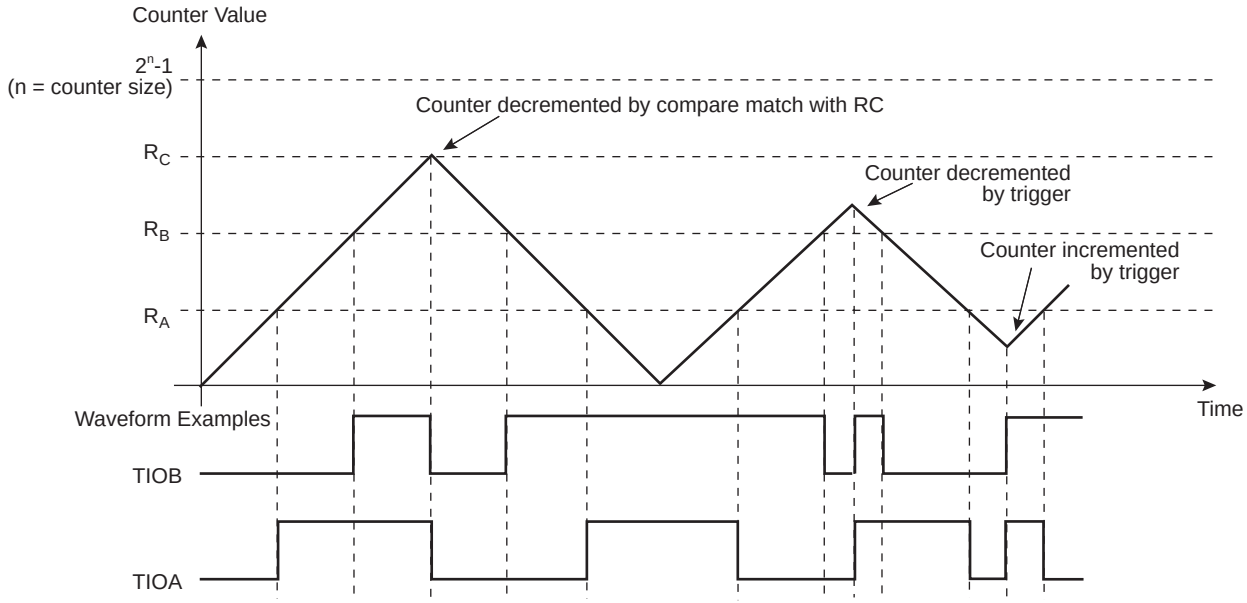


Figure 38-15. WAVSEL = 11 with Trigger



38.6.13 External Event/Trigger Conditions

An external event can be programmed to be detected on one of the clock sources (XC0, XC1, XC2) or TIOBx. 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 TIOBx is defined as an external event signal (EEVT = 0), TIOBx 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 TIOAx.

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.

38.6.14 Synchronization with PWM

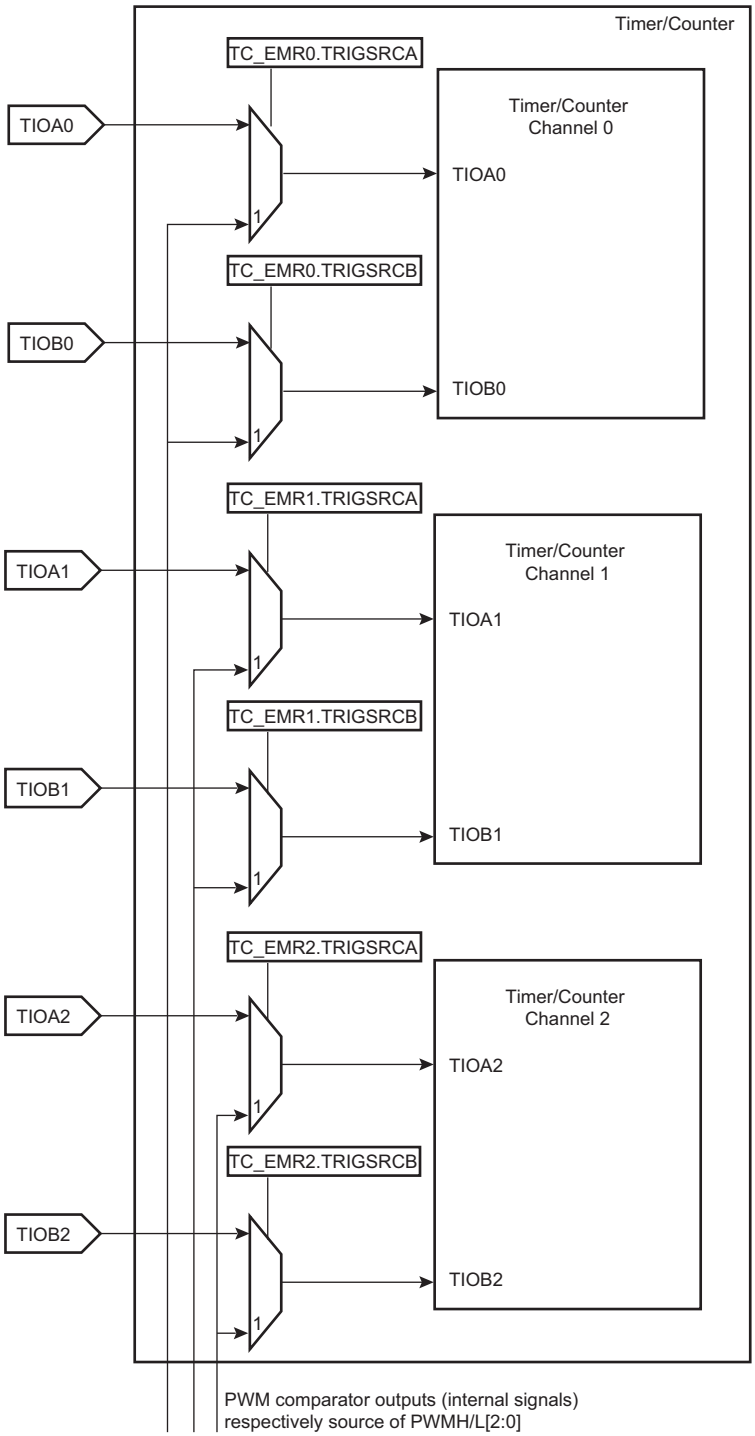
The inputs TIOAx/TIOBx can be bypassed, and thus channel trigger/capture events can be directly driven by the independent PWM module.

PWM comparator outputs (internal signals without dead-time insertion - OCx), respectively source of the PWMH/L[2:0] outputs, are routed to the internal TC inputs. These specific TC inputs are multiplexed with TIOA/B input signal to drive the internal trigger/capture events.

The selection can be programmed in the Extended Mode Register (TC_EMR) fields TRIGSRCA and TRIGSRCB (see Section 38.7.14 “TC Extended Mode Register”).

Each channel of the TC module can be synchronized by a different PWM channel as described in Figure 38-16.

Figure 38-16. Synchronization with PWM



38.6.15 Output Controller

The output controller defines the output level changes on TIOAx and TIOBx following an event. TIOBx Control is used only if TIOBx is defined as output (not as an external event).

The following events control TIOAx and TIOBx:

- Software Trigger
- External Event
- RC Compare

RA Compare controls TIOAx, and RB Compare controls TIOBx. Each of these events can be programmed to set, clear or toggle the output as defined in the corresponding parameter in TC_CMR.

38.6.16 Quadrature Decoder

38.6.16.1 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 38-17).

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.

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.

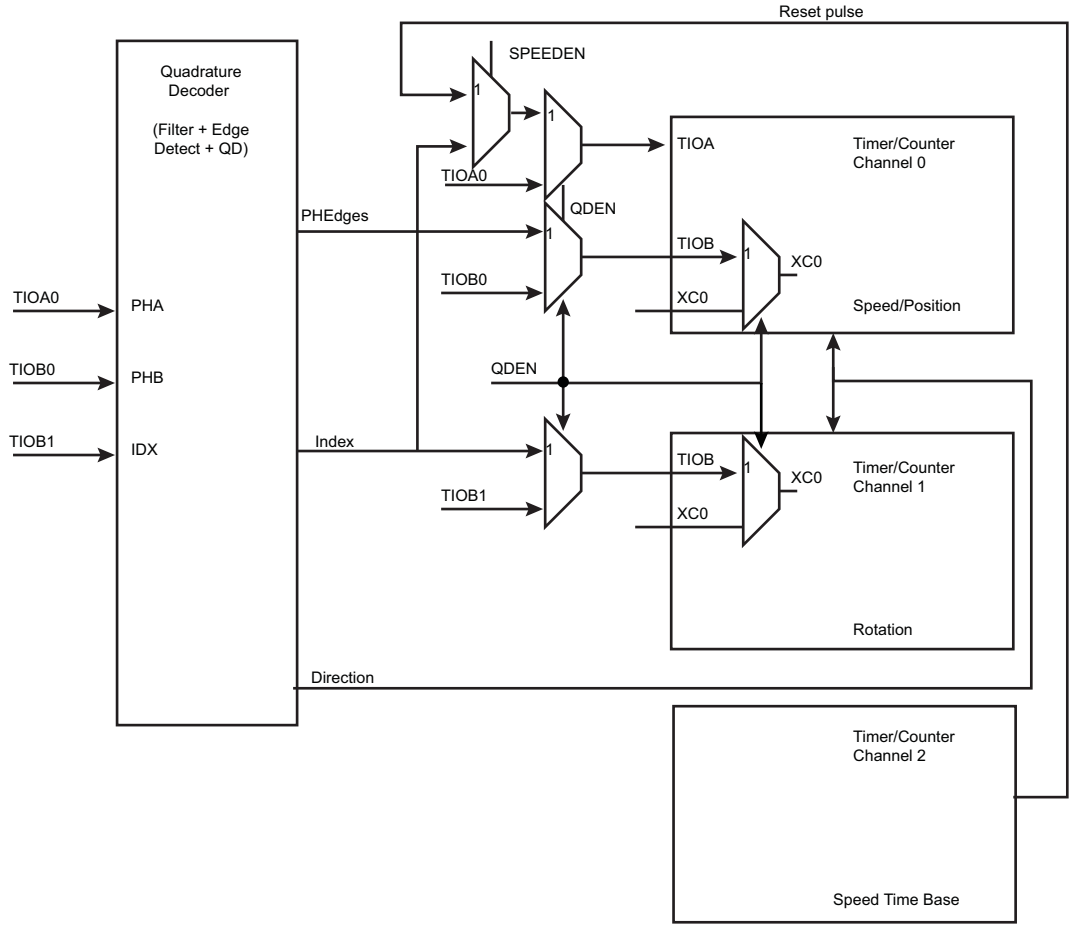
In Speed mode, position cannot be measured but revolution can be measured.

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 38-17. Predefined Connection of the Quadrature Decoder with Timer Counters



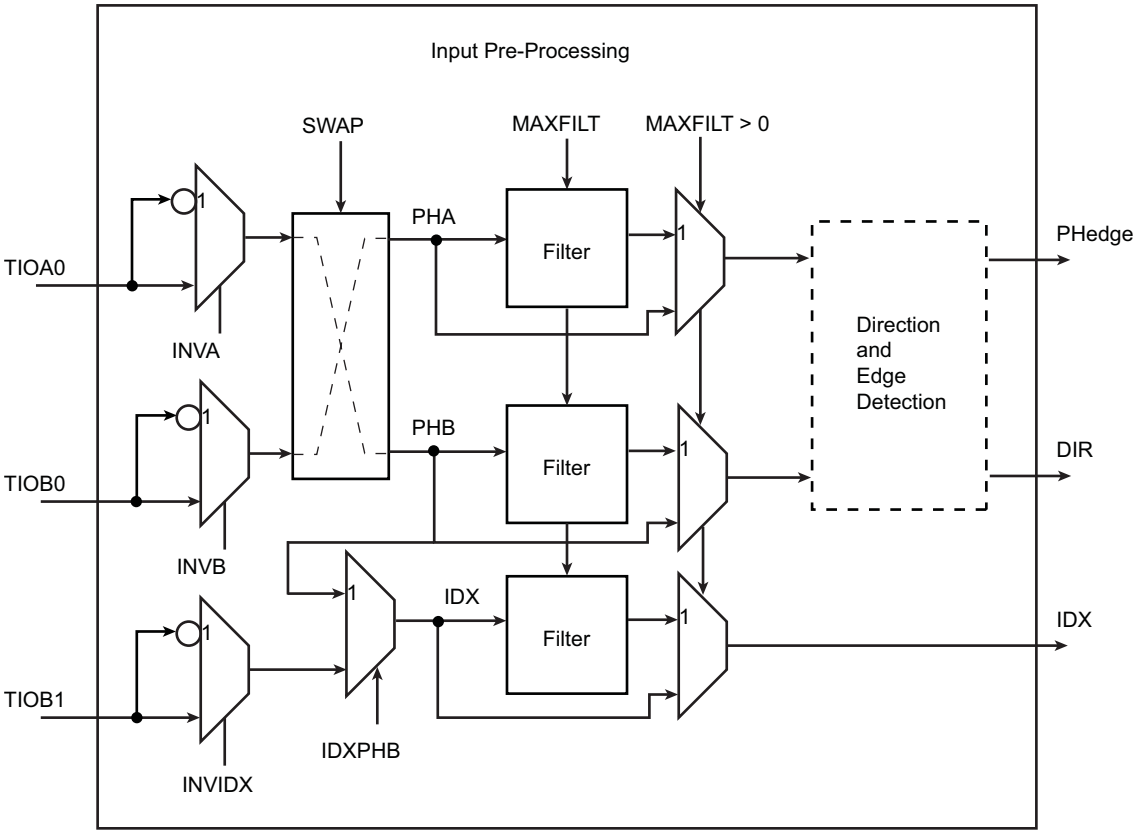
38.6.16.2 Input Pre-processing

Input pre-processing consists of capabilities to take into account rotary sensor factors such as polarities and phase definition followed by configurable digital filtering.

Each input can be negated and 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.

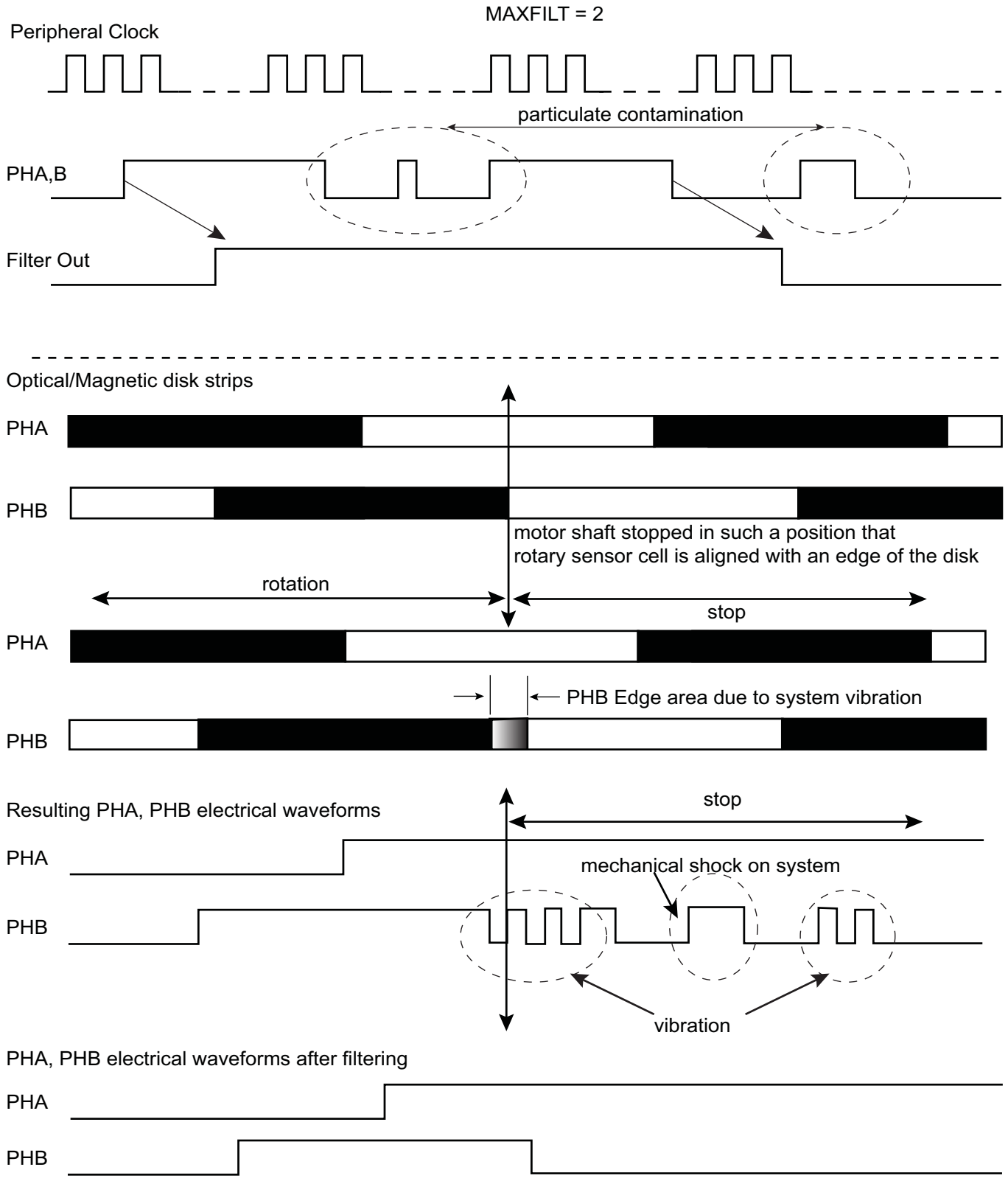
Figure 38-18. Input Stage



Input filtering can efficiently remove spurious pulses that might be generated by the presence of particulate contamination on the optical or magnetic disk of the rotary sensor.

Spurious pulses can also occur in environments with high levels of electro-magnetic interference. Or, simply if vibration occurs even when rotation is fully stopped and the shaft of the motor is in such a position that the beginning of one of the reflective or magnetic bars on the rotary sensor disk is aligned with the light or magnetic (Hall) receiver cell of the rotary sensor. Any vibration can make the PHA, PHB signals toggle for a short duration.

Figure 38-19. Filtering Examples



38.6.16.3 Direction Status and Change Detection

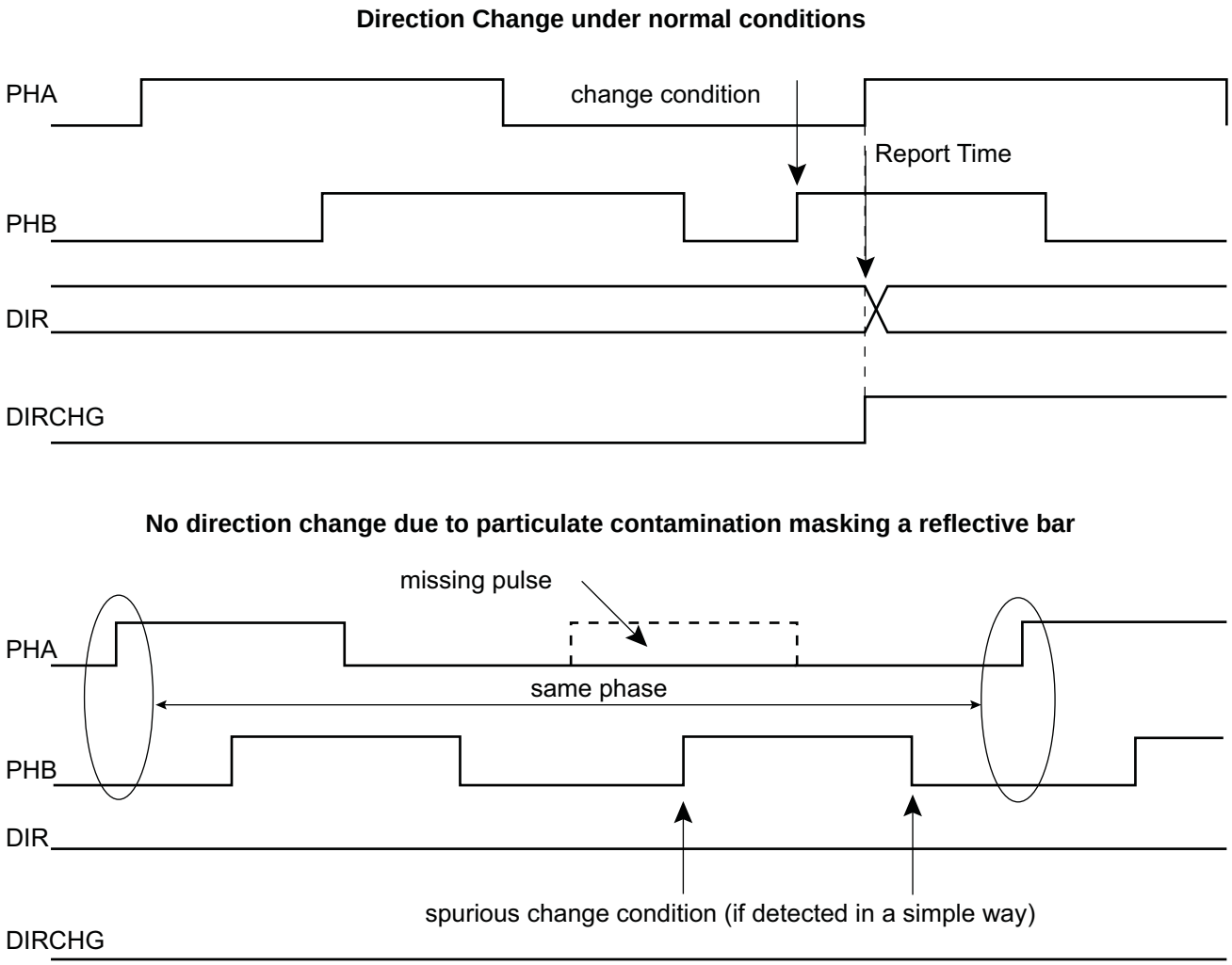
After filtering, the quadrature signals are analyzed to extract the rotation direction and edges of the two quadrature signals detected in order to be counted by timer/counter logic downstream.

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 38-20 for waveforms.

Figure 38-20. Rotation Change Detection

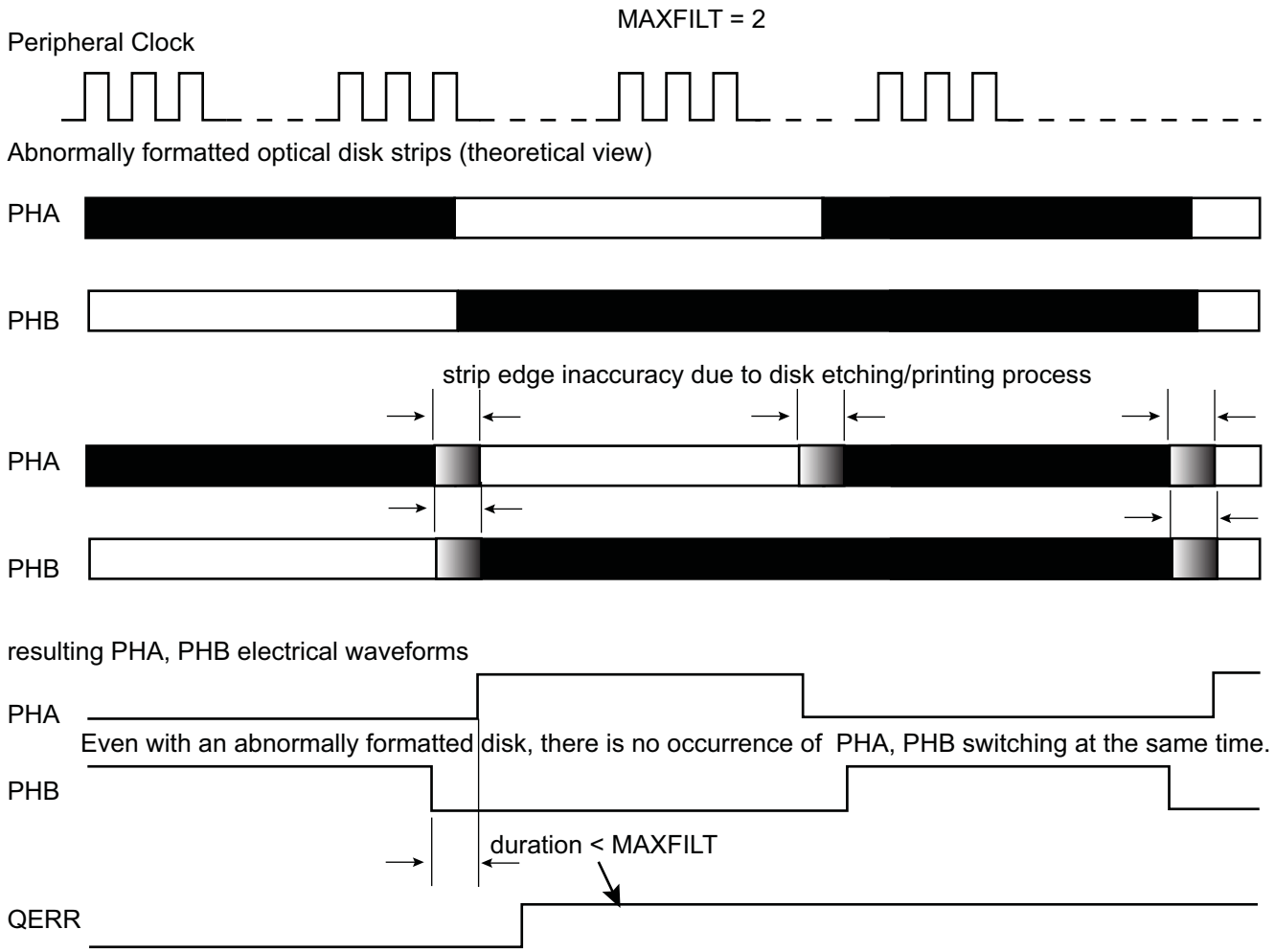


The direction change detection is disabled when QDTRANS is set in the TC_BMR. In this case, the DIR flag report must not be used.

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 38-21. 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.

38.6.16.4 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. If no IDX signal is available, the internal counter can be cleared for each revolution if the number of counts per revolution is configured in TC_RC0.RC and the TC_CMR.CPCTRG bit is written to 1. 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 ‘TIOAx’ 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.

38.6.16.5 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 TIOAx output.

This time base is automatically fed back to TIOAx 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 TIOAx as a trigger for this channel.

EDGTRG must be set to 0x01, to clear the counter on a rising edge of the TIOAx 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.

38.6.16.6 Detecting a Missing Index Pulse

To detect a missing index pulse due contamination, dust, etc., the TC_SR0.CPCS flag can be used. It is also possible to assert the interrupt line if the TC_SR0.CPCS flag is enabled as a source of the interrupt by writing a ‘1’ to TC_IER0.CPCS.

The TC_RC0.RC field must be written with the nominal number of counts per revolution provided by the rotary encoder, plus a margin to eliminate potential noise (e.g., if nominal count per revolution is 1024, then TC_RC0.RC=1028).

If the index pulse is missing, the timer value is not cleared and the nominal value is exceeded, then the comparator on the RC triggers an event, TC_SR0.CPCS=1, and the interrupt line is asserted if TC_IER0.CPCS=1.

38.6.17 2-bit Gray Up/Down Counter for Stepper Motor

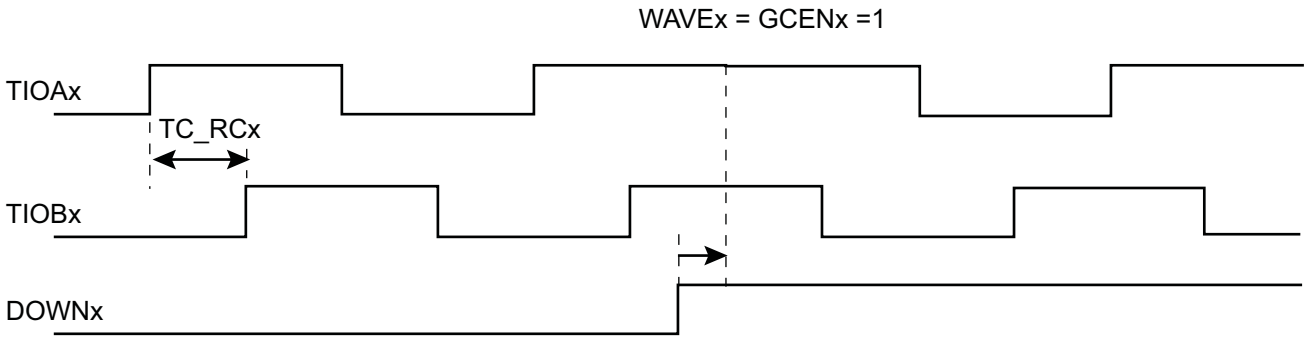
Each channel can be independently configured to generate a 2-bit Gray count waveform on corresponding TIOAx, TIOBx outputs by means of the GCEN bit in TC_SMMRx.

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 38-22. 2-bit Gray Up/Down Counter



38.6.18 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.

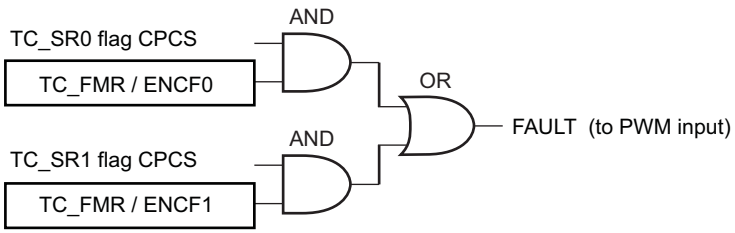
The CPCSx flags can be set accordingly and an interrupt can be generated.

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 38-23. Fault Output Generation



38.6.19 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
- TC Extended Mode Register

38.7 Timer Counter (TC) User Interface

Table 38-6. 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	Register AB	TC_RAB	Read-only	0
0x00 + channel * 0x40 + 0x10	Counter Value	TC_CV	Read-only	0
0x00 + channel * 0x40 + 0x14	Register A	TC_RA	Read/Write[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
0x00 + channel * 0x40 + 0x30	Extended Mode Register	TC_EMR	Read/Write	0
0xC0	Block Control Register	TC_BCR	Write-only	–
0xC4	Block Mode Register	TC_BMR	Read/Write	0
0xC8	QDEC Interrupt Enable Register	TC_QIER	Write-only	–
0xCC	QDEC Interrupt Disable Register	TC_QIDR	Write-only	–
0xD0	QDEC Interrupt Mask Register	TC_QIMR	Read-only	0
0xD4	QDEC Interrupt Status Register	TC_QISR	Read-only	0
0xD8	Fault Mode Register	TC_FMR	Read/Write	0
0xE4	Write Protection Mode Register	TC_WPMR	Read/Write	0
0xE8–0xFC	Reserved	–	–	–
0x100–0x1A4	Reserved for PDC Registers	–	–	–

Notes: 1. Channel index ranges from 0 to 2.
2. Read-only if TC_CMRx.WAVE = 0

38.7.1 TC Channel Control Register

Name: TC_CCRx [x=0..2]

Address: 0x40090000 (0)[0], 0x40090040 (0)[1], 0x40090080 (0)[2], 0x40094000 (1)[0], 0x40094040 (1)[1], 0x40094080 (1)[2], 0x40098000 (2)[0], 0x40098040 (2)[1], 0x40098080 (2)[2]

Access: Write-only

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

- **CLKEN: Counter Clock Enable Command**

0: No effect.

1: Enables the clock if CLKDIS is not 1.

- **CLKDIS: Counter Clock Disable Command**

0: No effect.

1: Disables the clock.

- **SWTRG: Software Trigger Command**

0: No effect.

1: A software trigger is performed: the counter is reset and the clock is started.

38.7.2 TC Channel Mode Register: Capture Mode

Name: TC_CMRx [x=0..2] (CAPTURE_MODE)

Address: 0x40090004 (0)[0], 0x40090044 (0)[1], 0x40090084 (0)[2], 0x40094004 (1)[0], 0x40094044 (1)[1], 0x40094084 (1)[2], 0x40098004 (2)[0], 0x40098044 (2)[1], 0x40098084 (2)[2]

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	SBSMPLR			LDRB		LDRA	
15	14	13	12	11	10	9	8
WAVE	CPCTRG	–	–	–	ABETRG	ETRGEDG	
7	6	5	4	3	2	1	0
LDBDIS	LDBSTOP	BURST		CLKI	TCCLKS		

This register can only be written if the WPEN bit is cleared in 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

To operate at maximum peripheral clock frequency, refer to Section 38.7.14 “TC Extended Mode Register”.

- **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: TIOAx or TIOBx External Trigger Selection**

0: TIOBx is used as an external trigger.

1: TIOAx 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 TIOAx
2	FALLING	Falling edge of TIOAx
3	EDGE	Each edge of TIOAx

- **LDRB: RB Loading Edge Selection**

Value	Name	Description
0	NONE	None
1	RISING	Rising edge of TIOAx
2	FALLING	Falling edge of TIOAx
3	EDGE	Each edge of TIOAx

- **SBSMPLR: Loading Edge Subsampling Ratio**

Value	Name	Description
0	ONE	Load a Capture Register each selected edge
1	HALF	Load a Capture Register every 2 selected edges
2	FOURTH	Load a Capture Register every 4 selected edges
3	EIGHTH	Load a Capture Register every 8 selected edges
4	SIXTEENTH	Load a Capture Register every 16 selected edges

38.7.3 TC Channel Mode Register: Waveform Mode

Name: TC_CMRx [x=0..2] (WAVEFORM_MODE)

Address: 0x40090004 (0)[0], 0x40090044 (0)[1], 0x40090084 (0)[2], 0x40094004 (1)[0], 0x40094044 (1)[1], 0x40094084 (1)[2], 0x40098004 (2)[0], 0x40098044 (2)[1], 0x40098084 (2)[2]

Access: Read/Write

31	30	29	28	27	26	25	24
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

To operate at maximum peripheral clock frequency, refer to Section 38.7.14 “TC Extended Mode Register”.

- **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 TIOAx output and TIOBx 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 TIOAx**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **ACPC: RC Compare Effect on TIOAx**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **AEEVT: External Event Effect on TIOAx**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **ASWTRG: Software Trigger Effect on TIOAx**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **BCPB: RB Compare Effect on TIOBx**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **BCPC: RC Compare Effect on TIOBx**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **BEEVT: External Event Effect on TIOBx**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **BSWTRG: Software Trigger Effect on TIOBx**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

38.7.4 TC Stepper Motor Mode Register

Name: TC_SMMRx [x=0..2]

Address: 0x40090008 (0)[0], 0x40090048 (0)[1], 0x40090088 (0)[2], 0x40094008 (1)[0], 0x40094048 (1)[1], 0x40094088 (1)[2], 0x40098008 (2)[0], 0x40098048 (2)[1], 0x40098088 (2)[2]

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in 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.

38.7.5 TC Register AB

Name: TC_RABx [x=0..2]

Address: 0x4009000C (0)[0], 0x4009004C (0)[1], 0x4009008C (0)[2], 0x4009400C (1)[0], 0x4009404C (1)[1], 0x4009408C (1)[2], 0x4009800C (2)[0], 0x4009804C (2)[1], 0x4009808C (2)[2]

Access: Read-only

31	30	29	28	27	26	25	24
RAB							

23	22	21	20	19	18	17	16
RAB							

15	14	13	12	11	10	9	8
RAB							

7	6	5	4	3	2	1	0
RAB							

- **RAB: Register A or Register B**

RAB contains the next unread capture Register A or Register B value in real time. It is usually read by the DMA after a request due to a valid load edge on TIOAx.

When DMA is used, the RAB register address must be configured as source address of the transfer.

38.7.6 TC Counter Value Register

Name: TC_CVx [x=0..2]

Address: 0x40090010 (0)[0], 0x40090050 (0)[1], 0x40090090 (0)[2], 0x40094010 (1)[0], 0x40094050 (1)[1], 0x40094090 (1)[2], 0x40098010 (2)[0], 0x40098050 (2)[1], 0x40098090 (2)[2]

Access: Read-only

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.

38.7.7 TC Register A

Name: TC_RAx [x=0..2]

Address: 0x40090014 (0)[0], 0x40090054 (0)[1], 0x40090094 (0)[2], 0x40094014 (1)[0], 0x40094054 (1)[1], 0x40094094 (1)[2], 0x40098014 (2)[0], 0x40098054 (2)[1], 0x40098094 (2)[2]

Access: Read-only if 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.

38.7.8 TC Register B

Name: TC_RBx [x=0..2]

Address: 0x40090018 (0)[0], 0x40090058 (0)[1], 0x40090098 (0)[2], 0x40094018 (1)[0], 0x40094058 (1)[1], 0x40094098 (1)[2], 0x40098018 (2)[0], 0x40098058 (2)[1], 0x40098098 (2)[2]

Access: Read-only if 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.

38.7.9 TC Register C

Name: TC_RCx [x=0..2]

Address: 0x4009001C (0)[0], 0x4009005C (0)[1], 0x4009009C (0)[2], 0x4009401C (1)[0], 0x4009405C (1)[1], 0x4009409C (1)[2], 0x4009801C (2)[0], 0x4009805C (2)[1], 0x4009809C (2)[2]

Access: Read/Write

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.

38.7.10 TC Status Register

Name: TC_SRx [x=0..2]

Address: 0x40090020 (0)[0], 0x40090060 (0)[1], 0x400900A0 (0)[2], 0x40094020 (1)[0], 0x40094060 (1)[1], 0x400940A0 (1)[2], 0x40098020 (2)[0], 0x40098060 (2)[1], 0x400980A0 (2)[2]

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	MTIOB	MTIOA	CLKSTA
15	14	13	12	11	10	9	8
–	–	–	–	–	–	RXBUFF	ENDRX
7	6	5	4	3	2	1	0
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow Status (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.

- **ENDRX: End of Receiver Transfer (cleared by writing TC_RCR or TC_RNCR)**

0: The Receive Counter Register has not reached 0 since the last write in TC_RCR[(1)] or TC_RNCR[(1)].

1: The Receive Counter Register has reached 0 since the last write in TC_RCR or TC_RNCR.

- **RXBUFF: Reception Buffer Full (cleared by writing TC_RCR or TC_RNCR)**

0: TC_RCR or TC_RNCR have a value other than 0.

1: Both TC_RCR and TC_RNCR have a value of 0.

Note: 1. TC_RCR and TC_RNCR are PDC registers.

- **CLKSTA: Clock Enabling Status**

0: Clock is disabled.

1: Clock is enabled.

- **MTIOA: TIOAx Mirror**

0: TIOAx is low. If TC_CMRx.WAVE = 0, this means that TIOAx pin is low. If TC_CMRx.WAVE = 1, this means that TIOAx is driven low.

1: TIOAx is high. If TC_CMRx.WAVE = 0, this means that TIOAx pin is high. If TC_CMRx.WAVE = 1, this means that TIOAx is driven high.

- **MTIOB: TIOBx Mirror**

0: TIOBx is low. If TC_CMRx.WAVE = 0, this means that TIOBx pin is low. If TC_CMRx.WAVE = 1, this means that TIOBx is driven low.

1: TIOBx is high. If TC_CMRx.WAVE = 0, this means that TIOBx pin is high. If TC_CMRx.WAVE = 1, this means that TIOBx is driven high.

38.7.11 TC Interrupt Enable Register

Name: TC_IERx [x=0..2]

Address: 0x40090024 (0)[0], 0x40090064 (0)[1], 0x400900A4 (0)[2], 0x40094024 (1)[0], 0x40094064 (1)[1], 0x400940A4 (1)[2], 0x40098024 (2)[0], 0x40098064 (2)[1], 0x400980A4 (2)[2]

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	RXBUFF	ENDRX
7	6	5	4	3	2	1	0
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow**

0: No effect.

1: Enables the Counter Overflow Interrupt.

- **LOVRS: Load Overrun**

0: No effect.

1: Enables the Load Overrun Interrupt.

- **CPAS: RA Compare**

0: No effect.

1: Enables the RA Compare Interrupt.

- **CPBS: RB Compare**

0: No effect.

1: Enables the RB Compare Interrupt.

- **CPCS: RC Compare**

0: No effect.

1: Enables the RC Compare Interrupt.

- **LDRAS: RA Loading**

0: No effect.

1: Enables the RA Load Interrupt.

- **LDRBS: RB Loading**

0: No effect.

1: Enables the RB Load Interrupt.

- **ETRGS: External Trigger**

0: No effect.

1: Enables the External Trigger Interrupt.

- **ENDRX: End of Receiver Transfer**

0: No effect.

1: Enables the PDC Receive End of Transfer Interrupt.

- **RXBUFF: Reception Buffer Full**

0: No effect.

1: Enables the PDC Receive Buffer Full Interrupt.

38.7.12 TC Interrupt Disable Register

Name: TC_IDRx [x=0..2]

Address: 0x40090028 (0)[0], 0x40090068 (0)[1], 0x400900A8 (0)[2], 0x40094028 (1)[0], 0x40094068 (1)[1], 0x400940A8 (1)[2], 0x40098028 (2)[0], 0x40098068 (2)[1], 0x400980A8 (2)[2]

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	RXBUFF	ENDRX
7	6	5	4	3	2	1	0
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow**

0: No effect.

1: Disables the Counter Overflow Interrupt.

- **LOVRS: Load Overrun**

0: No effect.

1: Disables the Load Overrun Interrupt (if 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.

- **ENDRX: End of Receiver Transfer**

0: No effect.

1: Disables the PDC Receive End of Transfer Interrupt.

- **RXBUFF: Reception Buffer Full**

0: No effect.

1: Disables the PDC Receive Buffer Full Interrupt.

38.7.13 TC Interrupt Mask Register

Name: TC_IMRx [x=0..2]

Address: 0x4009002C (0)[0], 0x4009006C (0)[1], 0x400900AC (0)[2], 0x4009402C (1)[0], 0x4009406C (1)[1], 0x400940AC (1)[1][2], 0x4009802C (2)[0], 0x4009806C (2)[1], 0x400980AC (2)[2]

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	RXBUFF	ENDRX
7	6	5	4	3	2	1	0
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow**

0: The Counter Overflow Interrupt is disabled.

1: The Counter Overflow Interrupt is enabled.

- **LOVRS: Load Overrun**

0: The Load Overrun Interrupt is disabled.

1: The Load Overrun Interrupt is enabled.

- **CPAS: RA Compare**

0: The RA Compare Interrupt is disabled.

1: The RA Compare Interrupt is enabled.

- **CPBS: RB Compare**

0: The RB Compare Interrupt is disabled.

1: The RB Compare Interrupt is enabled.

- **CPCS: RC Compare**

0: The RC Compare Interrupt is disabled.

1: The RC Compare Interrupt is enabled.

- **LDRAS: RA Loading**

0: The Load RA Interrupt is disabled.

1: The Load RA Interrupt is enabled.

- **LDRBS: RB Loading**

0: The Load RB Interrupt is disabled.

1: The Load RB Interrupt is enabled.

- **ETRGS: External Trigger**

0: The External Trigger Interrupt is disabled.

1: The External Trigger Interrupt is enabled.

- **ENDRX: End of Receiver Transfer**

0: The PDC Receive End of Transfer Interrupt is disabled.

1: The PDC Receive End of Transfer Interrupt is enabled.

- **RXBUFF: Reception Buffer Full**

0: The PDC Receive Buffer Full Interrupt is disabled.

1: The PDC Receive Buffer Full Interrupt is enabled.

38.7.14 TC Extended Mode Register

Name: TC_EMRx [x=0..2]

Address: 0x40090030 (0)[0], 0x40090070 (0)[1], 0x400900B0 (0)[2], 0x40094030 (1)[0], 0x40094070 (1)[1], 0x400940B0 (1)[2], 0x40098030 (2)[0], 0x40098070 (2)[1], 0x400980B0 (2)[2]

Access: Read/Write

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

- **TRIGSRCA: Trigger Source for Input A**

Value	Name	Description
0	EXTERNAL_TIOAx	The trigger/capture input A is driven by external pin TIOAx
1	PWMx	The trigger/capture input A is driven internally by PWMx

- **TRIGSRCB: Trigger Source for Input B**

Value	Name	Description
0	EXTERNAL_TIOBx	The trigger/capture input B is driven by external pin TIOBx
1	PWMx	The trigger/capture input B is driven internally by the comparator output (see Figure 38-16) of the PWMx.

- **NODIVCLK: No Divided Clock**

0: The selected clock is defined by field TCCLKS in TC_CMRx.

1: The selected clock is peripheral clock and TCCLKS field (TC_CMRx) has no effect.

38.7.15 TC Block Control Register

Name: TC_BCR

Address: 0x400900C0 (0), 0x400940C0 (1), 0x400980C0 (2)

Access: Write-only

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

- **SYNC: Synchro Command**

0: No effect.

1: Asserts the SYNC signal which generates a software trigger simultaneously for each of the channels.

38.7.16 TC Block Mode Register

Name: TC_BMR

Address: 0x400900C4 (0), 0x400940C4 (1), 0x400980C4 (2)

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	MAXFILT	

23	22	21	20	19	18	17	16
MAXFILT				–	–	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.

38.7.17 TC QDEC Interrupt Enable Register

Name: TC_QIER

Address: 0x400900C8 (0), 0x400940C8 (1), 0x400980C8 (2)

Access: Write-only

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

- **IDX: Index**

0: No effect.

1: Enables the interrupt when a rising edge occurs on IDX input.

- **DIRCHG: Direction Change**

0: No effect.

1: Enables the interrupt when a change on rotation direction is detected.

- **QERR: Quadrature Error**

0: No effect.

1: Enables the interrupt when a quadrature error occurs on PHA, PHB.

38.7.18 TC QDEC Interrupt Disable Register

Name: TC_QIDR

Address: 0x400900CC (0), 0x400940CC (1), 0x400980CC (2)

Access: Write-only

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

- **IDX: Index**

0: No effect.

1: Disables the interrupt when a rising edge occurs on IDX input.

- **DIRCHG: Direction Change**

0: No effect.

1: Disables the interrupt when a change on rotation direction is detected.

- **QERR: Quadrature Error**

0: No effect.

1: Disables the interrupt when a quadrature error occurs on PHA, PHB.

38.7.19 TC QDEC Interrupt Mask Register

Name: TC_QIMR

Address: 0x400900D0 (0), 0x400940D0 (1), 0x400980D0 (2)

Access: Read-only

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

- **IDX: Index**

0: The interrupt on IDX input is disabled.

1: The interrupt on IDX input is enabled.

- **DIRCHG: Direction Change**

0: The interrupt on rotation direction change is disabled.

1: The interrupt on rotation direction change is enabled.

- **QERR: Quadrature Error**

0: The interrupt on quadrature error is disabled.

1: The interrupt on quadrature error is enabled.

38.7.20 TC QDEC Interrupt Status Register

Name: TC_QISR

Address: 0x400900D4 (0), 0x400940D4 (1), 0x400980D4 (2)

Access: Read-only

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

- **IDX: Index**

0: No Index input change since the last read of TC_QISR.

1: The IDX input has changed since the last read of TC_QISR.

- **DIRCHG: Direction Change**

0: No change on rotation direction since the last read of TC_QISR.

1: The rotation direction changed since the last read of TC_QISR.

- **QERR: Quadrature Error**

0: No quadrature error since the last read of TC_QISR.

1: A quadrature error occurred since the last read of TC_QISR.

- **DIR: Direction**

Returns an image of the actual rotation direction.

38.7.21 TC Fault Mode Register

Name: TC_FMR

Address: 0x400900D8 (0), 0x400940D8 (1), 0x400980D8 (2)

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	–	–	ENCF1	ENCF0

This register can only be written if the WPEN bit is cleared in 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.

38.7.22 TC Write Protection Mode Register

Name: TC_WPMR

Address: 0x400900E4 (0), 0x400940E4 (1), 0x400980E4 (2)

Access: Read/Write

31	30	29	28	27	26	25	24
WPKEY							

23	22	21	20	19	18	17	16
WPKEY							

15	14	13	12	11	10	9	8
WPKEY							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WPEN

- **WPEN: Write 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 38.6.19 “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.

39. Pulse Width Modulation Controller (PWM)

39.1 Description

The Pulse Width Modulation Controller (PWM) generates output pulses on 4 channels independently according to parameters defined per channel. 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 peripheral clock.

All accesses to the PWM are made through registers mapped on the peripheral bus. All channels integrate a double buffering system in order to prevent an unexpected output waveform while modifying the period, the spread spectrum, the duty-cycle 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 DMA Controller channel which offers buffer transfer without processor Intervention.

The PWM includes a spread-spectrum counter to allow a constantly varying period (only for Channel 0). This counter may be useful to minimize electromagnetic interference or to reduce the acoustic noise of a PWM driven motor.

The PWM 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 DMA Controllertransfer requests.

PWM outputs can be overridden synchronously or asynchronously to their channel counter.

The PWM provides a fault protection mechanism with 8 fault inputs**,** capable to detect a fault condition and to override the PWM outputs asynchronously (outputs forced to ‘0’, ‘1’ or Hi-Z).

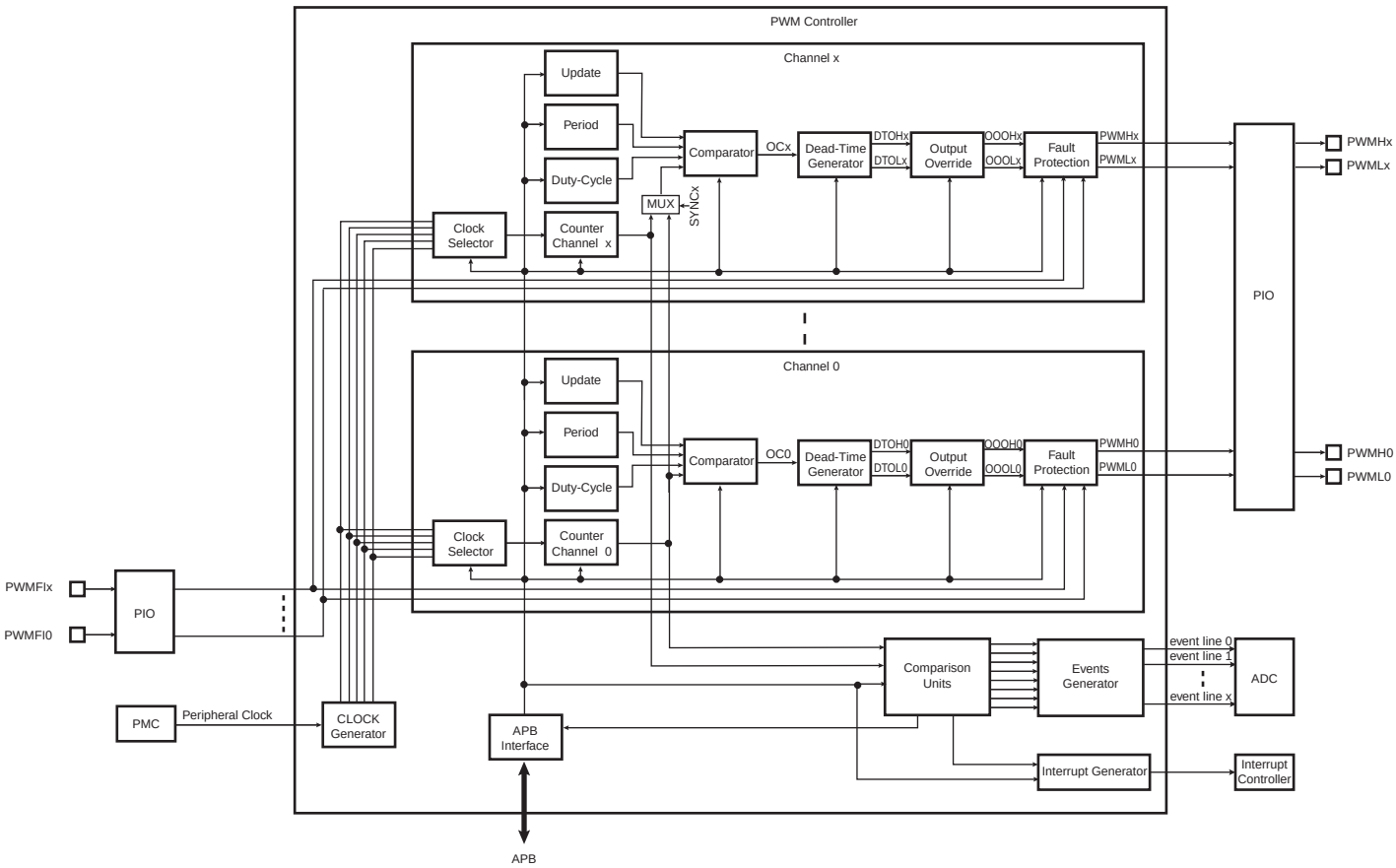
For safety usage, some configuration registers are write-protected.

39.2 Embedded Characteristics

- 4 Channels
- Common Clock Generator Providing Thirteen Different Clocks
  - A Modulo n Counter Providing Eleven Clocks
  - Two Independent Linear Dividers Working on Modulo n Counter Outputs
- Independent Channels
  - Independent 16-bit Counter for Each Channel
  - Independent Complementary Outputs with 12-bit Dead-Time Generator (Also Called Dead-Band or Non-Overlapping Time) for Each Channel
  - Independent Enable Disable Command for Each Channel
  - Independent Clock Selection for Each Channel
  - Independent Period, Duty-Cycle and Dead-Time for Each Channel
  - Independent Double Buffering of Period, Duty-Cycle and Dead-Times for Each Channel
  - Independent Programmable Selection of The Output Waveform Polarity for Each Channel, with Double Buffering
  - Independent Programmable Center- or Left-aligned Output Waveform for Each Channel
  - Independent Output Override for Each Channel
  - Independent Interrupt for Each Channel, at Each Period for Left-Aligned or Center-Aligned Configuration
  - Independent Update Time Selection of Double Buffering Registers (Polarity, Duty Cycle) for Each Channel, at Each Period for Left-Aligned or Center-Aligned Configuration
- 2 2-bit Gray Up/Down Channels for Stepper Motor Control
- Spread Spectrum Counter to Allow a Constantly Varying Duty Cycle (only for Channel 0)
- Synchronous Channel Mode
  - Synchronous Channels Share the Same Counter
  - Mode to Update the Synchronous Channels Registers after a Programmable Number of Periods
  - Synchronous Channels Supports Connection of one Peripheral DMA Controller Channel Which Offers Buffer Transfer Without Processor Intervention To Update Duty-Cycle Registers
- 2 Independent Events Lines Intended to Synchronize ADC Conversions
  - Programmable delay for Events Lines to delay ADC measurements
- 8 Comparison Units Intended to Generate Interrupts, Pulses on Event Lines and Peripheral DMA Controller Transfer Requests
- 8 Programmable Fault Inputs Providing an Asynchronous Protection of PWM Outputs
  - 1 User Driven through PIO Inputs
  - PMC Driven when Crystal Oscillator Clock Fails
  - ADC Controller Driven through Configurable Comparison Function
  - Analog Comparator Controller Driven
  - Timer/Counter Driven through Configurable Comparison Function
- Register Write Protection

39.3 Block Diagram

Figure 39-1. Pulse Width Modulation Controller Block Diagram



39.4 I/O Lines Description

Each channel outputs two complementary external I/O lines.

Table 39-1. 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

39.5 Product Dependencies

39.5.1 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 are assigned to PWM outputs.

39.5.2 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.

39.5.3 Interrupt Sources

The PWM interrupt line is connected on one of the internal sources of the Interrupt Controller. Using the PWM interrupt requires the Interrupt Controller to be programmed first.

39.5.4 Fault Inputs

The PWM has the fault inputs connected to the different modules. Refer to the implementation of these modules within the product for detailed information about the fault generation procedure. The PWM receives faults from:

- PIO inputs
- the PMC
- the ADC controller
- the Analog Comparator Controller
- Timer/Counters

Table 39-2. Fault Inputs

Fault Generator	External PWM Fault Input Number	Polarity Level(1)	Fault Input ID
Main OSC (PMC)	–	To be configured to 1	0
ADC	–	To be configured to 1	1
PXyy	PWMFI0	User-defined	2
PXyy	PWMFI1	User-defined	3
PXyy	PWMFI2	User-defined	4
PXyy	PWMFI3	User-defined	5
PXyy	PWMFI4	User-defined	6
PXyy	PWMFI5	User-defined	7

Note: 1. FPOL field in PWMC_FMR.

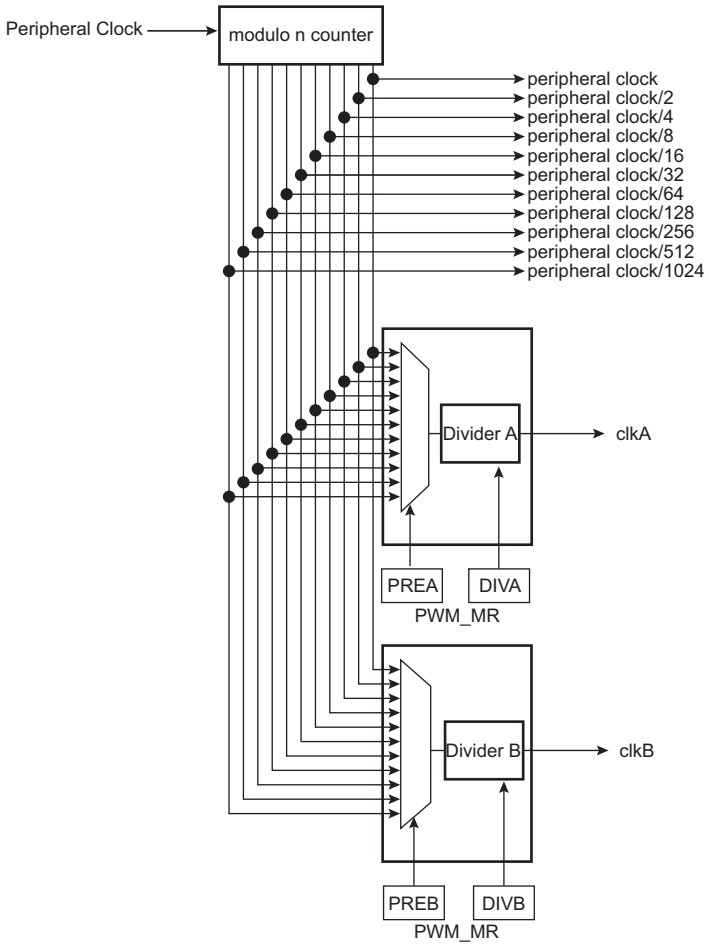
39.6 Functional Description

The PWM controller is primarily composed of a clock generator module and 4 channels.

- Clocked by the peripheral clock, 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.

39.6.1 PWM Clock Generator

Figure 39-2. Functional View of the Clock Generator Block Diagram



The PWM peripheral clock 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 into different blocks:

- a modulo n counter which provides 11 clocks: $f_{peripheral\ clock}$, $f_{peripheral\ clock}/2$, $f_{peripheral\ clock}/4$, $f_{peripheral\ clock}/8$, $f_{peripheral\ clock}/16$, $f_{peripheral\ clock}/32$, $f_{peripheral\ clock}/64$, $f_{peripheral\ clock}/128$, $f_{peripheral\ clock}/256$, $f_{peripheral\ clock}/512$, $f_{peripheral\ clock}/1024$
- 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 the peripheral clock. This situation is also true when the PWM peripheral clock is turned off through the Power Management Controller.

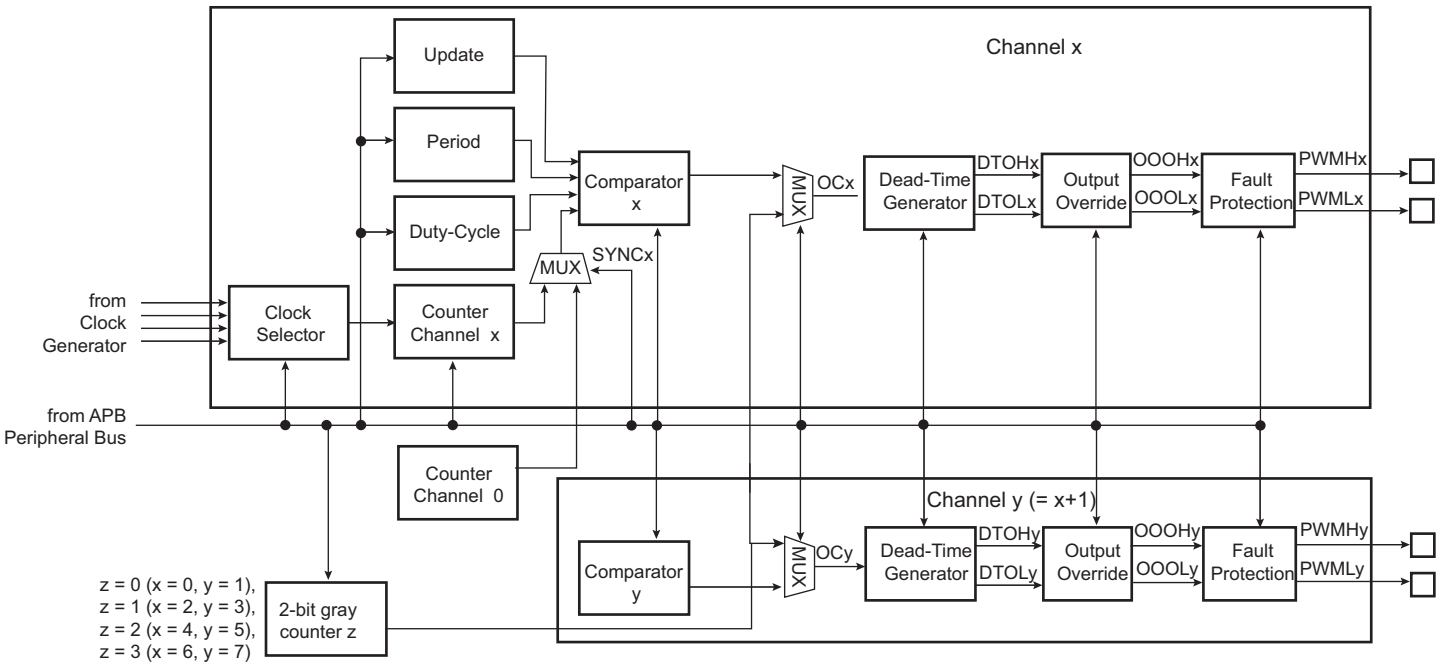
CAUTION:

Before using the PWM controller, the programmer must first enable the peripheral clock in the Power Management Controller (PMC).

39.6.2 PWM Channel

39.6.2.1 Channel Block Diagram

Figure 39-3. Functional View of the Channel Block Diagram



Each of the 4 channels is composed of six blocks:

- A clock selector which selects one of the clocks provided by the clock generator (described in Section 39.6.1 “PWM Clock Generator”).
- 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 (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 (PWMHx/PWMLx) in case of fault detection (outputs forced to ‘0’, ‘1’ or Hi-Z).

39.6.2.2 Comparator

The comparator continuously compares its counter value with the channel period defined by CPRD in the PWM Channel Period Register (PWM_CPRDx) and the duty-cycle defined by CDTY in the PWM Channel Duty Cycle Register (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 (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 peripheral clock divided by a given prescaler value "X" (where X = 2^{PREA} is 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024). The resulting period formula is:

  $$\frac{(X \times CPRD)}{f_{\text{peripheral clock}}}$$

  By using the PWM peripheral clock divided by a given prescaler value "X" (see above) and by either the DIVA or the DIVB divider. The formula becomes, respectively:

  $$\frac{(X \times CRPD \times DIVA)}{f_{\text{peripheral clock}}} \text{ or } \frac{(X \times CRPD \times DIVB)}{f_{\text{peripheral clock}}}$$

  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 peripheral clock divided by a given prescaler value "X" (where X = 2^{PREA} is 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024). The resulting period formula is:

  $$\frac{(2 \times X \times CPRD)}{f_{\text{peripheral clock}}}$$

  By using the PWM peripheral clock divided by a given prescaler value "X" (see above) and by either the DIVA or the DIVB divider. The formula becomes, respectively:

  $$\frac{(2 \times X \times CPRD \times DIVA)}{f_{\text{peripheral clock}}} \text{ or } \frac{(2 \times X \times CPRD \times DIVB)}{f_{\text{peripheral clock}}}$$

- 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:

  $$\text{duty cycle} = \frac{(period - 1/\text{fchannel\_x\_clock} \times CDTY)}{period}$$

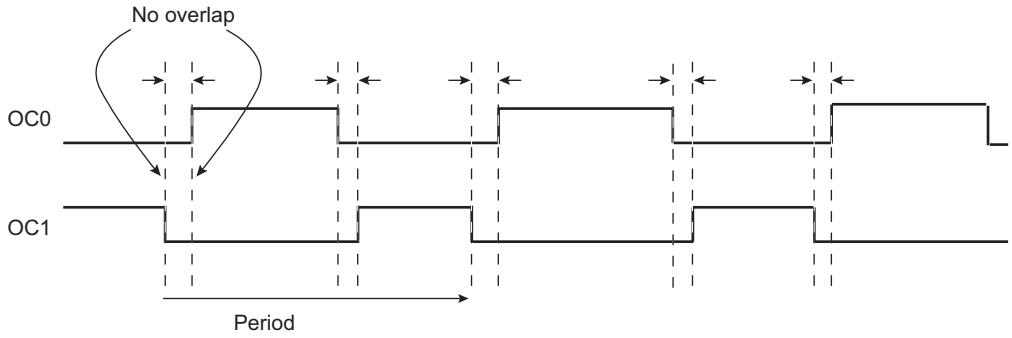
  If the waveform is center-aligned, then:

  $$\text{duty cycle} = \frac{((period/2) - 1/\text{fchannel\_x\_clock} \times CDTY))}{(period/2)}$$

- 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 bit of PWM_CMRx. 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 bit of PWM_CMRx. The default mode is left-aligned.

Figure 39-4. Non-Overlapped Center-Aligned Waveforms



Note: 1. See Figure 39-5 for a detailed description of center-aligned waveforms.

When center-aligned, the channel counter increases up to CPRD and decreases down to 0. This ends the period.

When left-aligned, the channel counter increases up to CPRD and is reset. This ends the period.

Thus, for the same CPRD value, the period for a center-aligned channel is twice the period for a left-aligned channel.

Waveforms are fixed at 0 when:

- CDTY = CPRD and CPOL = 0
- CDTY = 0 and CPOL = 1

Waveforms are fixed at 1 (once the channel is enabled) when:

- CDTY = 0 and CPOL = 0
- CDTY = CPRD and CPOL = 1

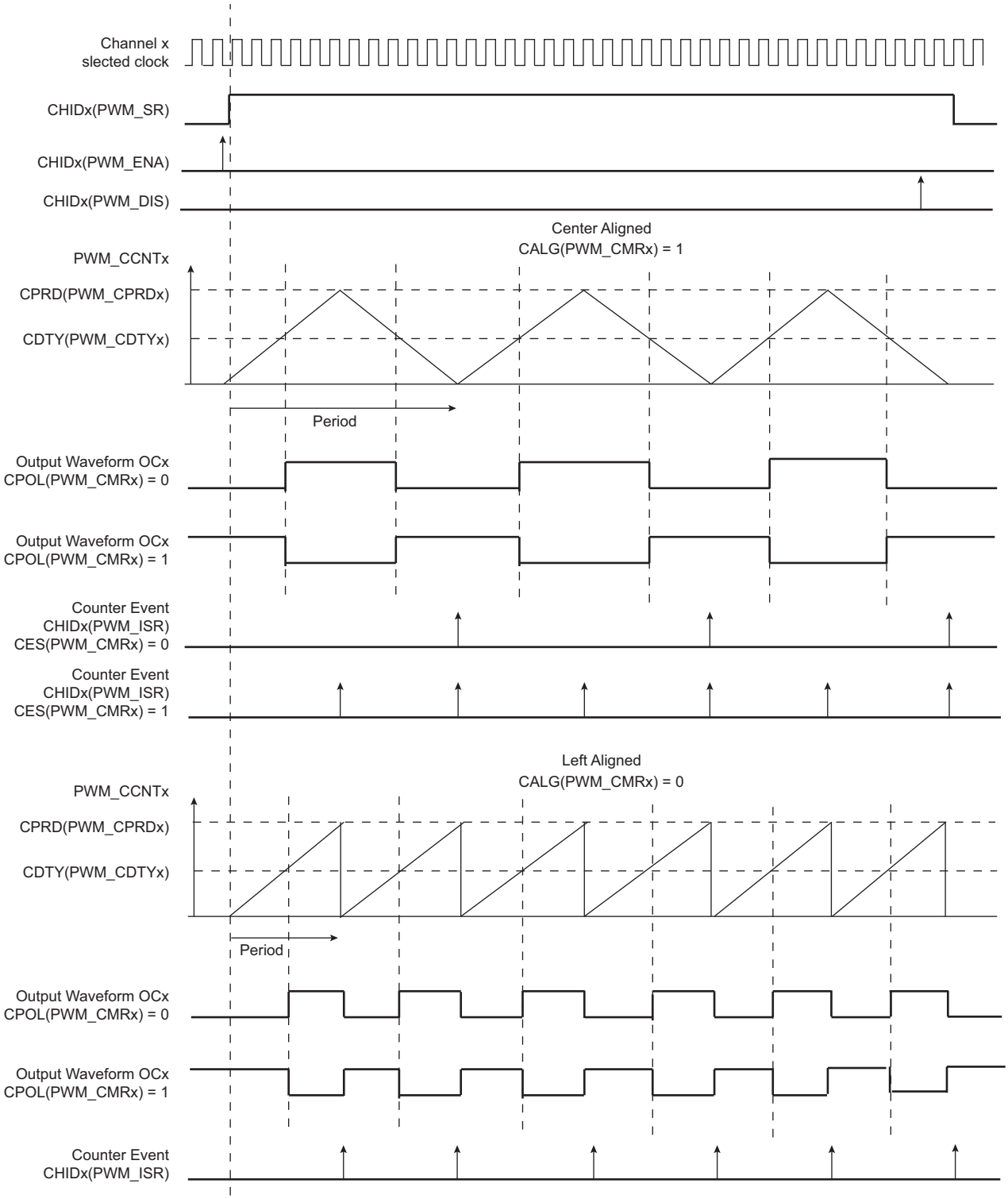
The waveform polarity must be set before enabling the channel. This immediately affects the channel output level.

Modifying CPOL in PWM Channel Mode Register while the channel is enabled can lead to an unexpected behavior of the device being driven by PWM.

In addition to generating the output signals OCx, the comparator generates interrupts depending on 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 PWM_CMRx 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 39-5 illustrates the counter interrupts depending on the configuration.

Figure 39-5. Waveform Properties



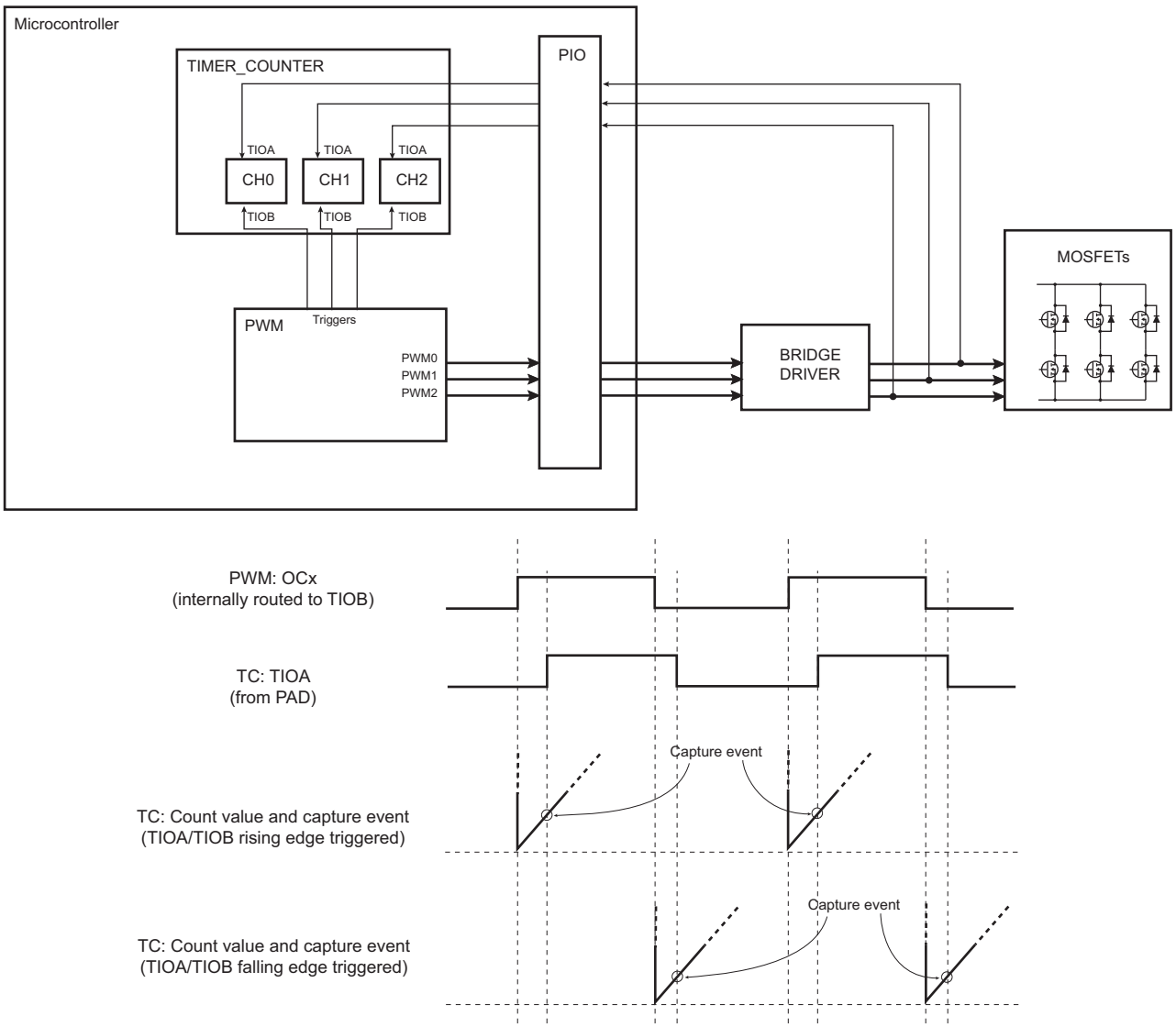
39.6.2.3 Trigger Selection for Timer Counter

The PWM controller can be used as a trigger source for the Timer Counter (TC) to achieve the two application examples described below.

Delay Measurement

To measure the delay between the channel x comparator output (OCx) and the feedback from the bridge driver of the MOSFETs (see Figure 39-6), the bit TCTS in the PWM Channel Mode Register must be at 0. This defines the comparator output of the channel x as the TC trigger source. The TIOB trigger (TC internal input) is used to start the TC; the TIOA input (from PAD) is used to capture the delay.

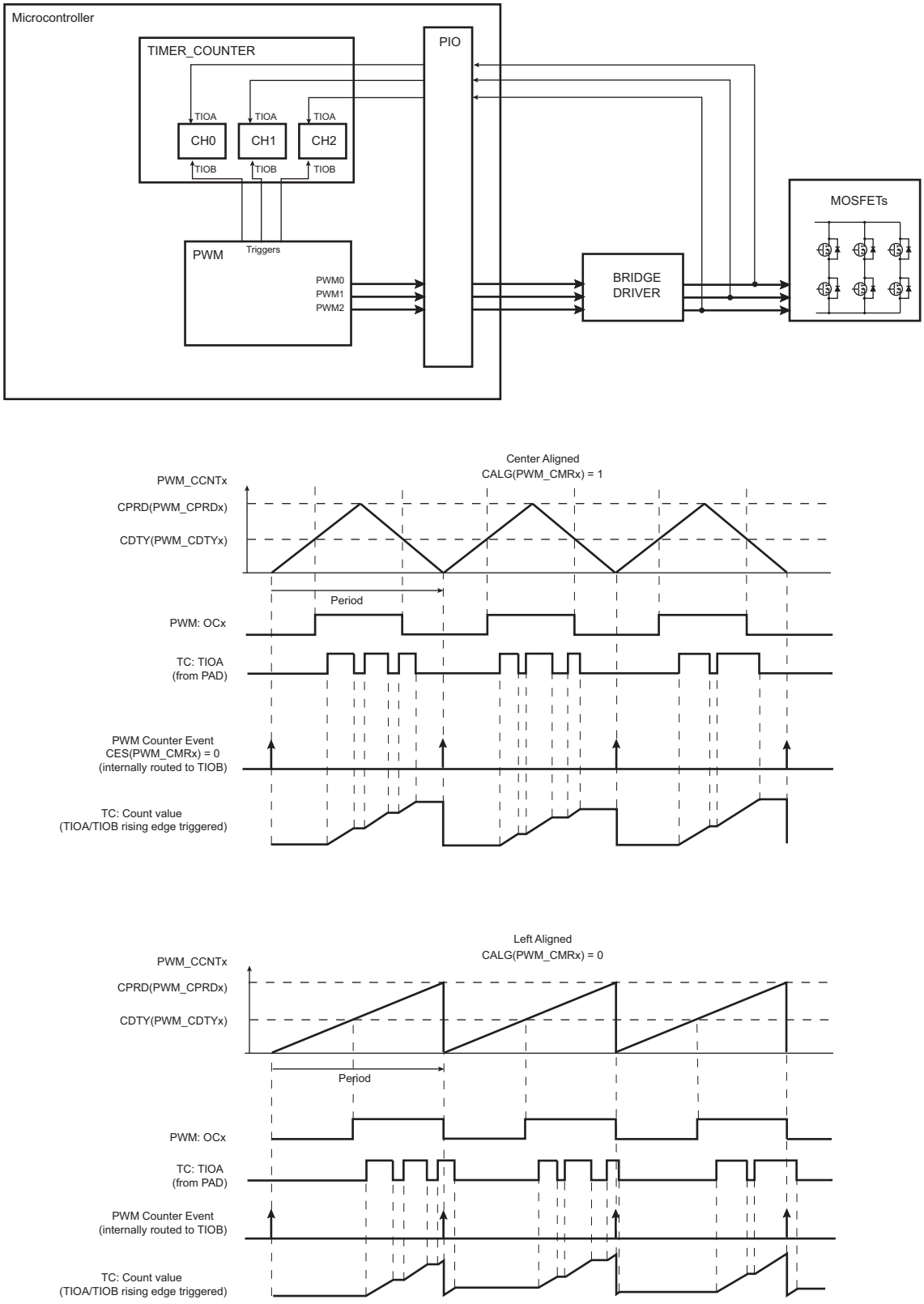
Figure 39-6. Triggering the TC: Delay Measurement



Cumulated ON Time Measurement

To measure the cumulated “ON” time of MOSFETs (see Figure 39-7), the bit TCTS of the PWM Channel Mode Register must be set to 1 to define the counter event (see Figure 39-5) as the Timer Counter trigger source.

Figure 39-7. Triggering the TC: Cumulated “ON” Time Measurement



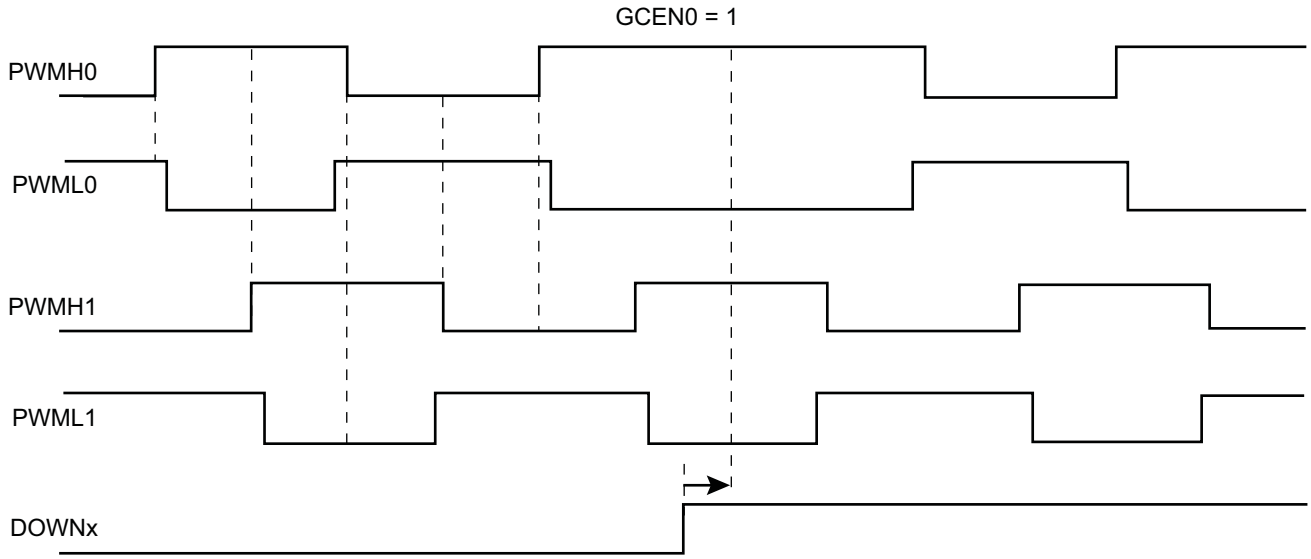
39.6.2.4 2-bit Gray Up/Down Counter for Stepper Motor

A pair of channels may provide a 2-bit gray count waveform on two 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 39-8. 2-bit Gray Up/Down Counter



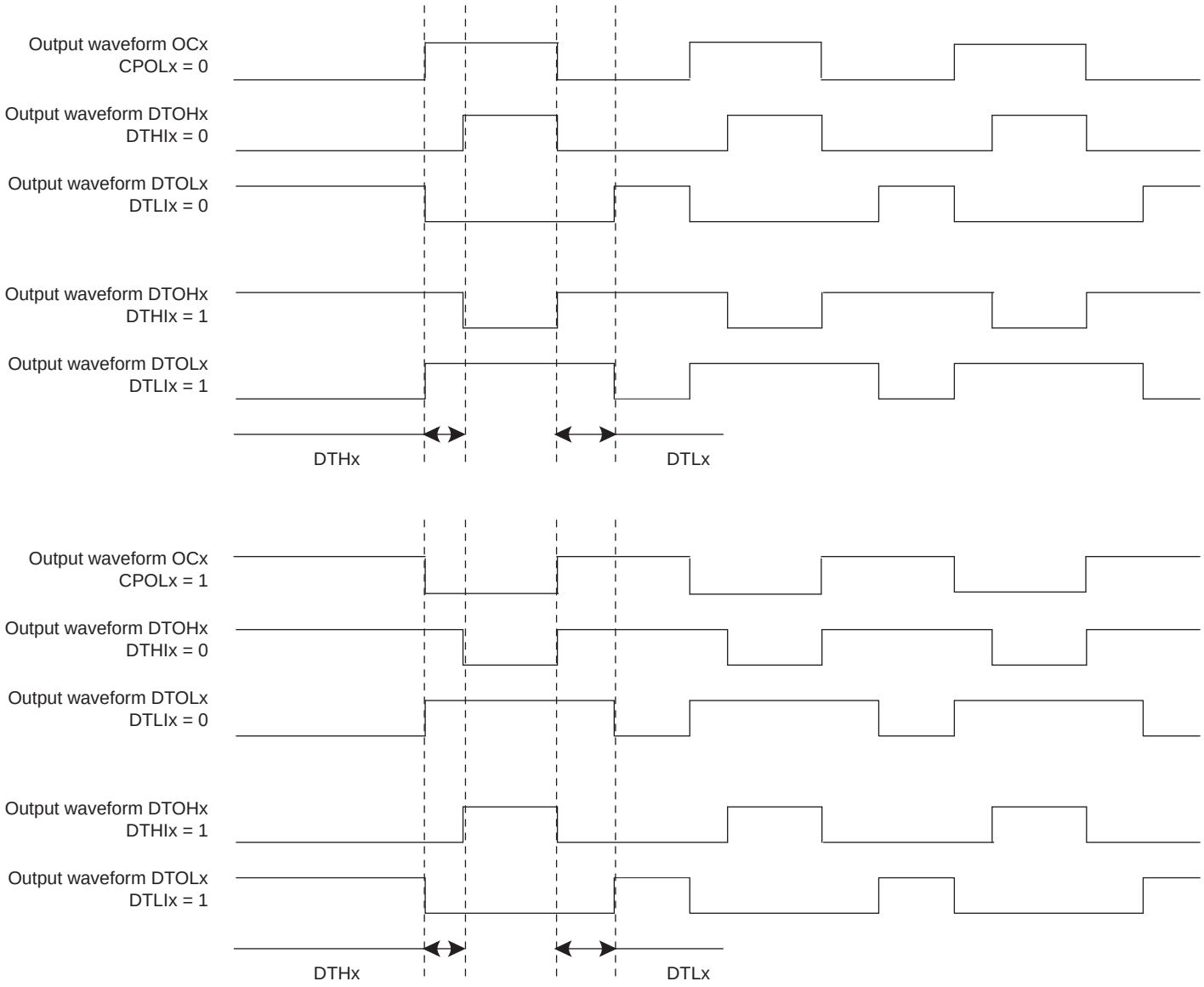
39.6.2.5 Dead-Time Generator

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). Each output 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 PWM_CMRx) is provided for each output to invert the dead-time outputs. The following figure shows the waveform of the dead-time generator.

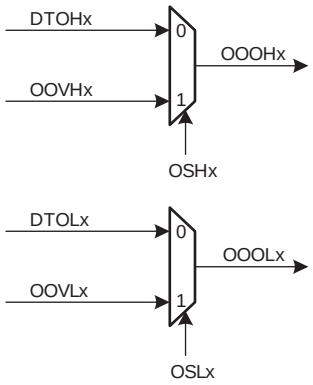
Figure 39-9. Complementary Output Waveforms



39.6.2.6 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 39-10. 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 (PWM_OSS) and PWM Output Selection Set Update Register (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 (PWM_OSC) and PWM Output Selection Clear Update Register (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.

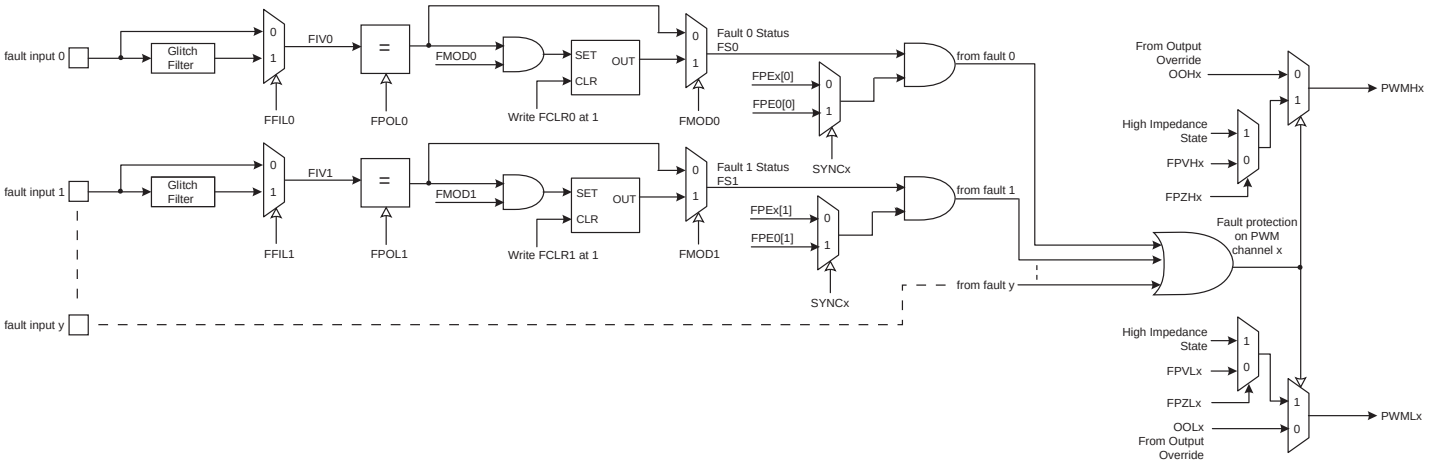
The value of the current output selection can be read in PWM_OS.

While overriding PWM outputs, the channel counters continue to run, only the PWM outputs are forced to user defined values.

39.6.2.7 Fault Protection

8 inputs provide fault protection which can force any of the PWM output pairs to a programmable value. This mechanism has priority over output overriding.

Figure 39-11. 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 or Timer Counter, 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 field 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. Refer to the corresponding peripheral documentation for details on handling fault generation.

Fault inputs may or may not be glitch-filtered depending on the FFIL field in PWM_FMR. When the filter is activated, glitches on fault inputs with a width inferior to the PWM peripheral clock 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 PWM_FMR, the fault remains active as long as the fault input is at this polarity level. If the corresponding FMOD field is set to ‘1’, the fault remains active until the fault input is no longer at this polarity level and until it is cleared by writing the corresponding bit FCLR in the PWM Fault Clear Register (PWM_FCR). In the PWM Fault Status Register (PWM_FSR), the field FIV indicates the current level of the fault inputs and the field FIS indicates whether a fault is currently active.

Each fault can be taken into account or not by the fault protection mechanism in each channel. To be taken into account in the channel x, the fault y must be enabled by the bit FPEx[y] in the PWM Fault Protection Enable registers (PWM_FPE1). However, synchronous channels (see Section 39.6.2.9 “Synchronous Channels”) do not use their own fault enable bits, but those of the channel 0 (bits FPE0[y]).

The fault protection on a channel is triggered when this channel is enabled and when any one of the faults that are enabled for this channel is active. It can be triggered even if the PWM peripheral clock is not running but only by a fault input that is not glitch-filtered.

When the fault protection is triggered on a channel, the fault protection mechanism resets the counter of this channel and forces the channel outputs to the values defined by the fields FPVHx and FPVLx in the PWM Fault Protection Value Register 1 (PWM_FPV) and fields FPZHx/FPZLx in the PWM Fault Protection Value Register 2, as shown in Table 39-3. The output forcing is made asynchronously to the channel counter.

Table 39-3. Forcing Values of PWM Outputs by Fault Protection

FPZH/Lx	FPVH/Lx	Forcing Value of PWMH/Lx
0	0	0
0	1	1
1	–	High impedance state (Hi-Z)

CAUTION:

- To prevent any unexpected activation of the status flag FSy in PWM_FSR, the FMODy bit can be set to ‘1’ only if the FPOLy bit has been previously configured to its final value.
- To prevent any 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 39.6.3 “PWM Comparison Units”) and if a fault is triggered in the channel 0, then 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.

39.6.2.8 Spread Spectrum Counter

The PWM macrocell includes a spread spectrum counter allowing the generation of a constantly varying duty cycle on the output PWM waveform (only for the channel 0). This feature may be useful to minimize electromagnetic interference or to reduce the acoustic noise of a PWM driven motor.

This is achieved by varying the effective period in a range defined by a spread spectrum value which is programmed by the field SPRD in the PWM Spread Spectrum Register (PWM_SSPR). The effective period of the output waveform is the value of the spread spectrum counter added to the programmed waveform period CPRD in the PWM Channel Period Register (PWM_CPRD0).

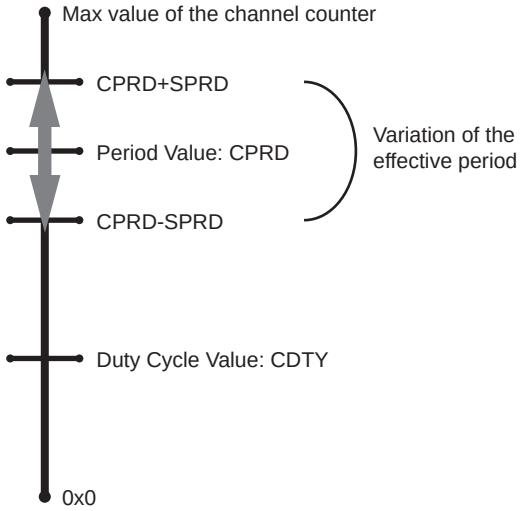
It will cause the effective period to vary from CPRD-SPRD to CPRD+SPRD. This leads to a constantly varying duty cycle on the PWM output waveform because the duty cycle value programmed is unchanged.

The value of the spread spectrum counter can change in two ways depending on the bit SPRDM in PWM_SSPR.

If SPRDM = 0, the Triangular mode is selected. The spread spectrum counter starts to count from -SPRD when the channel 0 is enabled or after reset and counts upwards at each period of the channel counter. When it reaches SPRD, it restarts to count from -SPRD again.

If SPRDM = 1, the Random mode is selected. A new random value is assigned to the spread spectrum counter at each period of the channel counter. This random value is between -SPRD and +SPRD and is uniformly distributed.

Figure 39-12. Spread Spectrum Counter



39.6.2.9 Synchronous Channels

Some channels can be linked together as synchronous channels. They have the same source clock, the same period, the same alignment and are started together. In this way, their counters are synchronized together.

The synchronous channels are defined by the SYNCx bits in the PWM Sync Channels Mode Register (PWM_SCM). Only one group of synchronous channels is allowed.

When a channel is defined as a synchronous channel, the channel 0 is also automatically defined as a synchronous channel. This is because the channel 0 counter configuration is used by all the synchronous channels.

If a channel x is defined as a synchronous channel, the fields/bits for the channel 0 are used instead of those of channel x:

- CPRE in PWM_CMR0 instead of CPRE in PWM_CMRx (same source clock)
- CPRD in PWM_CPRD0 instead of CPRD in PWM_CPRDx (same period)
- CALG in PWM_CMR0 instead of CALG in PWM_CMRx (same alignment)

Modifying the fields CPRE, CPRD and CALG of for channels with index greater than 0 has no effect on output waveforms.

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). 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 UPDM field (Update Mode) in the PWM_SCM register selects 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 processor 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”).

- 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 processor 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_SCUC register 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”).
- 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 (see “Method 3: Automatic write of duty-cycle values and automatic trigger of the update”). The user can choose to synchronize the Peripheral DMA Controller transfer request with a comparison match (see Section 39.6.3 “PWM Comparison Units”), by the fields PTRM and PTRCS in the PWM_SCM register. The DMA destination address must be configured to access only the PWM DMA Register (PWM_DMAR). The DMA buffer data structure must consist of sequentially repeated duty cycles. The number of duty cycles in each sequence corresponds to the number of synchronized channels. Duty cycles in each sequence must be ordered from the lowest to the highest channel index. The size of the duty cycle is 16 bits.

Table 39-4. Summary of the Update of Registers of Synchronous Channels

Register	UPDM = 0	UPDM = 1	UPDM = 2
Period Value (PWM_CPRDUPDx)	Write by the processor		
	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 processor		
	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 processor	Write by the processor	Write by the Peripheral DMA Controller
	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 processor	
	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 processor (respectively PWM_CPRDUPDx, PWM_CDTYUPDx and PWM_DTUPDx).

To trigger the update, the user must use the bit UPDULOCK in the PWM_SCUC register 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.

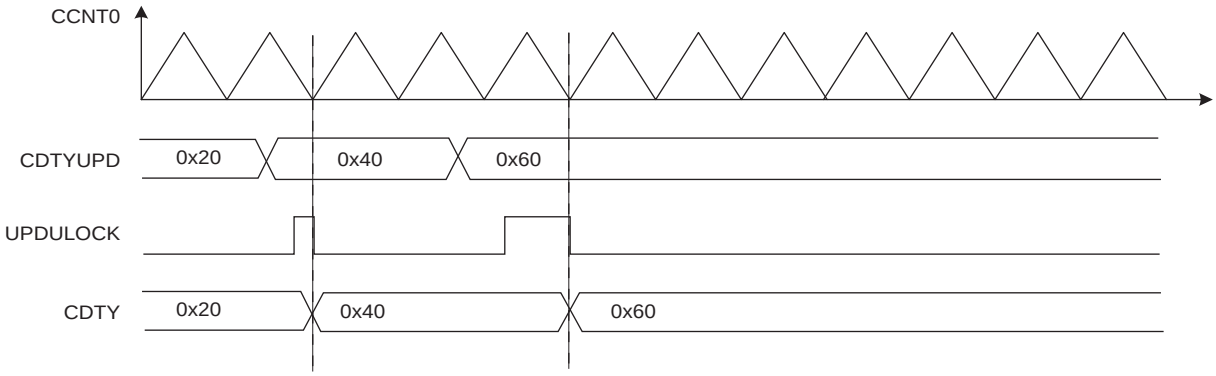
After writing the UPDULOCK bit to ‘1’, it is held at this value until the update occurs, then it is read 0.

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. When the UPDULOCK bit is reset, go to Step 4. for new values.

Figure 39-13. 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 processor (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 in the PWM_SCUC register, which updates 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_SCUP register. 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.

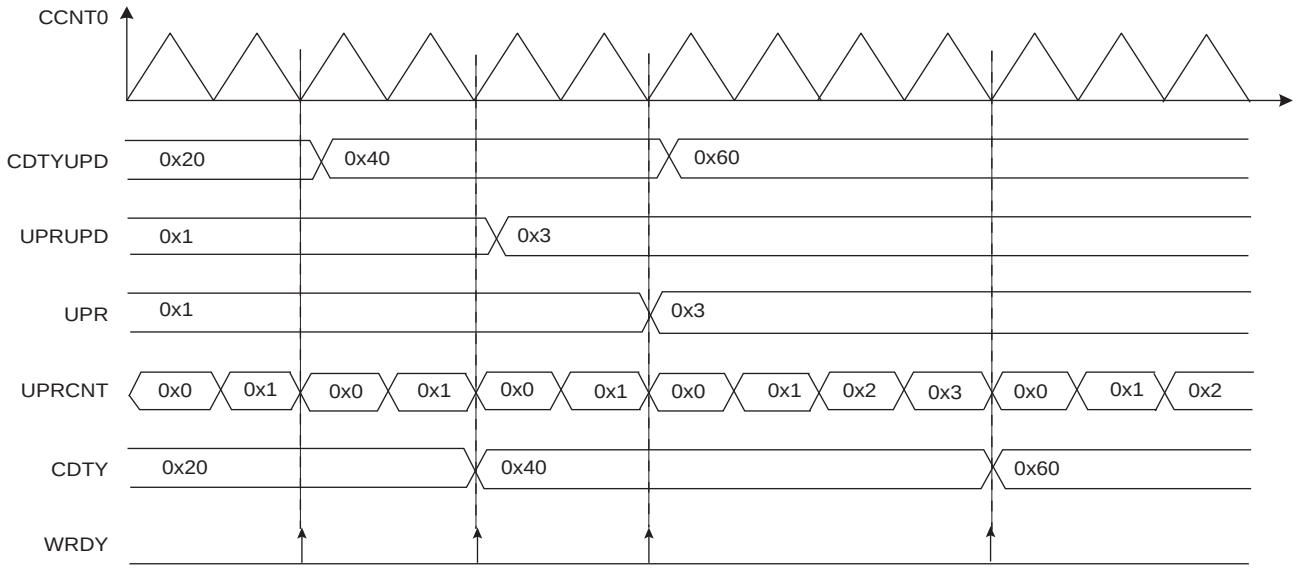
Depending on the interrupt mask in the PWM Interrupt Mask Register 2 (PWM_IMR2), an interrupt can be generated by these flags.

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 PWM_ISR2.
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 39-14. 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. 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 processor (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_SCUP register. The PWM controller waits UPR+1 periods of synchronous channels before updating automatically the duty values and the update period value.

Using the Peripheral DMA Controller 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 Peripheral DMA Controller 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 Peripheral DMA Controller 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 Peripheral DMA Controller transfer is reported in 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 PWM_ISR2 is read. The user can choose to synchronize the WRDY flag and the Peripheral DMA Controller transfer request with a comparison match (see Section 39.6.3 “PWM Comparison Units”), by the fields PTRM and PTRCS in the PWM_SCM register.
- ENDTX (not relevant if DMA is used): this flag is set to ‘1’ when a PDC transfer is completed
- TXBUFE (not relevant if DMA is used): 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 Peripheral DMA Controller. It is reset to ‘0’ when PWM_ISR2 is read.

Depending on the interrupt mask in PWM_IMR2, 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 Peripheral DMA Controller 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 Peripheral DMA Controller transfer settings for the duty-cycle values and enable it in the Peripheral DMA Controller 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 PWM_ISR2, else go to Step 12.
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.If DMA is used: Wait for the DMA status flag indicating that the buffer transfer is complete. If the transfer has ended, define a new DMA transfer for new duty-cycle values. Go to Step 5. If PDC is used: 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 39-15. Method 3 (UPDM = 2 and PTRM = 0)

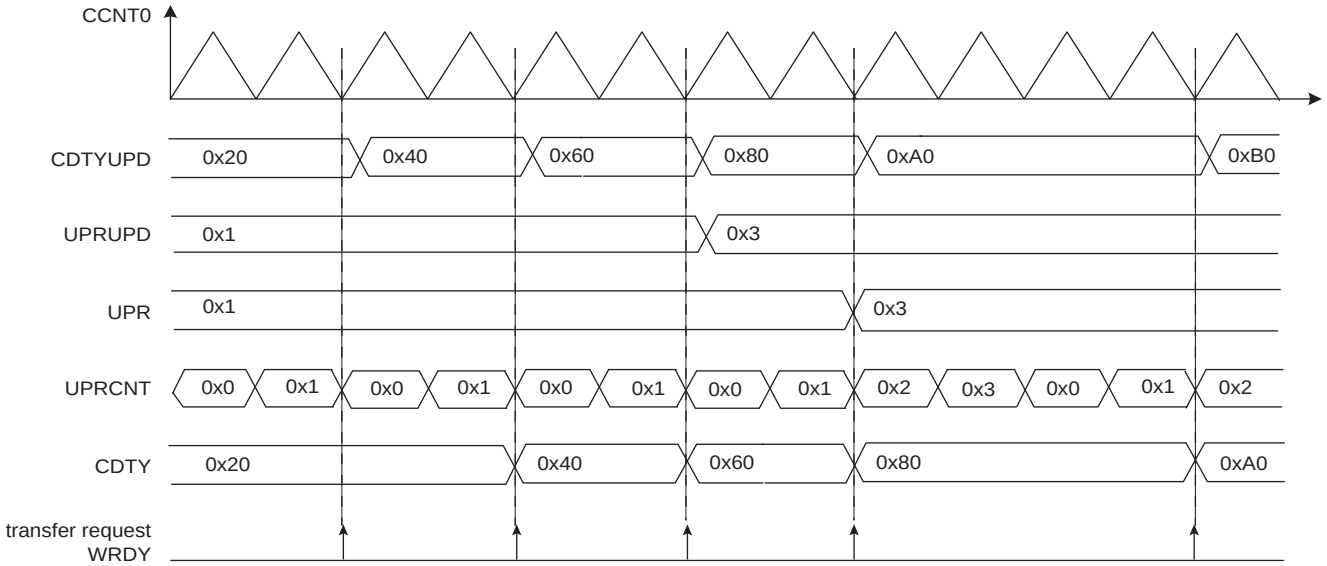
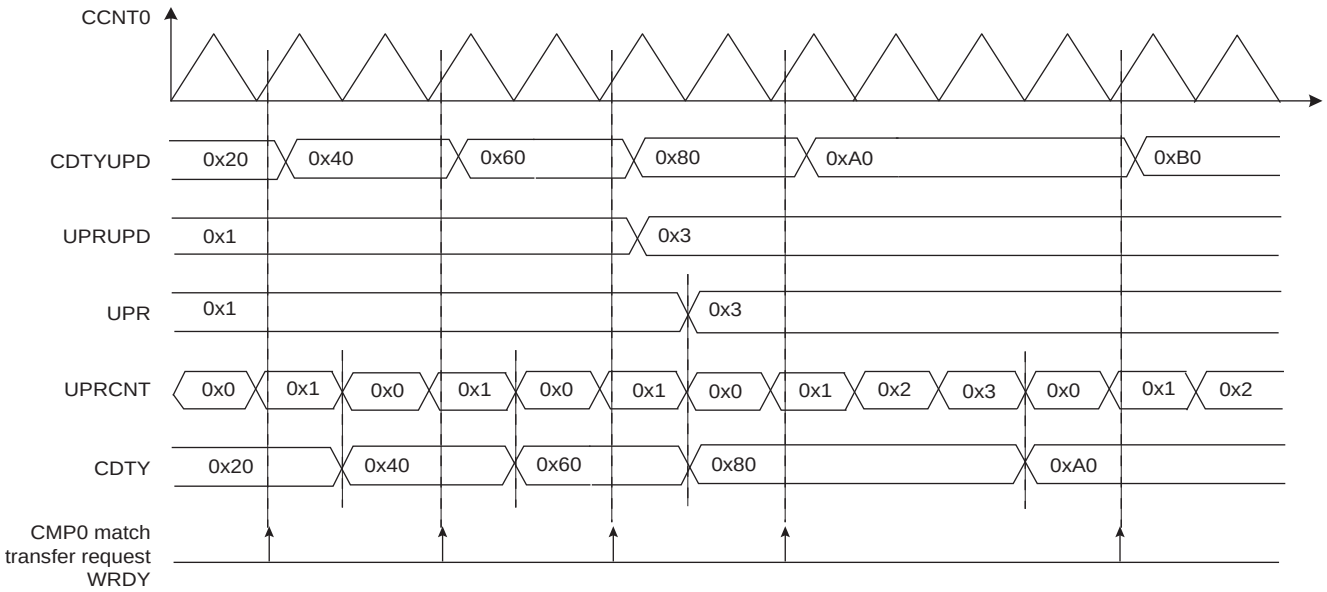


Figure 39-16. Method 3 (UPDM = 2 and PTRM = 1 and PTRCS = 0)



39.6.2.10 Update Time for Double-Buffering Registers

All channels integrate a double-buffering system in order to prevent an unexpected output waveform while modifying the period, the spread spectrum value, the polarity, the duty-cycle, the dead-times, the output override, and the synchronous channels update period.

This double-buffering system comprises the following update registers:

- PWM Sync Channels Update Period Update Register
- PWM Output Selection Set Update Register
- PWM Output Selection Clear Update Register
- PWM Spread Spectrum Update Register
- PWM Channel Duty Cycle Update Register
- PWM Channel Period Update Register

- PWM Channel Dead Time Update Register
- PWM Channel Mode Update Register

When one of these update registers is written to, the write is stored, but the values are updated only at the next PWM period border. In Left-aligned mode (CALG = 0), the update occurs when the channel counter reaches the period value CPRD. In Center-aligned mode, the update occurs when the channel counter value is decremented and reaches the 0 value.

In Center-aligned mode, it is possible to trigger the update of the polarity and the duty-cycle at the next half period border. This mode concerns the following update registers:

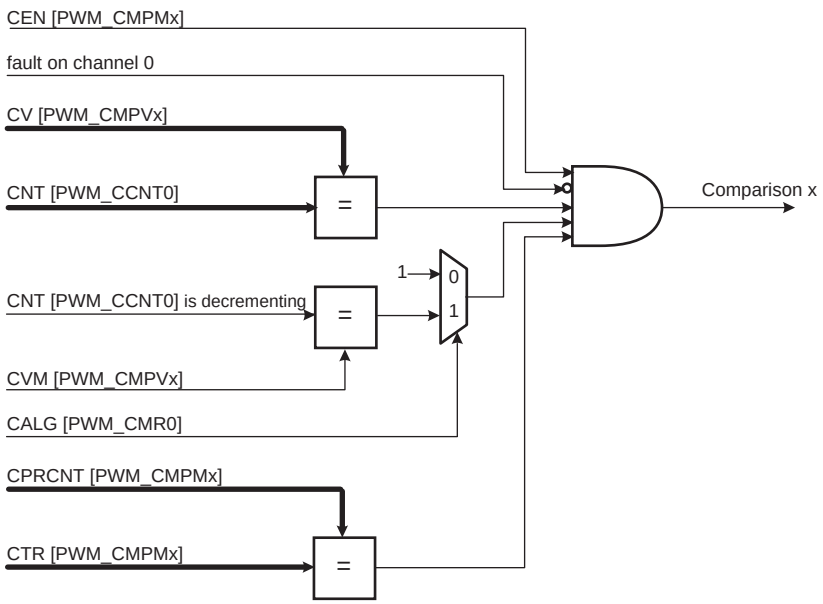
- PWM Channel Duty Cycle Update Register
- PWM Channel Mode Update Register

The update occurs at the first half period following the write of the update register (either when the channel counter value is incrementing and reaches the period value CPRD, or when the channel counter value is decrementing and reaches the 0 value). To activate this mode, the user must write a one to the bit UPDS in the PWM Channel Mode Register.

39.6.3 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 39.6.2.9 “Synchronous Channels”). These comparisons are intended to generate pulses on the event lines (used to synchronize ADC, see Section 39.6.4 “PWM Event Lines”), to generate software interrupts and to trigger Peripheral DMA Controller transfer requests for the synchronous channels (see “Method 3: Automatic write of duty-cycle values and automatic trigger of the update”).

Figure 39-17. 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 39.6.2.7 “Fault Protection”).

The user can define the periodicity of the comparison x by the fields CTR and CPR in PWM_CMPMx. 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.

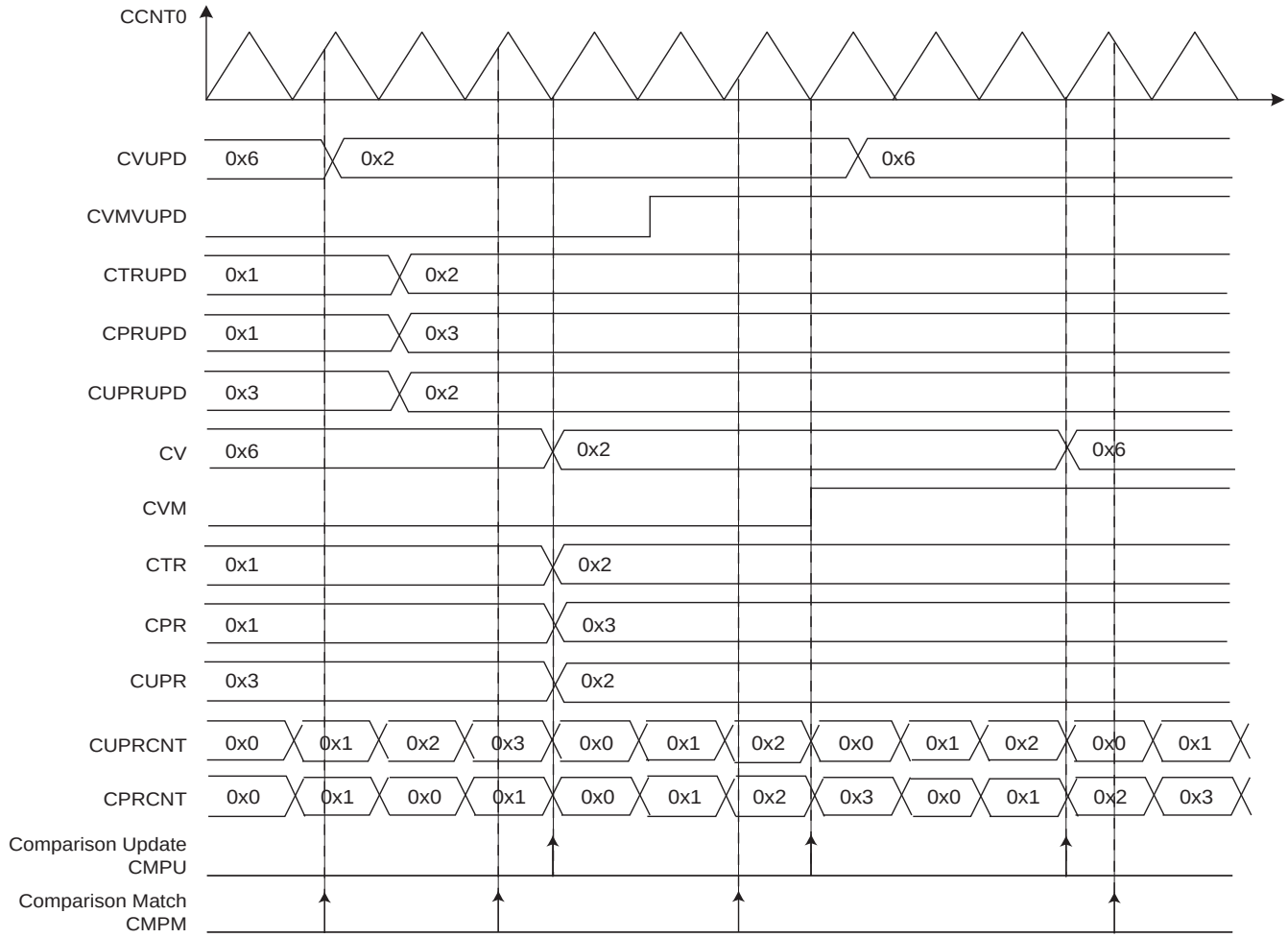
The comparison x configuration can be modified while the channel 0 is enabled by using the PWM Comparison x Mode Update Register (PWM_CMPMUPDx registers for the comparison x). In the same way, the comparison x value can be modified while the channel 0 is enabled by using the PWM Comparison x Value Update 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 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: The write of PWM_CMPVUPDx must be followed by a write of 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 39-18. Comparison Waveform



39.6.4 PWM Event Lines

The PWM provides 2 independent event lines intended to trigger actions in other peripherals (e.g., for the Analog-to-Digital Converter (ADC)).

A pulse (one cycle of the peripheral clock) 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).

An example of event generation is provided in Figure 39-20.

Figure 39-19. Event Line Block Diagram

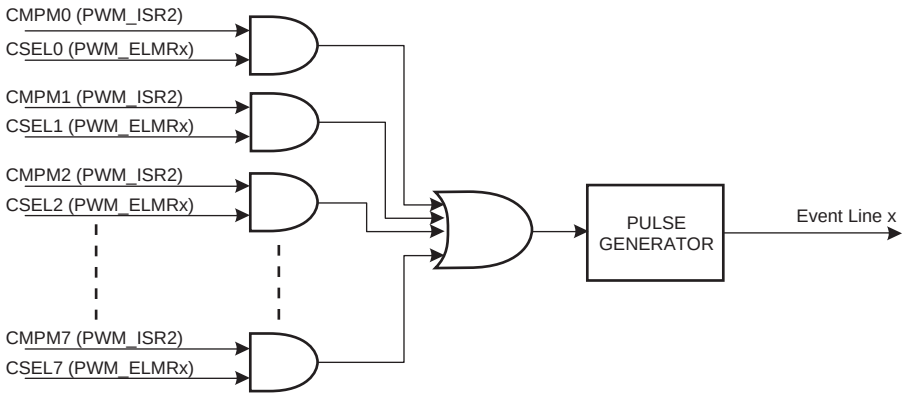
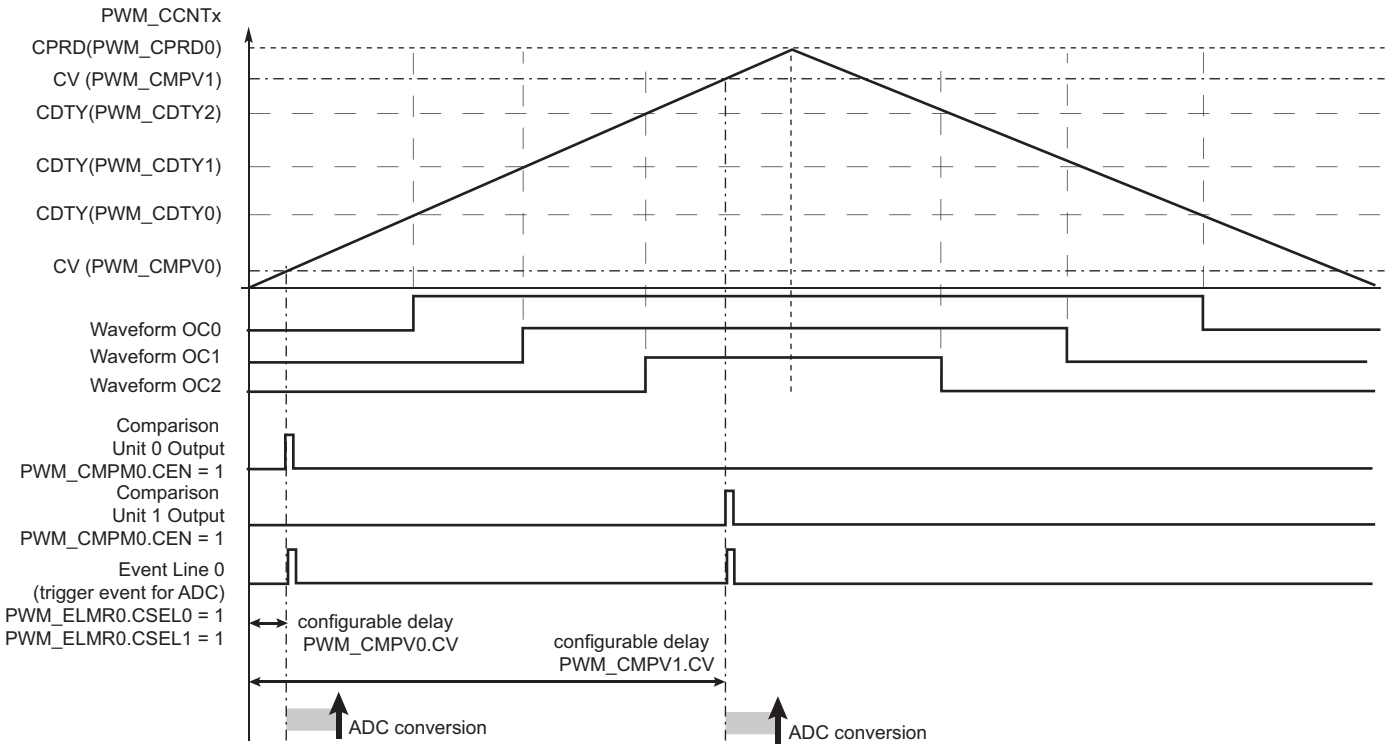


Figure 39-20. Event Line Generation Waveform (Example)



39.6.5 PWM Controller Operations

39.6.5.1 Initialization

Before enabling the channels, they must be configured by the software application as described below:

- Unlock User Interface by writing the WPCMD field in PWM_WPCR.
- 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 PWM_CMRx)
- Configuration of the waveform alignment for each channel (CALG field in PWM_CMRx)
- Selection of the counter event selection (if CALG = 1) for each channel (CES field in PWM_CMRx)
- Configuration of the output waveform polarity for each channel (CPOL bit in PWM_CMRx)

- 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 PWM_CMRx). 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 Peripheral DMA Controller transfer request are set (PTRM and PTRCS in the PWM_SCM register)
- Configuration of the Update mode (UPDM in PWM_SCM register)
- Configuration of the update period (UPR in PWM_SCUP register) if needed
- Configuration of the comparisons (PWM_CMPVx and PWM_CMPMx)
- Configuration of the event lines (PWM_ELMRx)
- Configuration of the fault inputs polarity (FPOL in PWM_FMR)
- Configuration of the fault protection (FMOD and FFIL in PWM_FMR, PWM_FPV and PWM_FPE1)
- Enable of the interrupts (writing CHIDx and FCHIDx in PWM_IER1, and writing WRDY, ENDTX, TXBUFE, UNRE, CMPMx and CMPUx in PWM_IER2)
- Enable of the PWM channels (writing CHIDx in the PWM_ENA register)

39.6.5.2 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) helps the user select the appropriate clock. 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. The resulting duty-cycle quantum cannot be lower than 1/15 of the PWM period.

39.6.5.3 Changing the Duty-Cycle, the Period and the Dead-Times

It is possible to modulate the output waveform duty-cycle, period and dead-times.

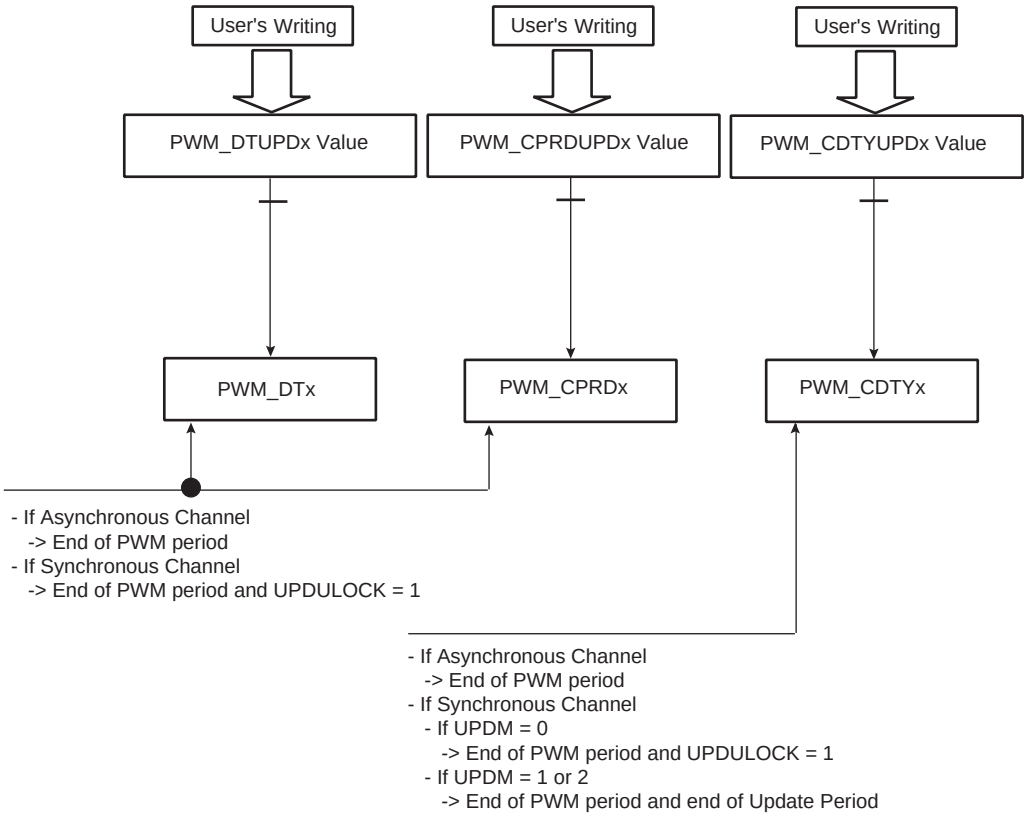
To prevent unexpected output waveform, the user must use the PWM Channel Duty Cycle Update Register (PWM_CDTYUPDx), the PWM Channel Period Update Register (PWM_CPRDUPDx) and the PWM Channel Dead Time Update Register (PWM_DTUPDx) to change waveform parameters while the channel is still enabled.

- 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) 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 39-21. Synchronized Period, Duty-Cycle and Dead-Time Update



39.6.5.4 Changing the Update Period of Synchronous Channels

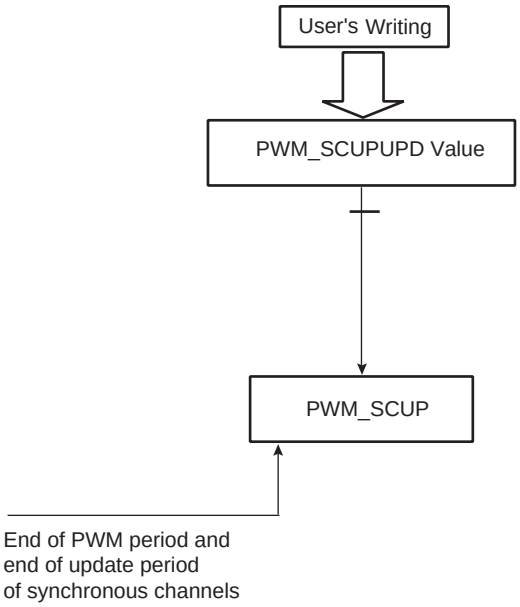
It is possible to change the update period of synchronous channels while they are enabled. See "Method 2: Manual write of duty-cycle values and automatic trigger of the update" and "Method 3: Automatic write of duty-cycle values and automatic trigger of the update" .

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_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 39-22. Synchronized Update of Update Period Value of Synchronous Channels



39.6.5.5 Changing the Comparison Value and the Comparison Configuration

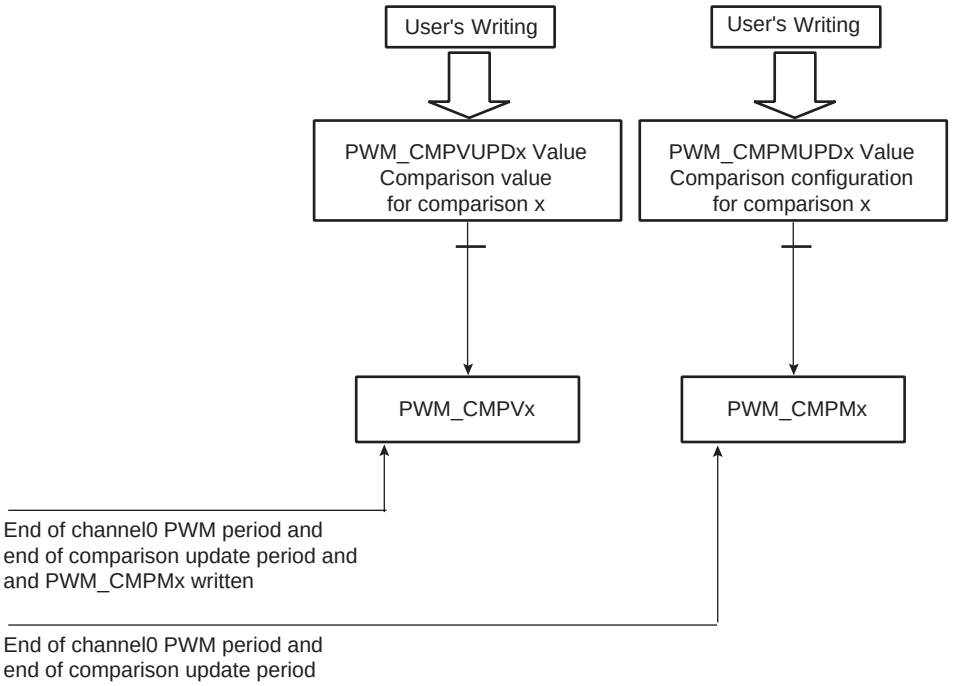
It is possible to change the comparison values and the comparison configurations while the channel 0 is enabled (see Section 39.6.3 “PWM Comparison Units”).

To prevent unexpected comparison match, the user must use the PWM Comparison x Value Update Register (PWM_CMPVUPDx) and the PWM Comparison x Mode Update Register (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: 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 39-23. Synchronized Update of Comparison Values and Configurations



39.6.5.6 Interrupt Sources

Depending on the interrupt mask in PWM_IMR1 and PWM_IMR2, an interrupt can be generated at the end of the corresponding channel period (CHIDx in the PWM Interrupt Status Register 1 (PWM_ISR1)), after a fault event (FCHIDx in PWM_ISR1), after a comparison match (CMPMx in PWM_ISR2), after a comparison update (CMPUx in PWM_ISR2) or according to the Transfer mode of the synchronous channels (WRDY, ENDTX, TXBUFE and UNRE in PWM_ISR2).

If the interrupt is generated by the flags CHIDx or FCHIDx, the interrupt remains active until a read operation in PWM_ISR1 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 PWM_ISR2 occurs.

A channel interrupt is enabled by setting the corresponding bit in PWM_IER1 and PWM_IER2. A channel interrupt is disabled by setting the corresponding bit in PWM_IDR1 and PWM_IDR2.

39.6.6 Register Write Protection

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 Protection Control Register (PWM_WPCR). They are divided into six groups:

- Register group 0:
  - PWM Clock Register
- Register group 1:
  - PWM Disable Register
- Register group 2:
  - PWM Sync Channels Mode Register
  - PWM Channel Mode Register
  - PWM Stepper Motor Mode Register
  - PWM Channel Mode Update Register
- Register group 3:
  - PWM Spread Spectrum Register
  - PWM Spread Spectrum Update Register
  - PWM Channel Period Register
  - PWM Channel Period Update Register
- Register group 4:
  - PWM Channel Dead Time Register
  - PWM Channel Dead Time Update Register
- Register group 5:
  - PWM Fault Mode Register
  - PWM Fault Protection Value Register 1

There are two types of write protection:

- SW write protection—can be enabled or disabled by software
- HW write protection—can be enabled by software but only disabled by a hardware reset of the PWM controller

Both types of write protection can be applied independently to a particular register group by means of the WPCMD and WPRGx fields in PWM_WPCR. If at least one type of write protection is active, the register group is write-protected. The value of field WPCMD defines the action to be performed:

- 0: Disables SW write protection of the register groups of which the bit WPRGx is at ‘1’
- 1: Enables SW write protection of the register groups of which the bit WPRGx is at ‘1’
- 2: Enables HW write protection of the register groups of which the bit WPRGx is at ‘1’

At any time, the user can determine whether SW or HW write protection is active in a particular register group by the fields WPSWS and WPHWS in the PWM Write Protection Status Register (PWM_WPSR).

If a write access to a write-protected register is detected, the WPVS flag in PWM_WPSR is set and the field WPVSRC indicates the register in which the write access has been attempted.

The WPVS and WPVSRC fields are automatically cleared after reading PWM_WPSR.

39.7 Pulse Width Modulation Controller (PWM) User Interface

Table 39-5. 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	PWM DMA Register	PWM_DMAR	Write-only	–
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	–
0x34	PWM Interrupt Enable Register 2	PWM_IER2	Write-only	–
0x38	PWM Interrupt Disable Register 2	PWM_IDR2	Write-only	–
0x3C	PWM Interrupt Mask Register 2	PWM_IMR2	Read-only	0x0
0x40	PWM Interrupt Status Register 2	PWM_ISR2	Read-only	0x0
0x44	PWM Output Override Value Register	PWM_OOV	Read/Write	0x0
0x48	PWM Output Selection Register	PWM_OS	Read/Write	0x0
0x4C	PWM Output Selection Set Register	PWM_OSS	Write-only	–
0x50	PWM Output Selection Clear Register	PWM_OSC	Write-only	–
0x54	PWM Output Selection Set Update Register	PWM_OSSUPD	Write-only	–
0x58	PWM Output Selection Clear Update Register	PWM_OSCUPD	Write-only	–
0x5C	PWM Fault Mode Register	PWM_FMR	Read/Write	0x0
0x60	PWM Fault Status Register	PWM_FSR	Read-only	0x0
0x64	PWM Fault Clear Register	PWM_FCR	Write-only	–
0x68	PWM Fault Protection Value Register 1	PWM_FPV1	Read/Write	0x0
0x6C	PWM Fault Protection Enable Register	PWM_FPE	Read/Write	0x0
0x70–0x78	Reserved	–	–	–
0x7C	PWM Event Line 0 Mode Register	PWM_ELMR0	Read/Write	0x0
0x80	PWM Event Line 1 Mode Register	PWM_ELMR1	Read/Write	0x0
0x84–0x9C	Reserved	–	–	–
0xA0	PWM Spread Spectrum Register	PWM_SSPR	Read/Write	0x0
0xA4	PWM Spread Spectrum Update Register	PWM_SSPUP	Write-only	–
0xA8–0xAC	Reserved	–	–	–

Table 39-5. Register Mapping (Continued)

Offset	Register	Name	Access	Reset
0xB0	PWM Stepper Motor Mode Register	PWM_SMMR	Read/Write	0x0
0xB4–0xBC	Reserved	–	–	–
0xC0	PWM Fault Protection Value 2 Register	PWM_FPV2	Read/Write	0x003F_003F
0xC4–0xE0	Reserved	–	–	–
0xE4	PWM Write Protection Control Register	PWM_WPCR	Write-only	–
0xE8	PWM Write Protection Status Register	PWM_WPSR	Read-only	0x0
0xEC–0xFC	Reserved	–	–	–
0x100–0x128	Reserved for PDC registers	–	–	–
0x12C	Reserved	–	–	–
0x130	PWM Comparison 0 Value Register	PWM_CMPV0	Read/Write	0x0
0x134	PWM Comparison 0 Value Update Register	PWM_CMPVUPD0	Write-only	–
0x138	PWM Comparison 0 Mode Register	PWM_CMPM0	Read/Write	0x0
0x13C	PWM Comparison 0 Mode Update Register	PWM_CMPMUPD0	Write-only	–
0x140	PWM Comparison 1 Value Register	PWM_CMPV1	Read/Write	0x0
0x144	PWM Comparison 1 Value Update Register	PWM_CMPVUPD1	Write-only	–
0x148	PWM Comparison 1 Mode Register	PWM_CMPM1	Read/Write	0x0
0x14C	PWM Comparison 1 Mode Update Register	PWM_CMPMUPD1	Write-only	–
0x150	PWM Comparison 2 Value Register	PWM_CMPV2	Read/Write	0x0
0x154	PWM Comparison 2 Value Update Register	PWM_CMPVUPD2	Write-only	–
0x158	PWM Comparison 2 Mode Register	PWM_CMPM2	Read/Write	0x0
0x15C	PWM Comparison 2 Mode Update Register	PWM_CMPMUPD2	Write-only	–
0x160	PWM Comparison 3 Value Register	PWM_CMPV3	Read/Write	0x0
0x164	PWM Comparison 3 Value Update Register	PWM_CMPVUPD3	Write-only	–
0x168	PWM Comparison 3 Mode Register	PWM_CMPM3	Read/Write	0x0
0x16C	PWM Comparison 3 Mode Update Register	PWM_CMPMUPD3	Write-only	–
0x170	PWM Comparison 4 Value Register	PWM_CMPV4	Read/Write	0x0
0x174	PWM Comparison 4 Value Update Register	PWM_CMPVUPD4	Write-only	–
0x178	PWM Comparison 4 Mode Register	PWM_CMPM4	Read/Write	0x0
0x17C	PWM Comparison 4 Mode Update Register	PWM_CMPMUPD4	Write-only	–
0x180	PWM Comparison 5 Value Register	PWM_CMPV5	Read/Write	0x0
0x184	PWM Comparison 5 Value Update Register	PWM_CMPVUPD5	Write-only	–
0x188	PWM Comparison 5 Mode Register	PWM_CMPM5	Read/Write	0x0
0x18C	PWM Comparison 5 Mode Update Register	PWM_CMPMUPD5	Write-only	–
0x190	PWM Comparison 6 Value Register	PWM_CMPV6	Read/Write	0x0
0x194	PWM Comparison 6 Value Update Register	PWM_CMPVUPD6	Write-only	–
0x198	PWM Comparison 6 Mode Register	PWM_CMPM6	Read/Write	0x0
0x19C	PWM Comparison 6 Mode Update Register	PWM_CMPMUPD6	Write-only	–

Table 39-5. Register Mapping (Continued)

Offset	Register	Name	Access	Reset
0x1A0	PWM Comparison 7 Value Register	PWM_CMPV7	Read/Write	0x0
0x1A4	PWM Comparison 7 Value Update Register	PWM_CMPVUPD7	Write-only	–
0x1A8	PWM Comparison 7 Mode Register	PWM_CMPM7	Read/Write	0x0
0x1AC	PWM Comparison 7 Mode Update Register	PWM_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	–
0x400 + ch_num * 0x20 + 0x00	PWM Channel Mode Update Register(1)	PWM_CMUPD	Write-only	–

Notes: 1. Some registers are indexed with “ch_num” index ranging from 0 to 3.

39.7.1 PWM Clock Register

Name: PWM_CLK

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 bits WPSWS0 and WPHWS0 are cleared in the PWM Write Protection Status Register.

- **DIVA: CLKA Divide Factor**

Value	Name	Description
0	CLKA_POFF	CLKA clock is turned off
1	PREA	CLKA clock is clock selected by PREA
2–255	PREA_DIV	CLKA clock is clock selected by PREA divided by DIVA factor

- **DIVB: CLKB Divide Factor**

Value	Name	Description
0	CLKB_POFF	CLKB clock is turned off
1	PREB	CLKB clock is clock selected by PREB
2–255	PREB_DIV	CLKB clock is clock selected by PREB divided by DIVB factor

- **PREA: CLKA Source Clock Selection**

Value	Name	Description
0	CLK	Peripheral clock
1	CLK_DIV2	Peripheral clock/2
2	CLK_DIV4	Peripheral clock/4
3	CLK_DIV8	Peripheral clock/8
4	CLK_DIV16	Peripheral clock/16
5	CLK_DIV32	Peripheral clock/32
6	CLK_DIV64	Peripheral clock/64
7	CLK_DIV128	Peripheral clock/128
8	CLK_DIV256	Peripheral clock/256
9	CLK_DIV512	Peripheral clock/512
10	CLK_DIV1024	Peripheral clock/1024
Other	–	Reserved

- **PREB: CLKB Source Clock Selection**

Value	Name	Description
0	CLK	Peripheral clock
1	CLK_DIV2	Peripheral clock/2
2	CLK_DIV4	Peripheral clock/4
3	CLK_DIV8	Peripheral clock/8
4	CLK_DIV16	Peripheral clock/16
5	CLK_DIV32	Peripheral clock/32
6	CLK_DIV64	Peripheral clock/64
7	CLK_DIV128	Peripheral clock/128
8	CLK_DIV256	Peripheral clock/256
9	CLK_DIV512	Peripheral clock/512
10	CLK_DIV1024	Peripheral clock/1024
Other	–	Reserved

39.7.2 PWM Enable Register

Name: PWM_ENA

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Channel ID**

0: No effect.

1: Enable PWM output for channel x.

39.7.3 PWM Disable Register

Name: PWM_DIS

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	CHID3	CHID2	CHID1	CHID0

This register can only be written if bits WPSWS1 and WPHWS1 are cleared in the PWM Write Protection Status Register.

- **CHIDx: Channel ID**

0: No effect.

1: Disable PWM output for channel x.

39.7.4 PWM Status Register

Name: PWM_SR

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Channel ID**

0: PWM output for channel x is disabled.

1: PWM output for channel x is enabled.

39.7.5 PWM Interrupt Enable Register 1

Name: PWM_IER1

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	FCHID3	FCHID2	FCHID1	FCHID0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Counter Event on Channel x Interrupt Enable**
- **FCHIDx: Fault Protection Trigger on Channel x Interrupt Enable**

39.7.6 PWM Interrupt Disable Register 1

Name: PWM_IDR1

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	FCHID3	FCHID2	FCHID1	FCHID0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Counter Event on Channel x Interrupt Disable**
- **FCHIDx: Fault Protection Trigger on Channel x Interrupt Disable**

39.7.7 PWM Interrupt Mask Register 1

Name: PWM_IMR1

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	FCHID3	FCHID2	FCHID1	FCHID0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Counter Event on Channel x Interrupt Mask**
- **FCHIDx: Fault Protection Trigger on Channel x Interrupt Mask**

39.7.8 PWM Interrupt Status Register 1

Name: PWM_ISR1

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	FCHID3	FCHID2	FCHID1	FCHID0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Counter Event on Channel x**

0: No new counter event has occurred since the last read of PWM_ISR1.

1: At least one counter event has occurred since the last read of PWM_ISR1.

- **FCHIDx: Fault Protection Trigger on Channel x**

0: No new trigger of the fault protection since the last read of PWM_ISR1.

1: At least one trigger of the fault protection since the last read of PWM_ISR1.

Note: Reading PWM_ISR1 automatically clears CHIDx and FCHIDx flags.

39.7.9 PWM Sync Channels Mode Register

Name: PWM_SCM

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
–	–	–	–	SYNC3	SYNC2	SYNC1	SYNC0

This register can only be written if bits WPSWS2 and WPHWS2 are cleared in the PWM Write Protection Status Register.

- **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 Peripheral DMA Controller and automatic update of synchronous channels(2)

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: Peripheral DMA Controller Transfer Request Mode**

UPDM	PTRM	WRDY Flag and Peripheral DMA Controller Transfer Request
0	x	The WRDY flag in PWM Interrupt Status Register 2 and the transfer request are never set to ‘1’.
1	x	The WRDY flag in PWM Interrupt Status Register 2 is set to ‘1’ as soon as the update period is elapsed, the Peripheral DMA Controller transfer request is never set to ‘1’.
2	0	The WRDY flag in PWM Interrupt Status Register 2 and the transfer request are set to ‘1’ as soon as the update period is elapsed.
	1	The WRDY flag in PWM Interrupt Status Register 2 and the transfer request are set to ‘1’ as soon as the selected comparison matches.

- **PTRCS: Peripheral DMA Controller Transfer Request Comparison Selection**

Selection of the comparison used to set the flag WRDY and the corresponding Peripheral DMA Controller transfer request.

39.7.10 PWM DMA Register

Name: PWM_DMAR

Access: Write- only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
DMADUTY							

15	14	13	12	11	10	9	8
DMADUTY							

7	6	5	4	3	2	1	0
DMADUTY							

Only the first 16 bits (channel counter size) are significant.

- **DMADUTY: Duty-Cycle Holding Register for DMA Access**

Each write access to PWM_DMAR sequentially updates the CDTY field of PWM_CDTYx with DMADUTY (only for channel configured as synchronous). See “Method 3: Automatic write of duty-cycle values and automatic trigger of the update” .

39.7.11 PWM Sync Channels Update Control Register

Name: PWM_SCUC

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, 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.

39.7.12 PWM Sync Channels Update Period Register

Name: PWM_SCUP

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). This time is equal to UPR+1 periods of the synchronous channels.

- **UPRCNT: Update Period Counter**

Reports the value of the update period counter.

39.7.13 PWM Sync Channels Update Period Update Register

Name: PWM_SCUPUPD

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). This time is equal to UPR+1 periods of the synchronous channels.

39.7.14 PWM Interrupt Enable Register 2

Name: PWM_IER2

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**

39.7.15 PWM Interrupt Disable Register 2

Name: PWM_IDR2

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**

39.7.16 PWM Interrupt Mask Register 2

Name: PWM_IMR2

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**

39.7.17 PWM Interrupt Status Register 2

Name: PWM_ISR2

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.

39.7.18 PWM Output Override Value Register

Name: PWM_OOV

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	OOVL3	OOVL2	OOVL1	OOVL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	OOVH3	OOVH2	OOVH1	OOVH0

- **OOVHx: Output Override Value for PWMH output of the channel x**

0: Override value is 0 for PWMH output of channel x.

1: Override value is 1 for PWMH output of channel x.

- **OOVLx: Output Override Value for PWML output of the channel x**

0: Override value is 0 for PWML output of channel x.

1: Override value is 1 for PWML output of channel x.

39.7.19 PWM Output Selection Register

Name: PWM_OS

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	OSL3	OSL2	OSL1	OSL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	OSH3	OSH2	OSH1	OSH0

- **OSHx: Output Selection for PWMH output of the channel x**

0: Dead-time generator output DTOHx selected as PWMH output of channel x.

1: Output override value OOVHx selected as PWMH output of channel x.

- **OSLx: Output Selection for PWML output of the channel x**

0: Dead-time generator output DTOLx selected as PWML output of channel x.

1: Output override value OOVLx selected as PWML output of channel x.

39.7.20 PWM Output Selection Set Register

Name: PWM_OSS

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	OSSL3	OSSL2	OSSL1	OSSL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	OSSH3	OSSH2	OSSH1	OSSH0

- **OSSHx: Output Selection Set for PWMH output of the channel x**

0: No effect.

1: Output override value OOVHx selected as PWMH output of channel x.

- **OSSLx: Output Selection Set for PWML output of the channel x**

0: No effect.

1: Output override value OOVLx selected as PWML output of channel x.

39.7.21 PWM Output Selection Clear Register

Name: PWM_OSC

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	OSCL3	OSCL2	OSCL1	OSCL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	OSCH3	OSCH2	OSCH1	OSCH0

- **OSCHx: Output Selection Clear for PWMH output of the channel x**

0: No effect.

1: Dead-time generator output DTOHx selected as PWMH output of channel x.

- **OSCLx: Output Selection Clear for PWML output of the channel x**

0: No effect.

1: Dead-time generator output DTOLx selected as PWML output of channel x.

39.7.22 PWM Output Selection Set Update Register

Name: PWM_OSSUPD

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	OSSUPL3	OSSUPL2	OSSUPL1	OSSUPL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	OSSUPH3	OSSUPH2	OSSUPH1	OSSUPH0

- **OSSUPHx: Output Selection Set for PWMH output of the channel x**

0: No effect.

1: Output override value OOVHx selected as PWMH output of channel x at the beginning of the next channel x PWM period.

- **OSSUPLx: Output Selection Set for PWML output of the channel x**

0: No effect.

1: Output override value OOVLx selected as PWML output of channel x at the beginning of the next channel x PWM period.

39.7.23 PWM Output Selection Clear Update Register

Name: PWM_OSCUPD

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	OSCUPL3	OSCUPL2	OSCUPL1	OSCUPL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	OSCUPH3	OSCUPH2	OSCUPH1	OSCUPH0

- **OSCUPHx: Output Selection Clear for PWMH output of the channel x**

0: No effect.

1: Dead-time generator output DTOHx selected as PWMH output of channel x at the beginning of the next channel x PWM period.

- **OSCUPLx: Output Selection Clear for PWML output of the channel x**

0: No effect.

1: Dead-time generator output DTOLx selected as PWML output of channel x at the beginning of the next channel x PWM period.

39.7.24 PWM Fault Mode Register

Name: PWM_FMR

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 bits WPSWS5 and WPHWS5 are cleared in the PWM Write Protection Status Register.

Refer to Section 39.5.4 “Fault Inputs” for details on fault generation.

- **FPOL: Fault Polarity**

For each bit y of FPOL, where y is the 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**

For each bit y of FMOD, where y is the 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**

For each bit y of FFIL, where y is the 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, the bit FMODy can be set to ‘1’ only if the FPOLy bit has been previously configured to its final value.

39.7.25 PWM Fault Status Register

Name: PWM_FSR

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							

Refer to Section 39.5.4 “Fault Inputs” for details on fault generation.

- **FIV: Fault Input Value**

For each bit y of FIV, where y is the 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**

For each bit y of FS, where y is the fault input number:

0: The fault y is not currently active.

1: The fault y is currently active.

39.7.26 PWM Fault Clear Register

Name: PWM_FCR

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							

Refer to Section 39.5.4 “Fault Inputs” for details on fault generation.

- **FCLR: Fault Clear**

For each bit y of FCLR, where y is the 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), else writing this bit to ‘1’ has no effect.

39.7.27 PWM Fault Protection Value Register 1

Name: PWM_FPV1

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	FPVL3	FPVL2	FPVL1	FPVL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	FPVH3	FPVH2	FPVH1	FPVH0

This register can only be written if bits WPSWS5 and WPHWS5 are cleared in the PWM Write Protection Status Register.

Refer to Section 39.5.4 “Fault Inputs” for details on fault generation.

- **FPVHx: Fault Protection Value for PWMH output on channel x**

This bit is taken into account only if the bit FPZHx is set to ‘0’ in PWM Fault Protection Value Register 2.

0: PWMH output of channel x is forced to ‘0’ when fault occurs.

1: PWMH output of channel x is forced to ‘1’ when fault occurs.

- **FPVLx: Fault Protection Value for PWML output on channel x**

This bit is taken into account only if the bit FPZLx is set to ‘0’ in PWM Fault Protection Value Register 2.

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.

39.7.28 PWM Fault Protection Enable Register

Name: PWM_FPE

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 bits WPSWS5 and WPHWS5 are cleared in the PWM Write Protection Status Register.

Only the first 8 bits (number of fault input pins) of fields FPE0, FPE1, FPE2 and FPE3 are significant.

Refer to Section 39.5.4 “Fault Inputs” for details on fault generation.

- **FPEx: Fault Protection Enable for channel x**

For each bit y of FPEx, where y is the 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 field has been previously configured to its final value in PWM Fault Mode Register.

39.7.29 PWM Event Line x Register

Name: PWM_ELMRx

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.

39.7.30 PWM Spread Spectrum Register

Name: PWM_SSPR

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	SPRDM

23	22	21	20	19	18	17	16
SPRD							

15	14	13	12	11	10	9	8
SPRD							

7	6	5	4	3	2	1	0
SPRD							

This register can only be written if bits WPSWS3 and WPHWS3 are cleared in the PWM Write Protection Status Register.

Only the first 16 bits (channel counter size) are significant.

- **SPRD: Spread Spectrum Limit Value**

The spread spectrum limit value defines the range for the spread spectrum counter. It is introduced in order to achieve constant varying PWM period for the output waveform.

- **SPRDM: Spread Spectrum Counter Mode**

0: Triangular mode. The spread spectrum counter starts to count from -SPRD when the channel 0 is enabled and counts upwards at each PWM period. When it reaches +SPRD, it restarts to count from -SPRD again.

1: Random mode. The spread spectrum counter is loaded with a new random value at each PWM period. This random value is uniformly distributed and is between -SPRD and +SPRD.

39.7.31 PWM Spread Spectrum Update Register

Name: PWM_SSPUP

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
SPRDUP							

15	14	13	12	11	10	9	8
SPRDUP							

7	6	5	4	3	2	1	0
SPRDUP							

This register can only be written if bits WPSWS3 and WPHWS3 are cleared in the PWM Write Protection Status Register.

This register acts as a double buffer for the SPRD value. This prevents an unexpected waveform when modifying the spread spectrum limit value.

Only the first 16 bits (channel counter size) are significant.

- **SPRDUP: Spread Spectrum Limit Value Update**

The spread spectrum limit value defines the range for the spread spectrum counter. It is introduced in order to achieve constant varying period for the output waveform.

39.7.32 PWM Stepper Motor Mode Register

Name: PWM_SMMR

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	DOWN1	DOWN0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	–	–	GCEN1	GCEN0

- **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.

39.7.33 PWM Fault Protection Value Register 2

Name: PWM_FPV2

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	FPZL3	FPZL2	FPZL1	FPZL0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	FPZH3	FPZH2	FPZH1	FPZH0

This register can only be written if bits WPSWS5 and WPHWS5 are cleared in the PWM Write Protection Status Register.

- **FPZHx: Fault Protection to Hi-Z for PWMH output on channel x**

0: When fault occurs, PWMH output of channel x is forced to value defined by the bit FPVHx in PWM Fault Protection Value Register 1.

1: When fault occurs, PWMH output of channel x is forced to high-impedance state.

- **FPZLx: Fault Protection to Hi-Z for PWML output on channel x**

0: When fault occurs, PWML output of channel x is forced to value defined by the bit FPVLx in PWM Fault Protection Value Register 1.

1: When fault occurs, PWML output of channel x is forced to high-impedance state.

39.7.34 PWM Write Protection Control Register

Name: PWM_WPCR

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	

See Section 39.6.6 “Register Write Protection” for the list of registers that can be write-protected.

- **WPCMD: Write Protection Command**

This command is performed only if the WPKEY corresponds to 0x50574D (“PWM” in ASCII).

Value	Name	Description
0	DISABLE_SW_PROT	Disables the software write protection of the register groups of which the bit WPRGx is at ‘1’.
1	ENABLE_SW_PROT	Enables the software write protection of the register groups of which the bit WPRGx is at ‘1’.
2	ENABLE_HW_PROT	Enables the hardware write protection of the register groups of which the bit WPRGx is at ‘1’. Only a hardware reset of the PWM controller can disable the hardware write protection. Moreover, to meet security requirements, the PIO lines associated with the PWM can not be configured through the PIO interface.

- **WPRGx: Write Protection 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 Protection Key**

Value	Name	Description
0x50574D	PASSWD	Writing any other value in this field aborts the write operation of the WPCMD field. Always reads as 0

39.7.35 PWM Write Protection Status Register

Name: PWM_WPSR

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 SW write protection x of the register group x is disabled.

1: The SW write protection x of the register group x is enabled.

- **WPHWSx: Write Protect HW Status**

0: The HW write protection x of the register group x is disabled.

1: The HW write protection x of the register group x is enabled.

- **WPVS: Write Protect Violation Status**

0: No write protection violation has occurred since the last read of PWM_WPSR.

1: At least one write protection violation has occurred since the last read of PWM_WPSR. 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 = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

39.7.36 PWM Comparison x Value Register

Name: PWM_CMPVx

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 not relevant if the counter of the channel 0 is left-aligned (CALG = 0 in PWM Channel Mode Register)

39.7.37 PWM Comparison x Value Update Register

Name: PWM_CMPVUPDx

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 not relevant if the counter of the channel 0 is left-aligned (CALG = 0 in PWM Channel Mode Register)

CAUTION: The write of the register PWM_CMPVUPDx must be followed by a write of the register PWM_CMPMUPDx.

39.7.38 PWM Comparison x Mode Register

Name: PWM_CMPMx

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

39.7.39 PWM Comparison x Mode Update Register

Name: PWM_CMPMUPDx

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.

39.7.40 PWM Channel Mode Register

Name: PWM_CMRx [x=0..3]

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	DTLI	DTHI	DTE
15	14	13	12	11	10	9	8
–	–	TCTS	–	UPDS	CES	CPOL	CALG
7	6	5	4	3	2	1	0
–	–	–	–	CPRE			

This register can only be written if bits WPSWS2 and WPHWS2 are cleared in the PWM Write Protection Status Register.

- **CPRE: Channel Pre-scaler**

Value	Name	Description
0	MCK	Peripheral clock
1	MCK_DIV_2	Peripheral clock/2
2	MCK_DIV_4	Peripheral clock/4
3	MCK_DIV_8	Peripheral clock/8
4	MCK_DIV_16	Peripheral clock/16
5	MCK_DIV_32	Peripheral clock/32
6	MCK_DIV_64	Peripheral clock/64
7	MCK_DIV_128	Peripheral clock/128
8	MCK_DIV_256	Peripheral clock/256
9	MCK_DIV_512	Peripheral clock/512
10	MCK_DIV_1024	Peripheral clock/1024
11	CLKA	Clock A
12	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 PWM Interrupt Status Register 1).

CALG = 0 (Left Alignment):

0/1: The channel counter event occurs at the end of the PWM period.

CALG = 1 (Center Alignment):

0: The channel counter event occurs at the end of the PWM period.

1: The channel counter event occurs at the end of the PWM period and at half the PWM period.

- **UPDS: Update Selection**

When the period is center aligned, the bit UPDS defines when the update of the duty cycle, the polarity value/mode occurs after writing the corresponding update registers.

CALG = 0 (Left Alignment):

0/1: The update always occurs at the end of the PWM period after writing the update register(s).

CALG = 1 (Center Alignment):

0: The update occurs at the next end of the PWM period after writing the update register(s).

1: The update occurs at the next end of the PWM half period after writing the update register(s).

- **TCTS: Timer Counter Trigger Selection**

0: The comparator of the channel x (OCx) is used as the trigger source for the Timer Counter (TC).

1: The counter events of the channel x is used as the trigger source for the Timer Counter (TC).

- **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.

39.7.41 PWM Channel Duty Cycle Register

Name: PWM_CDTYx [x=0..3]

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
CDTY							

15	14	13	12	11	10	9	8
CDTY							

7	6	5	4	3	2	1	0
CDTY							

Only the first 16 bits (channel counter size) are significant.

- **CDTY: Channel Duty-Cycle**

Defines the waveform duty-cycle. This value must be defined between 0 and CPRD (PWM_CPRDx).

39.7.42 PWM Channel Duty Cycle Update Register

Name: PWM_CDTYUPDx [x=0..3]

Access: Write-only.

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
CDTYUPD							

15	14	13	12	11	10	9	8
CDTYUPD							

7	6	5	4	3	2	1	0
CDTYUPD							

This register acts as a double buffer for the CDTY value. This prevents an unexpected waveform when modifying the waveform duty-cycle.

Only the first 16 bits (channel counter size) are significant.

- **CDTYUPD: Channel Duty-Cycle Update**

Defines the waveform duty-cycle. This value must be defined between 0 and CPRD (PWM_CPRDx).

39.7.43 PWM Channel Period Register

Name: PWM_CPRDx [x=0..3]

Access: Read/Write

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

This register can only be written if bits WPSWS3 and WPHWS3 are cleared in the PWM Write Protection Status Register.

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 peripheral clock divided by a given prescaler value “X” (where X = 2^{PREA} is 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024). The resulting period formula is:

$$\frac{(X \times CPRD)}{f_{\text{peripheral clock}}}$$

– By using the PWM peripheral clock divided by a given prescaler value “X” (see above) and by either the DIVA or the DIVB divider. The formula becomes, respectively:

$$\frac{(X \times CRPD \times DIVA)}{f_{\text{peripheral clock}}} \text{ or } \frac{(X \times CRPD \times DIVB)}{f_{\text{peripheral clock}}}$$

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 peripheral clock divided by a given prescaler value “X” (where X = 2^{PREA} is 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024). The resulting period formula is:

$$\frac{(2 \times X \times CPRD)}{f_{\text{peripheral clock}}}$$

– By using the PWM peripheral clock divided by a given prescaler value “X” (see above) and by either the DIVA or the DIVB divider. The formula becomes, respectively:

$$\frac{(2 \times X \times CPRD \times DIVA)}{f_{\text{peripheral clock}}} \text{ or } \frac{(2 \times X \times CPRD \times DIVB)}{f_{\text{peripheral clock}}}$$

39.7.44 PWM Channel Period Update Register

Name: PWM_CPRDUPDx [x=0..3]

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
CPRDUPD							

15	14	13	12	11	10	9	8
CPRDUPD							

7	6	5	4	3	2	1	0
CPRDUPD							

This register can only be written if bits WPSWS3 and WPHWS3 are cleared in the PWM Write Protection Status Register.

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 peripheral clock divided by a given prescaler value “X” (where X = 2^{PREA} is 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024). The resulting period formula is:

$$\frac{(X \times CPRDUPD)}{f_{\text{peripheral clock}}}$$

– By using the PWM peripheral clock divided by a given prescaler value “X” (see above) and by either the DIVA or the DIVB divider. The formula becomes, respectively:

$$\frac{(X \times CRPDUPD \times DIVA)}{f_{\text{peripheral clock}}} \text{ or } \frac{(X \times CRPDUPD \times DIVB)}{f_{\text{peripheral clock}}}$$

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 peripheral clock divided by a given prescaler value “X” (where X = 2^{PREA} is 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024). The resulting period formula is:

$$\frac{(2 \times X \times CPRDUPD)}{f_{\text{peripheral clock}}}$$

– By using the PWM peripheral clock divided by a given prescaler value “X” (see above) and by either the DIVA or the DIVB divider. The formula becomes, respectively:

$$\frac{(2 \times X \times CPRDUPD \times DIVA)}{f_{\text{peripheral clock}}} \text{ or } \frac{(2 \times X \times CPRDUPD \times DIVB)}{f_{\text{peripheral clock}}}$$

39.7.45 PWM Channel Counter Register

Name: PWM_CCNTx [x=0..3]

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
CNT							

15	14	13	12	11	10	9	8
CNT							

7	6	5	4	3	2	1	0
CNT							

Only the first 16 bits (channel counter size) are significant.

- **CNT: Channel Counter Register**

Channel counter value. This register is reset when:

- the channel is enabled (writing CHIDx in the PWM_ENA register).
- the channel counter reaches CPRD value defined in the PWM_CPRDx register if the waveform is left-aligned.

39.7.46 PWM Channel Dead Time Register

Name: PWM_DTx [x=0..3]

Access: Read/Write

31	30	29	28	27	26	25	24
DTL							

23	22	21	20	19	18	17	16
DTL							

15	14	13	12	11	10	9	8
DTH							

7	6	5	4	3	2	1	0
DTH							

This register can only be written if bits WPSWS4 and WPHWS4 are cleared in the PWM Write Protection Status Register.

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 the value (CPRD – CDTY) (PWM_CPRDx 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).

39.7.47 PWM Channel Dead Time Update Register

Name: PWM_DTUPDx [x=0..3]

Access: Write-only

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

This register can only be written if bits WPSWS4 and WPHWS4 are cleared in the PWM Write Protection Status Register.

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 the value (CPRD – CDTY) (PWM_CPRDx 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.

39.7.48 PWM Channel Mode Update Register

Name: PWM_CMUPDx [x=0..3]

Access: Read/Write

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

This register can only be written if bits WPSWS2 and WPHWS2 are cleared in the PWM Write Protection Status Register.

This register acts as a double buffer for the CPOL value. This prevents an unexpected waveform when modifying the polarity value.

- **CPOLUP: Channel Polarity Update**

The write of this bit is taken into account only if the bit CPOLINVUP is written at ‘0’ at the same time.

0: The OCx output waveform (output from the comparator) starts at a low level.

1: The OCx output waveform (output from the comparator) starts at a high level.

- **CPOLINVUP: Channel Polarity Inversion Update**

If this bit is written at ‘1’, the write of the bit CPOLUP is not taken into account.

0: No effect.

1: The OCx output waveform (output from the comparator) is inverted.

40. High Speed Multimedia Card Interface (HSMCI)

40.1 Description

The High Speed Multimedia Card Interface (HSMCI) supports the MultiMedia Card (MMC) Specification V4.3, the SD Memory Card Specification V2.0, the SDIO V2.0 specification and CE-ATA V1.1.

The HSMCI includes a command register, response registers, data registers, timeout counters and error detection logic that automatically handle the transmission of commands and, when required, the reception of the associated responses and data with a limited processor overhead.

The HSMCI supports stream, block and multi block data read and write, and is compatible with the Peripheral DMA Controller (PDC) Channels, minimizing processor intervention for large buffer transfers.

The HSMCI operates at a rate of up to Master Clock divided by 2 and supports the interfacing of 1 slot(s). Each slot may be used to interface with a High Speed MultiMedia Card bus (up to 30 Cards) or with an SD Memory Card. 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.

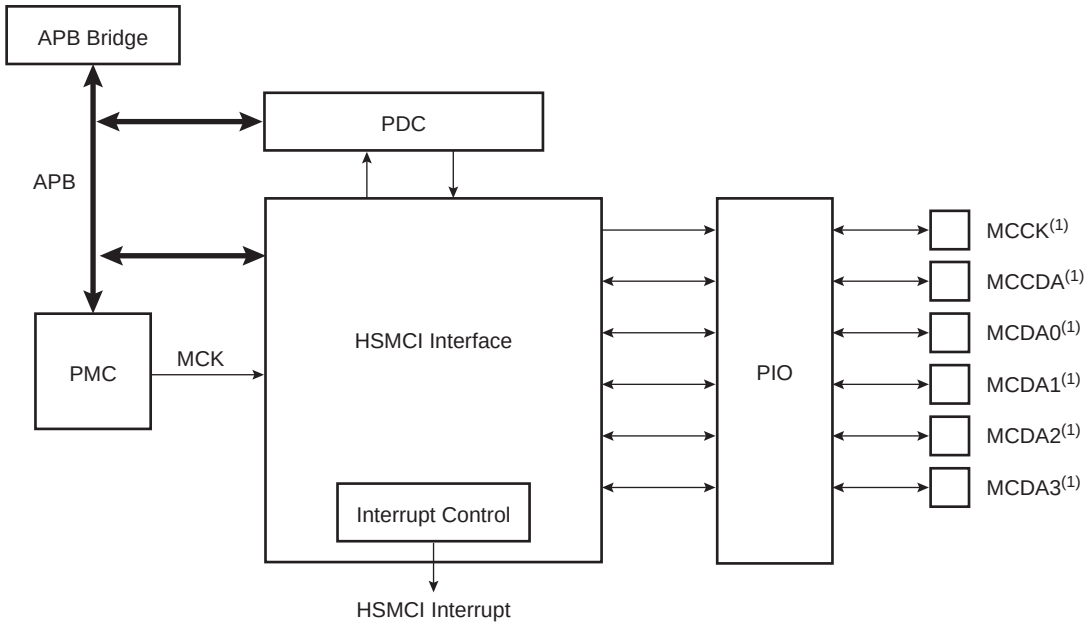
HSMCI fully supports CE-ATA Revision 1.1, built on the MMC System Specification v4.0. The module includes dedicated hardware to issue the command completion signal and capture the host command completion signal disable.

40.2 Embedded Characteristics

- Compatible with MultiMedia Card Specification Version 4.3
- Compatible with SD Memory Card Specification Version 2.0
- Compatible with SDIO Specification Version 2.0
- Compatible with CE-ATA Specification 1.1
- Cards Clock Rate Up to Master Clock Divided by 2
- Boot Operation Mode Support
- High Speed Mode Support
- Embedded Power Management to Slow Down Clock Rate When Not Used
- Supports 1 Multiplexed Slot(s)
  - Each Slot for either a High Speed MultiMedia Card Bus (Up to 30 Cards) or an SD Memory Card
- Support for Stream, Block and Multi-block Data Read and Write
- Supports Connection to Peripheral DMA Controller (PDC)
  - Minimizes Processor Intervention for Large Buffer Transfers
- Built in FIFO (from 16 to 256 bytes) with Large Memory Aperture Supporting Incremental Access
- Support for CE-ATA Completion Signal Disable Command
- Protection Against Unexpected Modification On-the-Fly of the Configuration Registers

40.3 Block Diagram

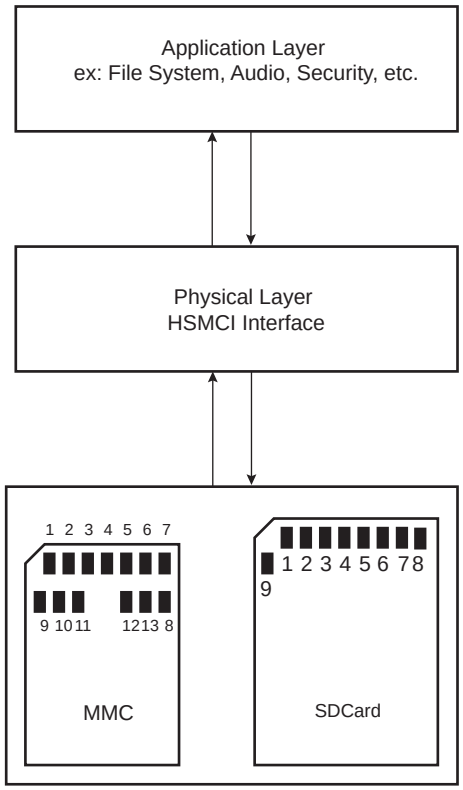
Figure 40-1. Block Diagram (4-bit configuration)



Note: 1. When several HSMCI (x HSMCI) are embedded in a product, MCCK refers to HSMCIx_CK, MCCDA to HSMCIx_CDA, MCDAy to HSMCIx_DAy.

40.4 Application Block Diagram

Figure 40-2. Application Block Diagram



40.5 Pin Name List

Table 40-1. I/O Lines Description for 4-bit Configuration

Pin Name[1]	Pin Description	Type[2]	Comments
MCCDA	Command/response	I/O/PP/OD	CMD of an MMC or SDCard/SDIO
MCCK	Clock	I/O	CLK of an MMC or SD Card/SDIO
MCDA0–MCDA3	Data 0..3 of Slot A	I/O/PP	DAT[0..3] of an MMC DAT[0..3] of an SD Card/SDIO

Notes:
1. When several HSMCI (x HSMCI) are embedded in a product, MCCK refers to HSMCIx_CK, MCCDA to HSMCIx_CDA, MCDAy to HSMCIx_DAy.
2. I: Input, O: Output, PP: Push/Pull, OD: Open Drain.

40.6 Product Dependencies

40.6.1 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 40-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
HSMCI	MCCDA	PA28	C
HSMCI	MCCK	PA29	C
HSMCI	MCDA0	PA30	C
HSMCI	MCDA1	PA31	C
HSMCI	MCDA2	PA26	C
HSMCI	MCDA3	PA27	C

40.6.2 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.

40.6.3 Interrupt Sources

The HSMCI has an interrupt line connected to the interrupt controller.

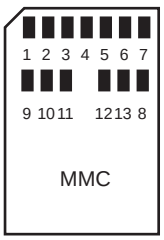
Handling the HSMCI interrupt requires programming the interrupt controller before configuring the HSMCI.

Table 40-3. Peripheral IDs

Instance	ID
HSMCI	16

40.7 Bus Topology

Figure 40-3. High Speed MultiMedia Memory Card Bus Topology



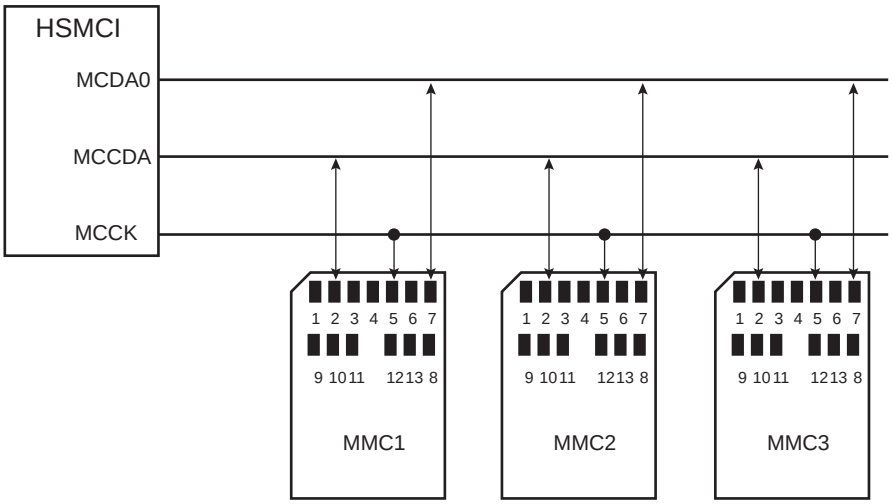
The High Speed MultiMedia Card communication is based on a 13-pin serial bus interface. It has three communication lines and four supply lines.

Table 40-4. 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

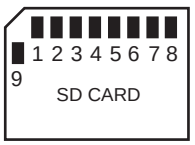
Notes: 1. I: Input, O: Output, PP: Push/Pull, OD: Open Drain.
2. When several HSMCI (x HSMCI) are embedded in a product, MCCK refers to HSMCIx_CK, MCCDA to HSMCIx_CDA, MCDAy to HSMCIx_DAy.

Figure 40-4. 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 40-5. SD Memory Card Bus Topology



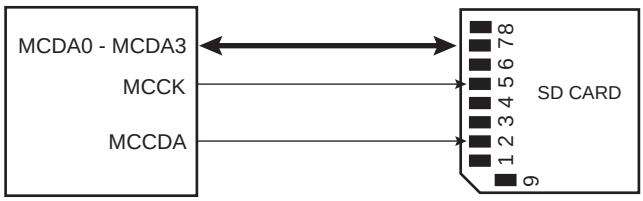
The SD Memory Card bus includes the signals listed in Table 40-5.

Table 40-5. 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, MCDAy to HSMCIx_DAy.

Figure 40-6. SD Card Bus Connections with One Slot



Note: When several HSMCI (x HSMCI) are embedded in a product, MCCK refers to HSMCIx_CK, MCCDA to HSMCIx_CDA MCDAy to HSMCIx_DAy.

When the HSMCI is configured to operate with SD memory cards, the width of the data bus can be selected in the HSMCI_SDCR. 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.

40.8 High Speed MultiMedia Card Operations

After a power-on reset, the cards are initialized by a special message-based High Speed MultiMedia Card bus protocol. Each message is represented by one of the following tokens:

- Command—A command is a token that starts an operation. A command is sent from the host either to a single card (addressed command) or to all connected cards (broadcast command). A command is transferred serially on the CMD line.
- Response—A response is a token which is sent from an addressed card or (synchronously) from all connected cards to the host as an answer to a previously received command. A response is transferred serially on the CMD line.
- Data—Data can be transferred from the card to the host or vice versa. Data is transferred via the data line.

Card addressing is implemented using a session address assigned during the initialization phase by the bus controller to all currently connected cards. Their unique CID number identifies individual cards.

The structure of commands, responses and data blocks is described in the High Speed MultiMedia Card System Specification. See also Table 40-6 on page 1099.

High Speed MultiMedia Card bus data transfers are composed of these tokens.

There are different types of operations. Addressed operations always contain a command and a response token. In addition, some operations have a data token; the others transfer their information directly within the command or response structure. In this case, no data token is present in an operation. The bits on the DAT and the CMD lines are transferred synchronous to the clock HSMCI clock.

Two types of data transfer commands are defined:

- Sequential commands—These commands initiate a continuous data stream. They are terminated only when a stop command follows on the CMD line. This mode reduces the command overhead to an absolute minimum.
- Block-oriented commands—These commands send a data block succeeded by CRC bits.

Both read and write operations allow either single or multiple block transmission. A multiple block transmission is terminated when a stop command follows on the CMD line similarly to the sequential read or when a multiple block transmission has a predefined block count (see Section 40.8.2 “Data Transfer Operation”).

The HSMCI provides a set of registers to perform the entire range of High Speed MultiMedia Card operations.

40.8.1 Command - Response Operation

After reset, the HSMCI is disabled and becomes valid after setting the MCIEN bit in the HSMCI_CR.

The PWSEN bit saves power by dividing the HSMCI clock by $2^{PWSDIV} + 1$ when the bus is inactive.

The two bits, RDPROOF and WRPROOF in the HSMCI Mode Register (HSMCI_MR) allow stopping the HSMCI clock during read or write access if the internal FIFO is full. This will guarantee data integrity, not bandwidth.

All the timings for High Speed MultiMedia Card are defined in the High Speed MultiMedia Card System Specification.

The two bus modes (open drain and push/pull) needed to process all the operations are defined in the HSMCI Command Register (HSMCI_CMDR). The HSMCI_CMDR allows a command to be carried out.

For example, to perform an ALL_SEND_CID command:

	Host Command					N_{ID} Cycles			Response			High Impedance State		
CMD	S	T	Content	CRC	E	Z	******	Z	S	T	CID Content	Z	Z	Z

The command ALL_SEND_CID and the fields and values for the HSMCI_CMDR are described in Table 40-6 and Table 40-7.

Table 40-6. ALL_SEND_CID Command Description

CMD Index	Type	Argument	Response	Abbreviation	Command Description
CMD2	bcr(1)	[31:0] stuff bits	R2	ALL_SEND_CID	Asks all cards to send their CID numbers on the CMD line

Note: 1. bcr means broadcast command with response.

Table 40-7. Fields and Values for HSMCI_CMDR

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 40-7).

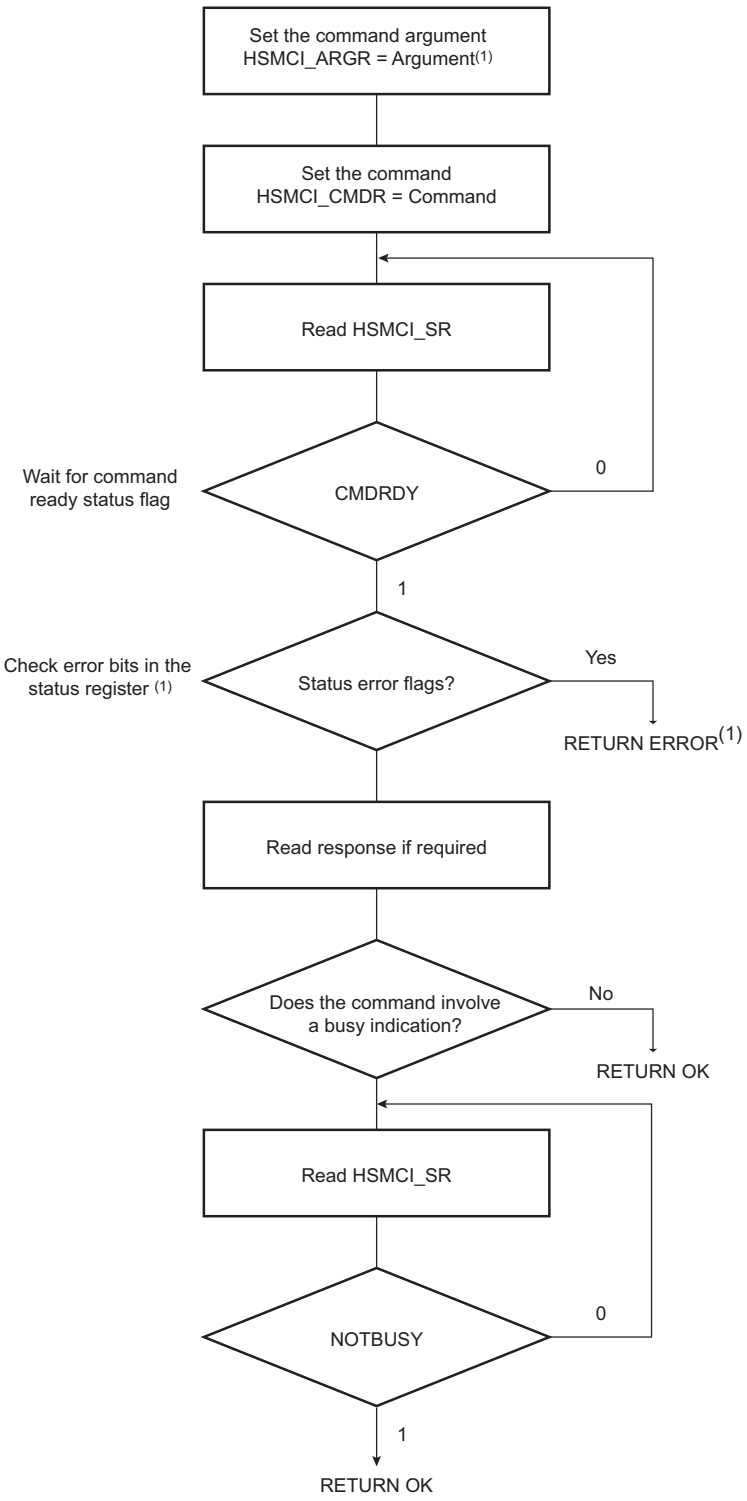
The command is sent immediately after writing the command register.

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 HSMCI Interrupt Enable Register (HSMCI_IER) allows using an interrupt method.

Figure 40-7. Command/Response Functional Flow Diagram



Note: 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).

40.8.2 Data Transfer Operation

The High Speed MultiMedia Card allows several read/write operations (single block, multiple blocks, stream, etc.). These kinds of transfer can be selected setting the Transfer Type (TRTYP) field in the HSMCI Command Register (HSMCI_CMDR).

These operations can be done using the features of the Peripheral DMA Controller (PDC). If the PDCMODE bit is set in HSMCI_MR, then all reads and writes use the PDC facilities.

In all cases, the block length (BLKLEN field) must be defined either in the HSMCI Mode Register (HSMCI_MR) or in the HSMCI 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.

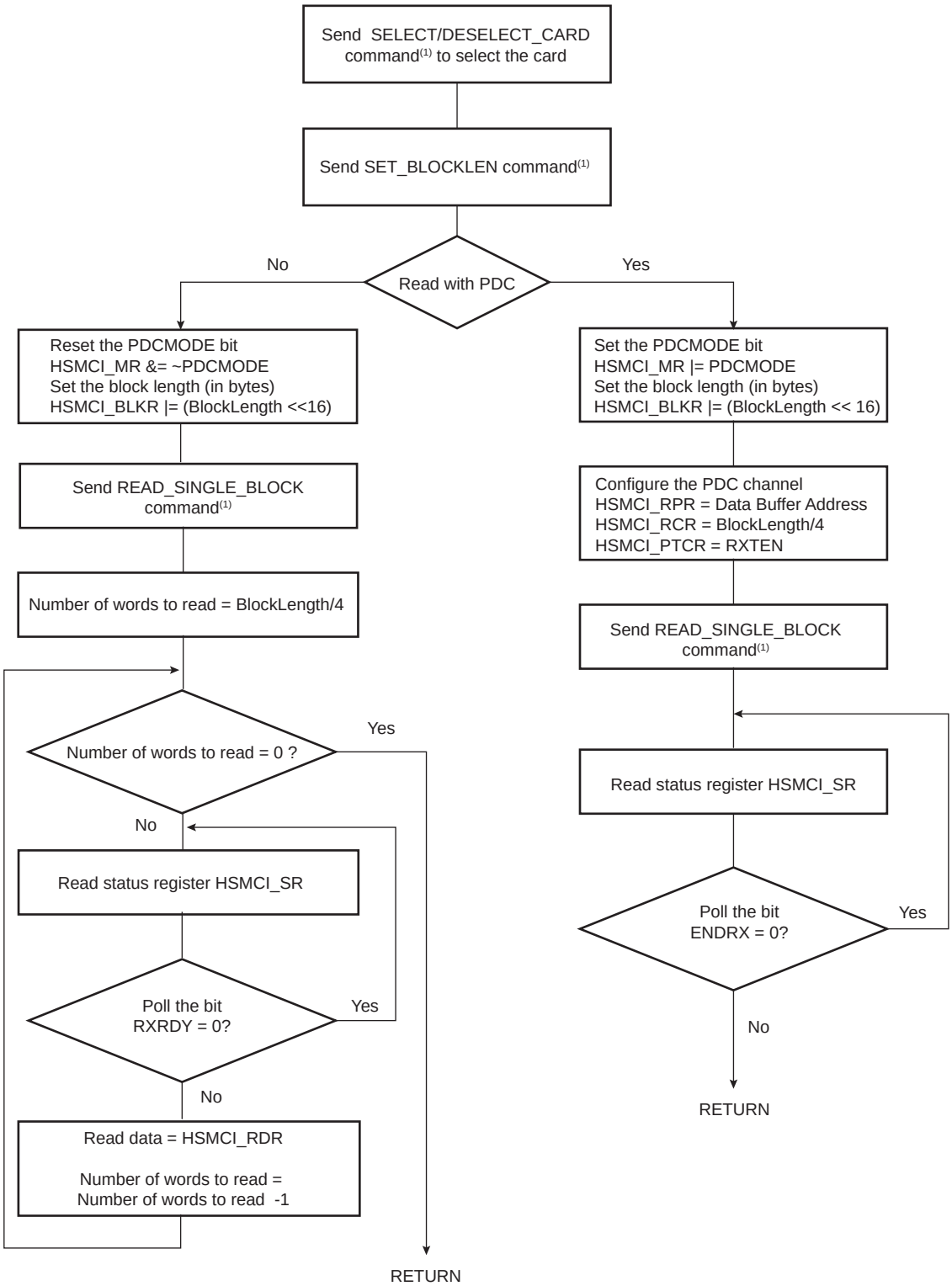
- Multiple block read (or write) with predefined block count (since version 3.1 and higher):

  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 predefined 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 HSMCI_BLKR 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.

40.8.3 Read Operation

The following flowchart (Figure 40-8) shows how to read a single block with or without use of PDC facilities. In this example, a polling method is used to wait for the end of read. Similarly, the user can configure the HSMCI Interrupt Enable Register (HSMCI_IER) to trigger an interrupt at the end of read.

Figure 40-8. Read Functional Flow Diagram



Note: 1. It is assumed that this command has been correctly sent (see Figure 40-7).

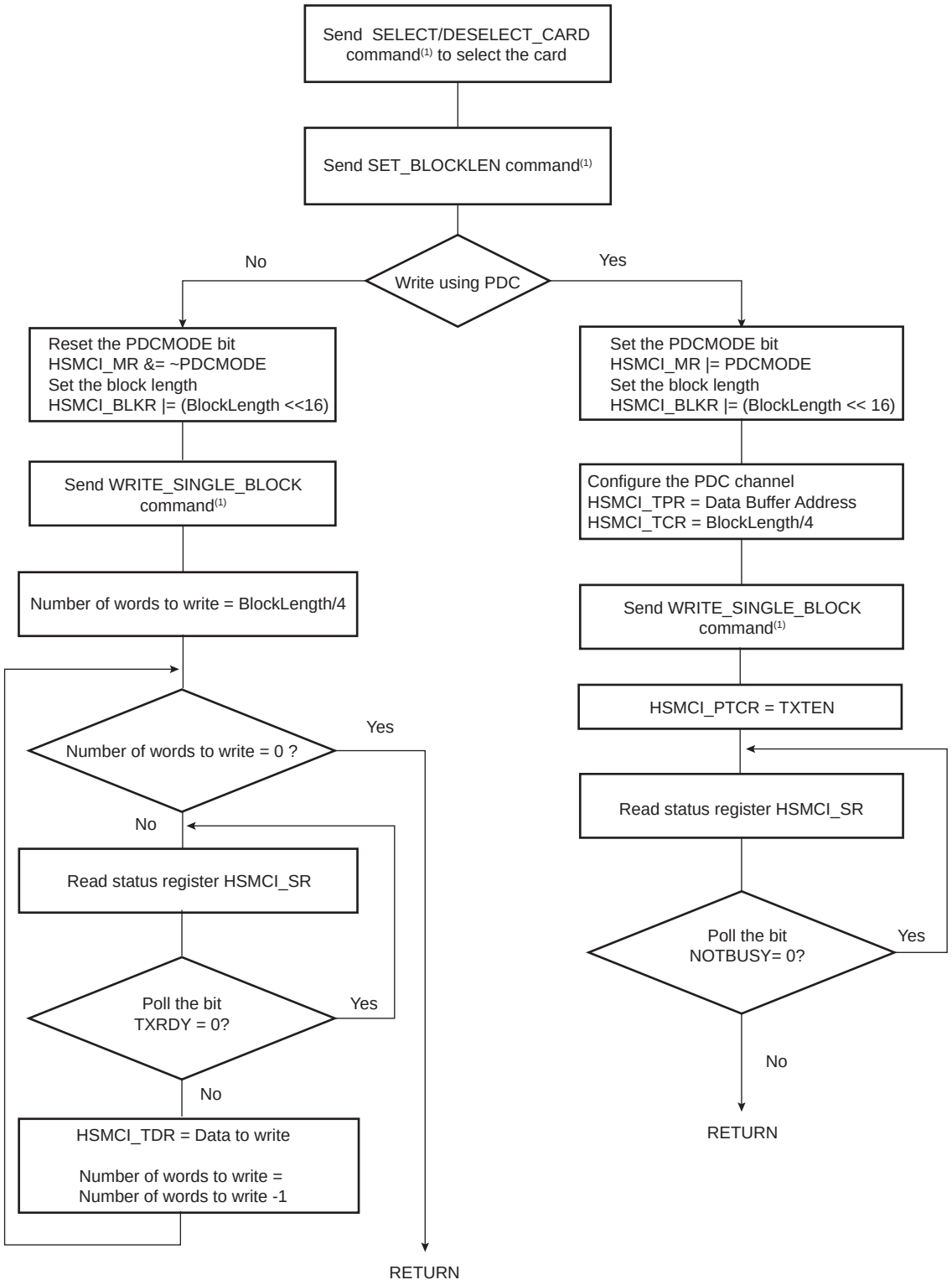
40.8.4 Write Operation

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 PDCMODE enables PDC transfer.

The flowchart in Figure 40-9 shows how to write a single block with or without use of PDC facilities. Polling or interrupt method can be used to wait for the end of write according to the contents of the HSMCI Interrupt Mask Register (HSMCI_IMR).

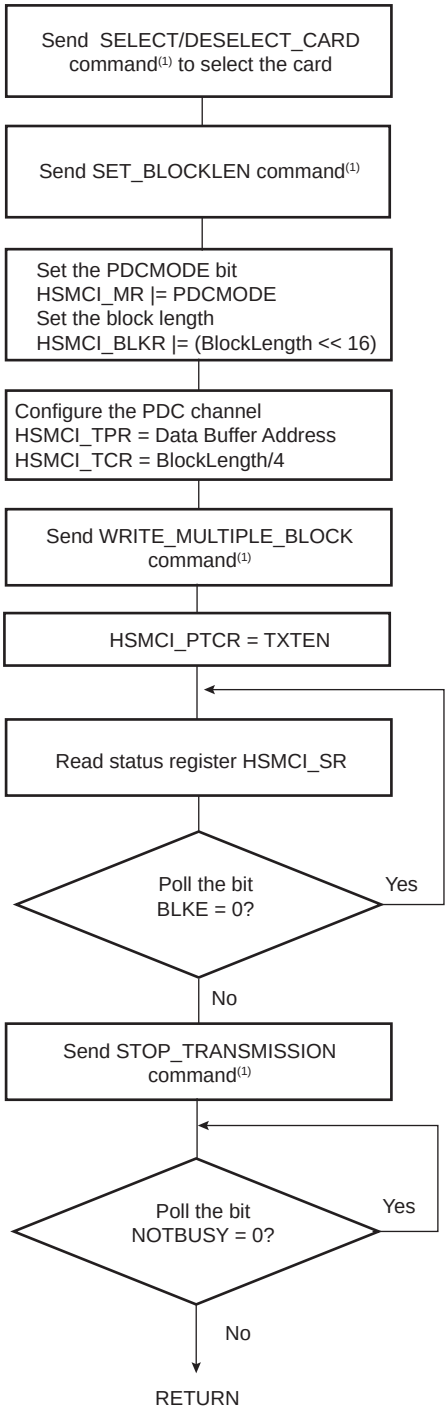
Figure 40-9. Write Functional Flow Diagram



Note: 1. It is assumed that this command has been correctly sent (see Figure 40-7).

The flowchart in Figure 40-10 shows how to manage a multiple write block transfer with the PDC. Polling or interrupt method can be used to wait for the end of write according to the contents of the HSMCI_IMR.

Figure 40-10. Multiple Write Functional Flow Diagram



Note: 1. It is assumed that this command has been correctly sent (see Figure 40-7).

40.9 SD/SDIO Card Operation

The High Speed MultiMedia Card Interface allows processing of SD Memory (Secure Digital Memory Card) and SDIO (SD Input Output) Card commands.

SD/SDIO cards are based on the MultiMedia Card (MMC) format, but are physically slightly thicker and feature higher data transfer rates, a lock switch on the side to prevent accidental overwriting and security features. The

physical form factor, pin assignment and data transfer protocol are forward-compatible with the High Speed MultiMedia Card with some additions. SD slots can actually be used for more than flash memory cards. Devices that support SDIO can use small devices designed for the SD form factor, such as GPS receivers, Wi-Fi or Bluetooth adapters, modems, barcode readers, IrDA adapters, FM radio tuners, RFID readers, digital cameras and more.

SD/SDIO is covered by numerous patents and trademarks, and licensing is only available through the Secure Digital Card Association.

The SD/SDIO Card communication is based on a 9-pin interface (Clock, Command, 4 x Data and 3 x Power lines). The communication protocol is defined as a part of this specification. The main difference between the SD/SDIO Card and the High Speed MultiMedia Card is the initialization process.

The SD/SDIO Card Register (HSMCI_SDCR) allows selection of the Card Slot and the data bus width.

The SD/SDIO Card bus allows dynamic configuration of the number of data lines. After power up, by default, the SD/SDIO Card uses only DAT0 for data transfer. After initialization, the host can change the bus width (number of active data lines).

40.9.1 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.

40.9.2 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.

40.10 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.

40.10.1 Executing an ATA Polling Command

1. Issue READ_DMA_EXT with RW_MULTIPLE_REGISTER (CMD60) for 8 KB 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 configured to 0.

40.10.2 Executing an ATA Interrupt Command

1. Issue READ_DMA_EXT with RW_MULTIPLE_REGISTER (CMD60) for 8 KB 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.

40.10.3 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.

40.10.4 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.

40.11 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.

40.11.1 Boot Procedure, Processor Mode

1. Configure the HSMCI data bus width programming SDCBUS Field in the HSMCI_SDCR. 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.
3. Issue the Boot Operation Request command by writing to the HSMCI_CMDR 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 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 with SPCMD field set to BOOTEND.

40.12 HSMCI Transfer Done Timings

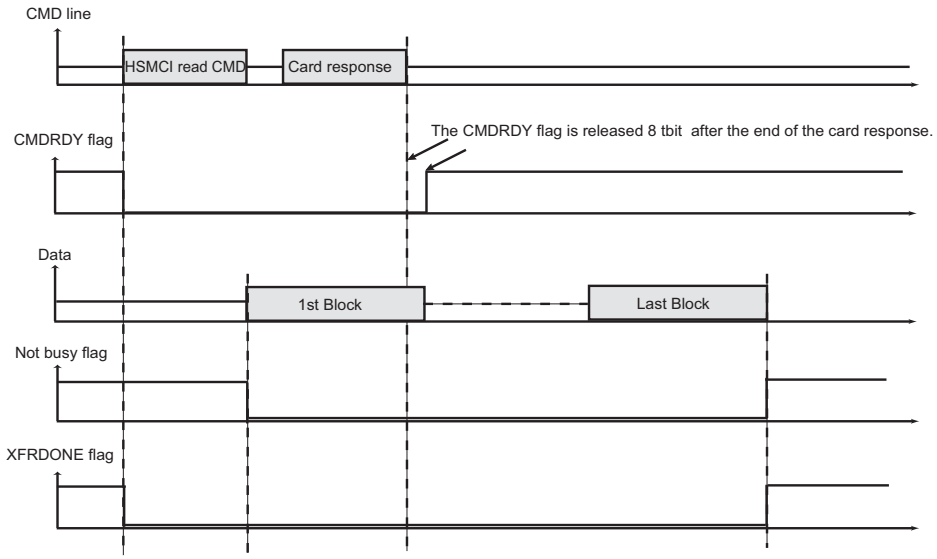
40.12.1 Definition

The XFRDONE flag in the HSMCI_SR indicates exactly when the read or write sequence is finished.

40.12.2 Read Access

During a read access, the XFRDONE flag behaves as shown in Figure 40-11.

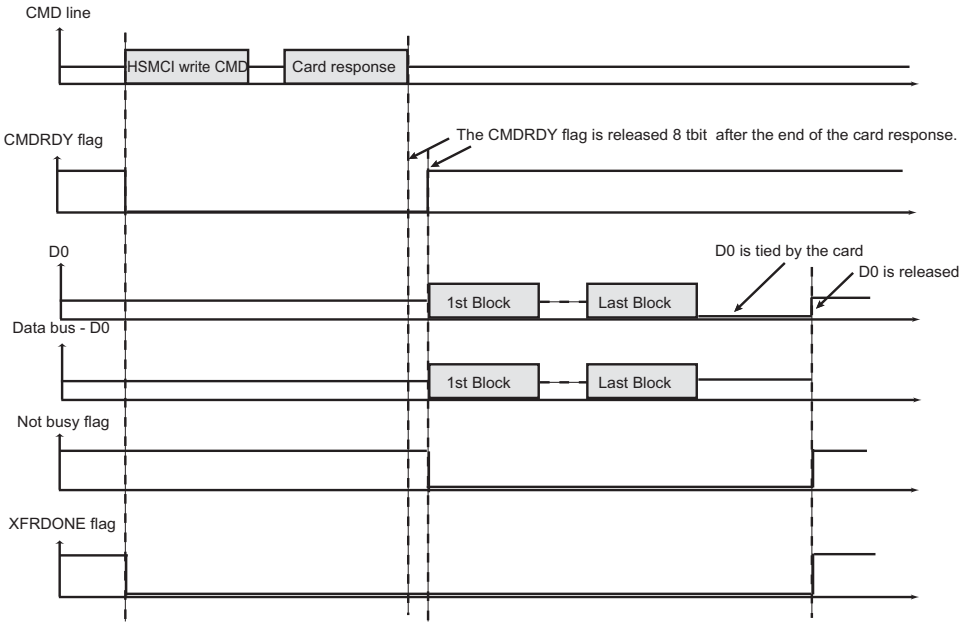
Figure 40-11. XFRDONE During a Read Access



40.12.3 Write Access

During a write access, the XFRDONE flag behaves as shown in Figure 40-12.

Figure 40-12. XFRDONE During a Write Access



40.13 Register Write Protection

To prevent any single software error from corrupting HSMCI behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the HSMCI Write Protection Mode Register (HSMCI_WPMR).

If a write access to a write-protected register is detected, the WPVS bit in the HSMCI Write Protection Status Register (HSMCI_WPSR) is set and the field WPVSRC indicates the register in which the write access has been attempted.

The WPVS bit is automatically cleared after reading the HSMCI_WPSR.

The following registers can be protected:

- HSMCI Mode Register
- HSMCI Data Timeout Register
- HSMCI SDCard/SDIO Register
- HSMCI Completion Signal Timeout Register
- HSMCI Configuration Register

40.14 High Speed MultiMedia Card Interface (HSMCI) User Interface

Table 40-8. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	HSMCI_CR	Write-only	–
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-only	–
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-only	0x0
0x24	Response Register(1)	HSMCI_RSPR	Read-only	0x0
0x28	Response Register(1)	HSMCI_RSPR	Read-only	0x0
0x2C	Response Register(1)	HSMCI_RSPR	Read-only	0x0
0x30	Receive Data Register	HSMCI_RDR	Read-only	0x0
0x34	Transmit Data Register	HSMCI_TDR	Write-only	–
0x38–0x3C	Reserved	–	–	–
0x40	Status Register	HSMCI_SR	Read-only	0xC0E5
0x44	Interrupt Enable Register	HSMCI_IER	Write-only	–
0x48	Interrupt Disable Register	HSMCI_IDR	Write-only	–
0x4C	Interrupt Mask Register	HSMCI_IMR	Read-only	0x0
0x50	Reserved	–	–	–
0x54	Configuration Register	HSMCI_CFG	Read/Write	0x00
0x58–0xE0	Reserved	–	–	–
0xE4	Write Protection Mode Register	HSMCI_WPMR	Read/Write	–
0xE8	Write Protection Status Register	HSMCI_WPSR	Read-only	–
0xEC–0xFC	Reserved	–	–	–
0x100–0x128	Reserved for PDC registers	–	–	–
0x12C–0x1FC	Reserved	–	–	–
0x200	FIFO Memory Aperture0	HSMCI_FIFO0	Read/Write	0x0
...	...	...	...	...
0x5FC	FIFO Memory Aperture255	HSMCI_FIFO255	Read/Write	0x0

Notes: 1. The Response Register can be read by N accesses at the same HSMCI_RSPR or at consecutive addresses (0x20 to 0x2C). N depends on the size of the response.

40.14.1 HSMCI Control Register

Name: HSMCI_CR

Address: 0x40080000

Access: Write-only

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

- **MCIEN: Multi-Media Interface Enable**

0: No effect.

1: Enables the Multi-Media Interface if MCDIS is 0.

- **MCIDIS: Multi-Media Interface Disable**

0: No effect.

1: Disables the Multi-Media Interface.

- **PWSEN: Power Save Mode Enable**

0: No effect.

1: Enables the Power Saving Mode if PWSDIS is 0.

Warning: Before enabling this mode, the user must set a value different from 0 in the PWSDIV field of the 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 is performed.

40.14.2 HSMCI Mode Register

Name: HSMCI_MR

Address: 0x40080004

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	CLKODD
15	14	13	12	11	10	9	8
PDCMODE	PADV	FBYTE	WRPROOF	RDPROOF	PWSDIV		
7	6	5	4	3	2	1	0
CLKDIV							

This register can only be written if the WPEN bit is cleared in the HSMCI Write Protection Mode Register.

- **CLKDIV: Clock Divider**

High Speed MultiMedia Card Interface clock (MCCK or HSMCI_CK) is Master Clock (MCK) divided by ({CLKDIV,CLKODD}+2).

- **PWSDIV: Power Saving Divider**

High Speed MultiMedia Card Interface clock is divided by $2^{(PWSDIV)}$ + 1 when entering Power Saving Mode.

Warning: This value must be different from 0 before enabling the Power Save Mode in the HSMCI_CR (HSMCI_PWSEN bit).

- **RDPROOF: Read Proof Enable**

Enabling Read Proof allows to stop the HSMCI Clock during read access if the internal FIFO is full. This will guarantee data integrity, not bandwidth.

0: Disables Read Proof.

1: Enables Read Proof.

- **WRPROOF: Write Proof Enable**

Enabling Write Proof allows to stop the HSMCI Clock during write access if the internal FIFO is full. This will guarantee data integrity, not bandwidth.

0: Disables Write Proof.

1: Enables Write Proof.

- **FBYTE: Force Byte Transfer**

Enabling Force Byte Transfer allow byte transfers, so that transfer of blocks with a size different from modulo 4 can be supported.

Warning: BLKLEN value depends on FBYTE.

0: Disables Force Byte Transfer.

1: Enables Force Byte Transfer.

- **PADV: Padding Value**

0: 0x00 value is used when padding data in write transfer.

1: 0xFF value is used when padding data in write transfer.

PADV may be only in manual transfer.

- **PDCMODE: PDC-oriented Mode**

0: Disables PDC transfer

1: Enables PDC transfer. In this case, UNRE and OVRE flags in the HSMCI Status Register (HSMCI_SR) are deactivated after the PDC transfer has been completed.

- **CLKODD: Clock divider is odd**

This bit is the least significant bit of the clock divider and indicates the clock divider parity.

40.14.3 HSMCI Data Timeout Register

Name: HSMCI_DTOR

Address: 0x40080008

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the HSMCI Write Protection Mode Register.

- **DTOCYC: Data Timeout Cycle Number**

This field determines 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**

Value	Name	Description
0	1	DTOCYC
1	16	DTOCYC x 16
2	128	DTOCYC x 128
3	256	DTOCYC x 256
4	1024	DTOCYC x 1024
5	4096	DTOCYC x 4096
6	65536	DTOCYC x 65536
7	1048576	DTOCYC x 1048576

If the data time-out set by DTOCYC and DTOMUL has been exceeded, the Data Time-out Error flag (DTOE) in the HSMCI Status Register (HSMCI_SR) rises.

40.14.4 HSMCI SDCard/SDIO Register

Name: HSMCI_SDCR

Address: 0x4008000C

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the HSMCI Write Protection Mode Register.

- **SDCSEL: SDCard/SDIO Slot**

Value	Name	Description
0	SLOTA	Slot A is selected.
1	SLOTB	–
2	SLOTC	–
3	SLOTD	–

- **SDCBUS: SDCard/SDIO Bus Width**

Value	Name	Description
0	1	1 bit
1	–	Reserved
2	4	4 bits
3	8	8 bits

40.14.5 HSMCI Argument Register

Name: HSMCI_ARGR

Address: 0x40080010

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**

40.14.6 HSMCI Command Register

Name: HSMCI_CMDR

Address: 0x40080014

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 writable 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/SD Card Single Block
1	MULTIPLE	MMC/SD Card Multiple Block
2	STREAM	MMC Stream
4	BYTE	SDIO Byte
5	BLOCK	SDIO Block

- **IOSPCMD: SDIO Special Command**

Value	Name	Description
0	STD	Not an SDIO Special Command
1	SUSPEND	SDIO Suspend Command
2	RESUME	SDIO Resume Command

- **ATACS: ATA with Command Completion Signal**

0 (NORMAL): Normal operation mode.

1 (COMPLETION): This bit indicates that a completion signal is expected within a programmed amount of time (HSMCI_CSTOR).

- **BOOT_ACK: Boot Operation Acknowledge**

The master can choose to receive the boot acknowledge from the slave when a Boot Request command is issued. When set to one this field indicates that a Boot acknowledge is expected within a programmable amount of time defined with DTOMUL and DTOCYC fields located in the HSMCI_DTOR. 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.

40.14.7 HSMCI Block Register

Name: HSMCI_BLKR

Address: 0x40080018

Access: Read/Write

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

- **BCNT: MMC/SDIO Block Count - SDIO Byte Count**

This field determines the number of data byte(s) or block(s) to transfer.

The transfer data type and the authorized values for BCNT field are determined by the TRTYP field in the HSMCI Command Register (HSMCI_CMDR).

When TRTYP = 1 (MMC/SDCARD Multiple Block), BCNT can be programmed from 1 to 65535, 0 corresponds to an infinite block transfer.

When TRTYP = 4 (SDIO Byte), BCNT can be programmed from 1 to 511, 0 corresponds to 512-byte transfer. Values in range 512 to 65536 are forbidden.

When TRTYP = 5 (SDIO Block), BCNT can be programmed from 1 to 511, 0 corresponds to an infinite block transfer. Values in range 512 to 65536 are forbidden.

Warning: In SDIO Byte and Block modes (TRTYP = 4 or 5), writing the 7 last bits of BCNT field with a value which differs from 0 is forbidden and may lead to unpredictable results.

- **BLKLEN: Data Block Length**

This field determines the size of the data block.

Bits 16 and 17 must be configured to 0 if FBYTE is disabled.

Note: In SDIO Byte mode, BLKLEN field is not used.

40.14.8 HSMCI Completion Signal Timeout Register

Name: HSMCI_CSTOR

Address: 0x4008001C

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the HSMCI Write Protection Mode Register.

- **CSTOCYC: Completion Signal Timeout Cycle Number**

This field determines 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**

This field determines the maximum number of Master Clock cycles that the HSMCI waits between two data block transfers. Its value is calculated by (CSTOCYC x Multiplier).

These fields determine the maximum number of Master Clock cycles that the HSMCI waits between the end of the data transfer and the assertion of the completion signal. The data transfer comprises data phase and the optional busy phase. If a non-DATA ATA command is issued, the HSMCI starts waiting immediately after the end of the response until the completion signal.

Multiplier is defined by CSTOMUL as shown in the following table:

Value	Name	Description
0	1	CSTOCYC x 1
1	16	CSTOCYC x 16
2	128	CSTOCYC x 128
3	256	CSTOCYC x 256
4	1024	CSTOCYC x 1024
5	4096	CSTOCYC x 4096
6	65536	CSTOCYC x 65536
7	1048576	CSTOCYC x 1048576

If the data time-out set by CSTOCYC and CSTOMUL has been exceeded, the Completion Signal Time-out Error flag (CSTOE) in the HSMCI Status Register (HSMCI_SR) rises.

40.14.9 HSMCI Response Register

Name: HSMCI_RSPR

Address: 0x40080020

Access: Read-only

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

- **RSP: Response**

Note: 1. The response register can be read by N accesses at the same HSMCI_RSPR or at consecutive addresses (0x20 to 0x2C). N depends on the size of the response.

40.14.10 HSMCI Receive Data Register

Name: HSMCI_RDR

Address: 0x40080030

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**

40.14.11 HSMCI Transmit Data Register

Name: HSMCI_TDR

Address: 0x40080034

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**

40.14.12 HSMCI Status Register

Name: HSMCI_SR

Address: 0x40080040

Access: Read-only

31	30	29	28	27	26	25	24
UNRE	OVRE	ACKRCVE	ACKRCV	XFRDONE	FIFOEMPTY	–	–
23	22	21	20	19	18	17	16
CSTOE	DTOE	DCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	CSRCV	SDIOWAIT	–	–	–	SDIOIRQA
7	6	5	4	3	2	1	0
ENDTX	ENDRX	NOTBUSY	DTIP	BLKE	TXRDY	RXRDY	CMDRDY

- **CMDRDY: Command Ready (cleared by writing in HSMCI_CMDR)**

0: A command is in progress.

1: The last command has been sent.

- **RXRDY: Receiver Ready (cleared by reading HSMCI_RDR)**

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 (cleared by writing in HSMCI_TDR)**

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 (cleared on read)**

This flag must be used only for Write Operations.

0: A data block transfer is not yet finished.

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 (cleared at the end of CRC16 calculation)**

0: No data transfer in progress.

1: The current data transfer is still in progress, including CRC16 calculation.

- **NOTBUSY: HSMCI Not Busy**

A block write operation uses a simple busy signalling of the write operation duration on the data (DAT0) line: during a data transfer block, if the card does not have a free data receive buffer, the card indicates this condition by pulling down the data line (DAT0) to LOW. The card stops pulling down the data line as soon as at least one receive buffer for the defined data transfer block length becomes free.

Refer to the MMC or SD Specification for more details concerning the busy behavior.

For all the read operations, the NOTBUSY flag is cleared at the end of the host command.
For the Infinite Read Multiple Blocks, the NOTBUSY flag is set at the end of the STOP_TRANSMISSION host command (CMD12).

For the Single Block Reads, the NOTBUSY flag is set at the end of the data read block.

For the Multiple Block Reads with predefined block count, the NOTBUSY flag is set at the end of the last received data block.

The NOTBUSY flag allows to deal with these different states.

0: The HSMCI is not ready for new data transfer. Cleared at the end of the card response.

1: The HSMCI is ready for new data transfer. Set when the busy state on the data line has ended. This corresponds to a free internal data receive buffer of the card.

- **ENDRX: End of RX Buffer (cleared by writing HSMCI_RCR or HSMCI_RNCR(1))**

0: The Receive Counter Register has not reached 0 since the last write in HSMCI_RCR or HSMCI_RNCR.

1: The Receive Counter Register has reached 0 since the last write in HSMCI_RCR or HSMCI_RNCR.

- **ENDTX: End of TX Buffer (cleared by writing HSMCI_TCR or HSMCI_TNCR(1))**

0: The Transmit Counter Register has not reached 0 since the last write in HSMCI_TCR or HSMCI_TNCR.

1: The Transmit Counter Register has reached 0 since the last write in HSMCI_TCR or HSMCI_TNCR.

Note: BLKE and NOTBUSY flags can be used to check that the data has been successfully transmitted on the data lines and not only transferred from the PDC to the HSMCI Controller.

- **SDIOIRQA: SDIO Interrupt for Slot A (cleared on read)**

0: No interrupt detected on SDIO Slot A.

1: An SDIO Interrupt on Slot A occurred.

- **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 (cleared on read)**

0: No completion signal received since last status read operation.

1: The device has issued a command completion signal on the command line.

- **RXBUFF: RX Buffer Full (cleared by writing HSMCI_RCR or HSMCI_RNCR(1))**

0: HSMCI_RCR or HSMCI_RNCR has a value other than 0.

1: Both HSMCI_RCR and HSMCI_RNCR have a value of 0.

- **TXBUFE: TX Buffer Empty (cleared by writing HSMCI_TCR or HSMCI_TNCR(1))**

0: HSMCI_TCR or HSMCI_TNCR has a value other than 0.

1: Both HSMCI_TCR and HSMCI_TNCR have a value of 0.

Note: BLKE and NOTBUSY flags can be used to check that the data has been successfully transmitted on the data lines and not only transferred from the PDC to the HSMCI Controller.

- **RINDE: Response Index Error (cleared by writing in HSMCI_CMDR)**

0: No error.

1: A mismatch is detected between the command index sent and the response index received.

- **RDIRE: Response Direction Error (cleared by writing in HSMCI_CMDR)**

0: No error.

1: The direction bit from card to host in the response has not been detected.

- **RCRCE: Response CRC Error (cleared by writing in HSMCI_CMDR)**

0: No error.

1: A CRC7 error has been detected in the response.

- **RENDE: Response End Bit Error (cleared by writing in HSMCI_CMDR)**

0: No error.

1: The end bit of the response has not been detected.

- **RTOE: Response Time-out Error (cleared by writing in HSMCI_CMDR)**

0: No error.

1: The response time-out set by MAXLAT in the HSMCI_CMDR has been exceeded.

- **DCRCE: Data CRC Error (cleared on read)**

0: No error.

1: A CRC16 error has been detected in the last data block.

- **DTOE: Data Time-out Error (cleared on read)**

0: No error.

1: The data time-out set by DTOCYC and DTOMUL in HSMCI_DTOR has been exceeded.

- **CSTOE: Completion Signal Time-out Error (cleared on read)**

0: No error.

1: The completion signal time-out set by CSTOCYC and CSTOMUL in HSMCI_CSTOR has been exceeded.

- **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 (cleared on read)**

0: No Boot acknowledge received since the last read of the HSMCI_SR.

1: A Boot acknowledge signal has been received since the last read of HSMCI_SR.

- **ACKRCVE: Boot Operation Acknowledge Error (cleared on read)**

0: No boot operation error since the last read of HSMCI_SR

1: Corrupted Boot Acknowledge signal received since the last read of HSMCI_SR.

- **OVRE: Overrun (if FERRCTRL = 1, cleared by writing in HSMCI_CMDR or cleared on read if FERRCTRL = 0)**

0: No error.

1: At least one 8-bit received data has been lost (not read).

If FERRCTRL = 1 in HSMCI_CFG, OVRE is cleared on read.

If FERRCTRL = 0 in HSMCI_CFG, OVRE is cleared by writing HSMCI_CMDR.

- **UNRE: Underrun (if FERRCTRL = 1, cleared by writing in HSMCI_CMDR or cleared on read if FERRCTRL = 0)**

0: No error.

1: At least one 8-bit data has been sent without valid information (not written).

If FERRCTRL = 1 in HSMCI_CFG, OVRE is cleared on read.

If FERRCTRL = 0 in HSMCI_CFG, OVRE is cleared by writing HSMCI_CMDR.

Note: 1. HSMCI_RCR, HSMCI_RNCR, HSMCI_TCR, HSMCI_TNCR are PDC registers.

40.14.13 HSMCI Interrupt Enable Register

Name: HSMCI_IER

Address: 0x40080044

Access: Write-only

31	30	29	28	27	26	25	24
UNRE	OVRE	ACKRCVE	ACKRCV	XFRDONE	FIFOEMPTY	–	–
23	22	21	20	19	18	17	16
CSTOE	DTOE	DCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	CSRCV	SDIOWAIT	–	–	–	SDIOIRQA
7	6	5	4	3	2	1	0
ENDTX	ENDRX	NOTBUSY	DTIP	BLKE	TXRDY	RXRDY	CMDRDY

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **CMDRDY: Command Ready Interrupt Enable**
- **RXRDY: Receiver Ready Interrupt Enable**
- **TXRDY: Transmit Ready Interrupt Enable**
- **BLKE: Data Block Ended Interrupt Enable**
- **DTIP: Data Transfer in Progress Interrupt Enable**
- **NOTBUSY: Data Not Busy Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable**
- **ENDTX: End of Transmit Buffer Interrupt Enable**
- **SDIOIRQA: SDIO Interrupt for Slot A Interrupt Enable**
- **SDIOWAIT: SDIO Read Wait Operation Status Interrupt Enable**
- **CSRCV: Completion Signal Received Interrupt Enable**
- **RXBUFF: Receive Buffer Full Interrupt Enable**
- **TXBUFE: Transmit Buffer Empty Interrupt Enable**
- **RINDE: Response Index Error Interrupt Enable**
- **RDIRE: Response Direction Error Interrupt Enable**
- **RCRCE: Response CRC Error Interrupt Enable**
- **RENDE: Response End Bit Error Interrupt Enable**

- **RTOE: Response Time-out Error Interrupt Enable**
- **DCRCE: Data CRC Error Interrupt Enable**
- **DTOE: Data Time-out Error Interrupt Enable**
- **CSTOE: Completion Signal Timeout Error Interrupt Enable**
- **FIFOEMPTY: FIFO empty Interrupt enable**
- **XFRDONE: Transfer Done Interrupt enable**
- **ACKRCV: Boot Acknowledge Interrupt Enable**
- **ACKRCVE: Boot Acknowledge Error Interrupt Enable**
- **OVRE: Overrun Interrupt Enable**
- **UNRE: Underrun Interrupt Enable**

40.14.14 HSMCI Interrupt Disable Register

Name: HSMCI_IDR

Address: 0x40080048

Access: Write-only

31	30	29	28	27	26	25	24
UNRE	OVRE	ACKRCVE	ACKRCV	XFRDONE	FIFOEMPTY	–	–
23	22	21	20	19	18	17	16
CSTOE	DTOE	DCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	CSRCV	SDIOWAIT	–	–	–	SDIOIRQA
7	6	5	4	3	2	1	0
ENDTX	ENDRX	NOTBUSY	DTIP	BLKE	TXRDY	RXRDY	CMDRDY

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Disables the corresponding interrupt.

- **CMDRDY: Command Ready Interrupt Disable**
- **RXRDY: Receiver Ready Interrupt Disable**
- **TXRDY: Transmit Ready Interrupt Disable**
- **BLKE: Data Block Ended Interrupt Disable**
- **DTIP: Data Transfer in Progress Interrupt Disable**
- **NOTBUSY: Data Not Busy Interrupt Disable**
- **ENDRX: End of Receive Buffer Interrupt Disable**
- **ENDTX: End of Transmit Buffer Interrupt Disable**
- **SDIOIRQA: SDIO Interrupt for Slot A Interrupt Disable**
- **SDIOWAIT: SDIO Read Wait Operation Status Interrupt Disable**
- **CSRCV: Completion Signal received interrupt Disable**
- **RXBUFF: Receive Buffer Full Interrupt Disable**
- **TXBUFE: Transmit Buffer Empty Interrupt Disable**
- **RINDE: Response Index Error Interrupt Disable**
- **RDIRE: Response Direction Error Interrupt Disable**
- **RCRCE: Response CRC Error Interrupt Disable**
- **RENDE: Response End Bit Error Interrupt Disable**

- **RTOE: Response Time-out Error Interrupt Disable**
- **DCRCE: Data CRC Error Interrupt Disable**
- **DTOE: Data Time-out Error Interrupt Disable**
- **CSTOE: Completion Signal Time out Error Interrupt Disable**
- **FIFOEMPTY: FIFO empty Interrupt Disable**
- **XFRDONE: Transfer Done Interrupt Disable**
- **ACKRCV: Boot Acknowledge Interrupt Disable**
- **ACKRCVE: Boot Acknowledge Error Interrupt Disable**
- **OVRE: Overrun Interrupt Disable**
- **UNRE: Underrun Interrupt Disable**

40.14.15 HSMCI Interrupt Mask Register

Name: HSMCI_IMR

Address: 0x4008004C

Access: Read-only

31	30	29	28	27	26	25	24
UNRE	OVRE	ACKRCVE	ACKRCV	XFRDONE	FIFOEMPTY	–	–
23	22	21	20	19	18	17	16
CSTOE	DTOE	DCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	CSRCV	SDIOWAIT	–	–	–	SDIOIRQA
7	6	5	4	3	2	1	0
ENDTX	ENDRX	NOTBUSY	DTIP	BLKE	TXRDY	RXRDY	CMDRDY

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

- **CMDRDY: Command Ready Interrupt Mask**
- **RXRDY: Receiver Ready Interrupt Mask**
- **TXRDY: Transmit Ready Interrupt Mask**
- **BLKE: Data Block Ended Interrupt Mask**
- **DTIP: Data Transfer in Progress Interrupt Mask**
- **NOTBUSY: Data Not Busy Interrupt Mask**
- **ENDRX: End of Receive Buffer Interrupt Mask**
- **ENDTX: End of Transmit Buffer Interrupt Mask**
- **SDIOIRQA: SDIO Interrupt for Slot A Interrupt Mask**
- **SDIOWAIT: SDIO Read Wait Operation Status Interrupt Mask**
- **CSRCV: Completion Signal Received Interrupt Mask**
- **RXBUFF: Receive Buffer Full Interrupt Mask**
- **TXBUFE: Transmit Buffer Empty Interrupt Mask**
- **RINDE: Response Index Error Interrupt Mask**
- **RDIRE: Response Direction Error Interrupt Mask**
- **RCRCE: Response CRC Error Interrupt Mask**
- **RENDE: Response End Bit Error Interrupt Mask**

- **RTOE: Response Time-out Error Interrupt Mask**
- **DCRCE: Data CRC Error Interrupt Mask**
- **DTOE: Data Time-out Error Interrupt Mask**
- **CSTOE: Completion Signal Time-out Error Interrupt Mask**
- **FIFOEMPTY: FIFO Empty Interrupt Mask**
- **XFRDONE: Transfer Done Interrupt Mask**
- **ACKRCV: Boot Operation Acknowledge Received Interrupt Mask**
- **ACKRCVE: Boot Operation Acknowledge Error Interrupt Mask**
- **OVRE: Overrun Interrupt Mask**
- **UNRE: Underrun Interrupt Mask**

40.14.16 HSMCI Configuration Register

Name: HSMCI_CFG

Address: 0x40080054

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the HSMCI Write Protection Mode Register.

- **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).

40.14.17 HSMCI Write Protection Mode Register

Name: HSMCI_WPMR

Address: 0x400800E4

Access: Read/Write

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

- **WPEN: Write Protect Enable**

0: Disables the Write Protection if WPKEY corresponds to 0x4D4349 (“MCI” in ASCII).

1: Enables the Write Protection if WPKEY corresponds to 0x4D4349 (“MCI” in ASCII).

See Section 40.13 “Register Write Protection” for the list of registers that can be write-protected.

- **WPKEY: Write Protect Key**

Value	Name	Description
0x4D4349	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

40.14.18 HSMCI Write Protection Status Register

Name: HSMCI_WPSR

Address: 0x400800E8

Access: Read-only

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

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of the HSMCI_WPSR.

1: A write protection violation has occurred since the last read of the HSMCI_WPSR. 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**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

40.14.19 HSMCI FIFOx Memory Aperture

Name: HSMCI_FIFOx [x=0..255]

Address: 0x40080200

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**

41. USB Device Port (UDP)

41.1 Description

The USB Device Port (UDP) is compliant with the Universal Serial Bus (USB) 2.0 full-speed device specification.

Each endpoint can be configured in one of several USB transfer types. It can be associated with one or two banks of a dual-port RAM used to store the current data payload. If two banks are used, one DPR bank is read or written by the processor, while the other is read or written by the USB device peripheral. This feature is mandatory for isochronous endpoints. Thus the device maintains the maximum bandwidth (1 Mbyte/s) by working with endpoints with two banks of DPR.

Table 41-1. USB Endpoint Description

Endpoint No.	Mnemonic	Dual-Bank(1)	Max. Endpoint Size	Endpoint Type
0	EP0	No	64	Control/Bulk/Interrupt
1	EP1	Yes	64	Bulk/Iso/Interrupt
2	EP2	Yes	64	Bulk/Iso/Interrupt
3	EP3	No	64	Control/Bulk/Interrupt
4	EP4	Yes	512	Bulk/Iso/Interrupt
5	EP5	Yes	512	Bulk/Iso/Interrupt
6	EP6	Yes	64	Bulk/Iso/Interrupt
7	EP7	Yes	64	Bulk/Iso/Interrupt

Note: 1. The Dual-Bank function provides two banks for an endpoint. This feature is used for ping-pong mode.

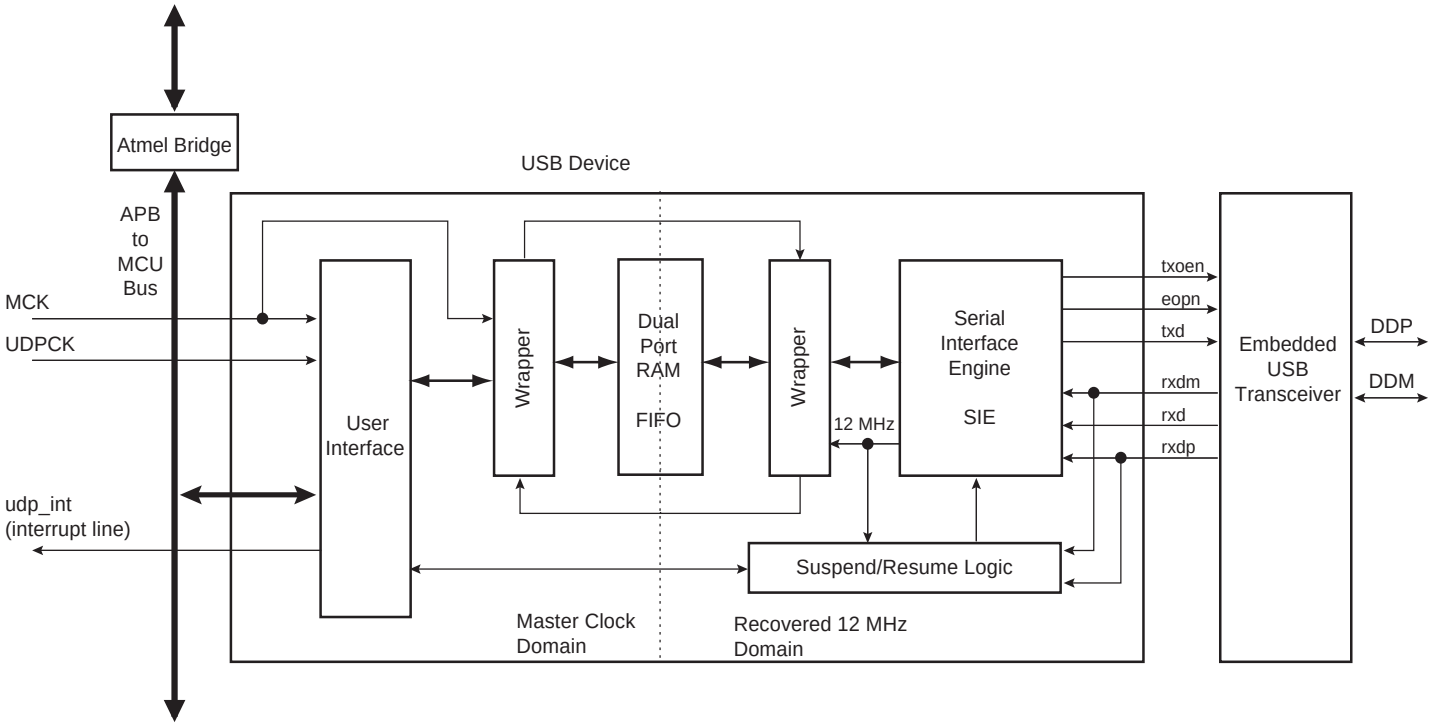
Suspend and resume are automatically detected by the USB device, which notifies the processor by raising an interrupt. Depending on the product, an external signal can be used to send a wakeup request to the USB host controller.

41.2 Embedded Characteristics

- USB 2.0 Full-speed Compliant, 12 Mbit/s
- Embedded USB 2.0 Full-speed Transceiver
- Integrated Pull-up on DDP
- Integrated Pull-down on DDM
- 8 Endpoints
- Embedded Dual-port RAM for Endpoints
- Suspend/Resume Logic
- Ping-pong Mode (2 Memory Banks) for Isochronous and Bulk Endpoints

41.3 Block Diagram

Figure 41-1. Block Diagram



Access to the UDP is via the APB bus interface. Read and write to the data FIFO are done by reading and writing 8-bit values to APB registers.

The UDP peripheral requires two clocks: one peripheral clock used by the Master Clock domain (MCK) and a 48 MHz clock (UDPCK) used by the 12 MHz domain.

A USB 2.0 full-speed pad is embedded and controlled by the Serial Interface Engine (SIE).

The signal external_resume is optional. It allows the UDP peripheral to wake up once in system mode. The host is then notified that the device asks for a resume. This optional feature must also be negotiated with the host during the enumeration.

41.3.1 Signal Description

Table 41-2. Signal Names

Signal Name	Description	Type
UDPCK	48 MHz clock	Input
MCK	Master clock	Input
udp_int	Interrupt line connected to the Interrupt Controller	Input
DDP	USB D+ line	I/O
DDM	USB D- line	I/O

41.4 Product Dependencies

For further details on the USB Device hardware implementation, see the specific Product Properties document.

The USB physical transceiver is integrated into the product. The bidirectional differential signals DDP and DDM are available from the product boundary.

One I/O line may be used by the application to check that VBUS is still available from the host. Self-powered devices may use this entry to be notified that the host has been powered off. In this case, the pull-up on DDP must be disabled in order to prevent feeding current to the host. The application should disconnect the transceiver, then remove the pull-up.

41.4.1 I/O Lines

The USB pins are shared with PIO lines. By default, the USB function is activated, and pins DDP and DDM are used for USB. To configure DDP or DDM as PIOs, the user needs to configure the system I/O configuration register (CCFG_SYSIO) in the MATRIX.

41.4.2 Power Management

The USB device peripheral requires a 48 MHz clock. This clock must be generated by a PLL driven by a clock source with an accuracy of ± 0.25% (note that the fast RC oscillator cannot be used).

Thus, the USB device receives two clocks from the Power Management Controller (PMC): the master clock, MCK, used to drive the peripheral user interface, and the UDPCK, used to interface with the bus USB signals (recovered 12 MHz domain).

WARNING: The UDP peripheral clock in the Power Management Controller (PMC) must be enabled before any read/write operations to the UDP registers including the UDP_TXVC register.

41.4.3 Interrupt

The USB device interface has an interrupt line connected to the Interrupt Controller.

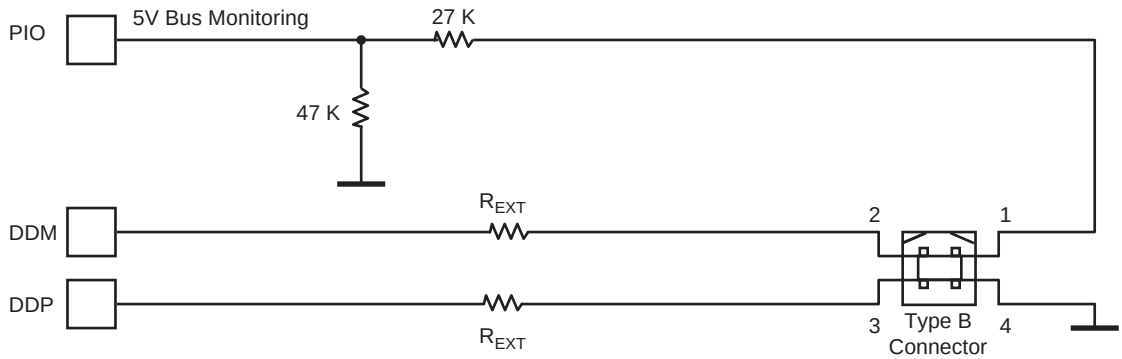
Handling the USB device interrupt requires programming the Interrupt Controller before configuring the UDP.

Table 41-3. Peripheral IDs

Instance	ID
UDP	35

41.5 Typical Connection

Figure 41-2. Board Schematic to Interface Device Peripheral



41.5.1 USB Device Transceiver

The USB device transceiver is embedded in the product. However, discrete components are required for each of the following actions:

- to monitor VBUS voltage
- for line termination
- to disconnect the host for reduced power consumption

41.5.2 VBUS Monitoring

VBUS monitoring is required to detect host connection. VBUS monitoring is done using a standard PIO with internal pull-up disabled. When the host is switched off, it should be considered as a disconnect, the pull-up must be disabled in order to prevent powering the host through the pull-up resistor.

When the host is disconnected and the transceiver is enabled, then DDP and DDM are floating. This may lead to over consumption. A solution is to enable the integrated pull-down by disabling the transceiver (UDP_TXVC.TXVDIS = 1) and then remove the pull-up (UDP_TXVC.PUON = 0).

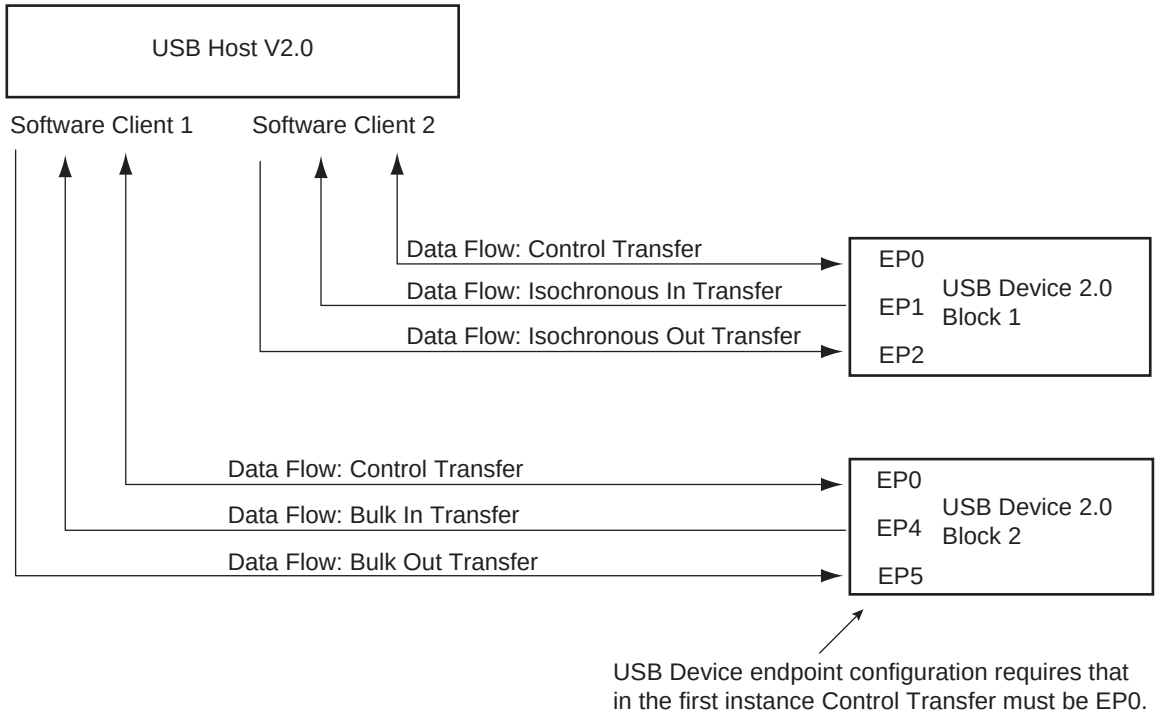
A termination serial resistor must be connected to DDP and DDM. The resistor value is defined in the electrical specification of the product (R_{EXT}).

41.6 Functional Description

41.6.1 USB 2.0 Full-speed Introduction

The USB 2.0 full-speed provides communication services between host and attached USB devices. Each device is offered with a collection of communication flows (pipes) associated with each endpoint. Software on the host communicates with a USB device through a set of communication flows.

Figure 41-3. Example of USB 2.0 Full-speed Communication Control



The Control Transfer endpoint EP0 is always used when a USB device is first configured (USB 2.0 specifications).

41.6.1.1 USB 2.0 Full-speed Transfer Types

A communication flow is carried over one of four transfer types defined by the USB device.

Table 41-4. USB Communication Flow

Transfer	Direction	Bandwidth	Supported Endpoint Size	Error Detection	Retrying
Control	Bidirectional	Not guaranteed	8, 16, 32, 64	Yes	Automatic
Isochronous	Unidirectional	Guaranteed	512	Yes	No
Interrupt	Unidirectional	Not guaranteed	≤ 64	Yes	Yes
Bulk	Unidirectional	Not guaranteed	8, 16, 32, 64	Yes	Yes

41.6.1.2 USB Bus Transactions

Each transfer results in one or more transactions over the USB bus. There are three kinds of transactions flowing across the bus in packets:

- Setup Transaction
- Data IN Transaction
- Data OUT Transaction

41.6.1.3 USB Transfer Event Definitions

As indicated below, transfers are sequential events carried out on the USB bus.

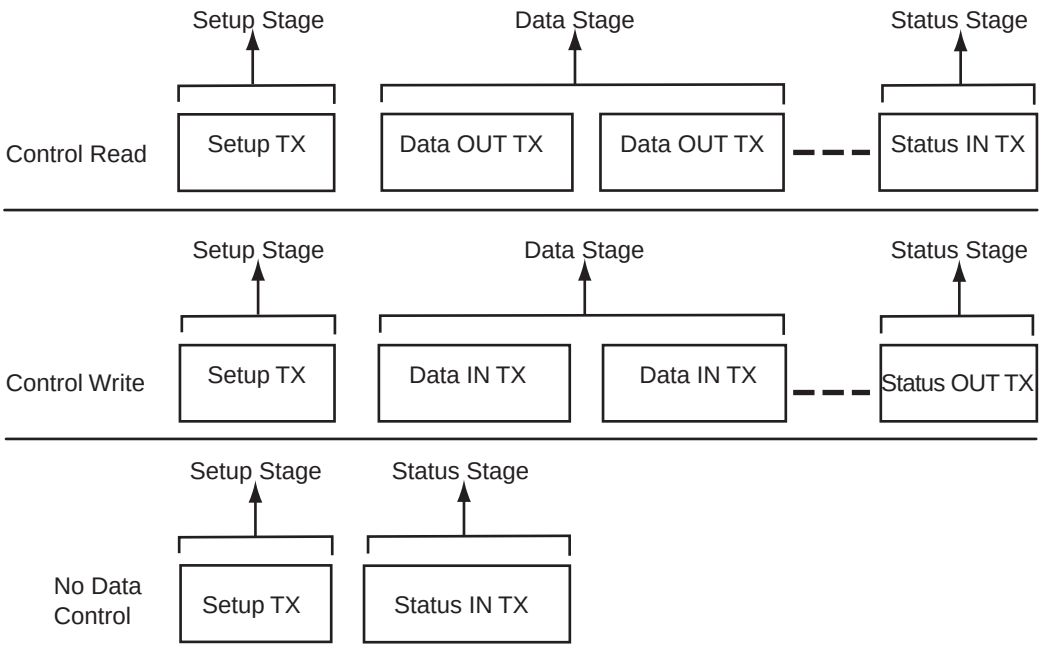
Table 41-5. USB Transfer Events

Transfer		Transaction
Direction	**Type**	
CONTROL (bidirectional)	Control[1][3]	Setup transaction → Data IN transactions → Status OUT transaction
		Setup transaction → Data OUT transactions → Status IN transaction
		Setup transaction → Status IN transaction
IN (device toward host)	Interrupt IN	Data IN transaction → Data IN transaction
	Isochronous IN[2]	
	Bulk IN	
OUT (host toward device)	Interrupt OUT	Data OUT transaction → Data OUT transaction
	Isochronous OUT[2]	
	Bulk OUT	

Notes:
1. Control transfer must use endpoints with no ping-pong attributes.
2. Isochronous transfers must use endpoints with ping-pong attributes.
3. Control transfers can be aborted using a stall handshake.

A status transaction is a special type of host-to-device transaction used only in a control transfer. The control transfer must be performed using endpoints with no ping-pong attributes. According to the control sequence (read or write), the USB device sends or receives a status transaction.

Figure 41-4. Control Read and Write Sequences



Notes: 1. During the Status IN stage, the host waits for a zero length packet (Data IN transaction with no data) from the device using DATA1 PID. Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev. 2.0,* for more information on the protocol layer.

2. During the Status OUT stage, the host emits a zero length packet to the device (Data OUT transaction with no data).

41.6.2 Handling Transactions with USB 2.0 Device Peripheral

41.6.2.1 Setup Transaction

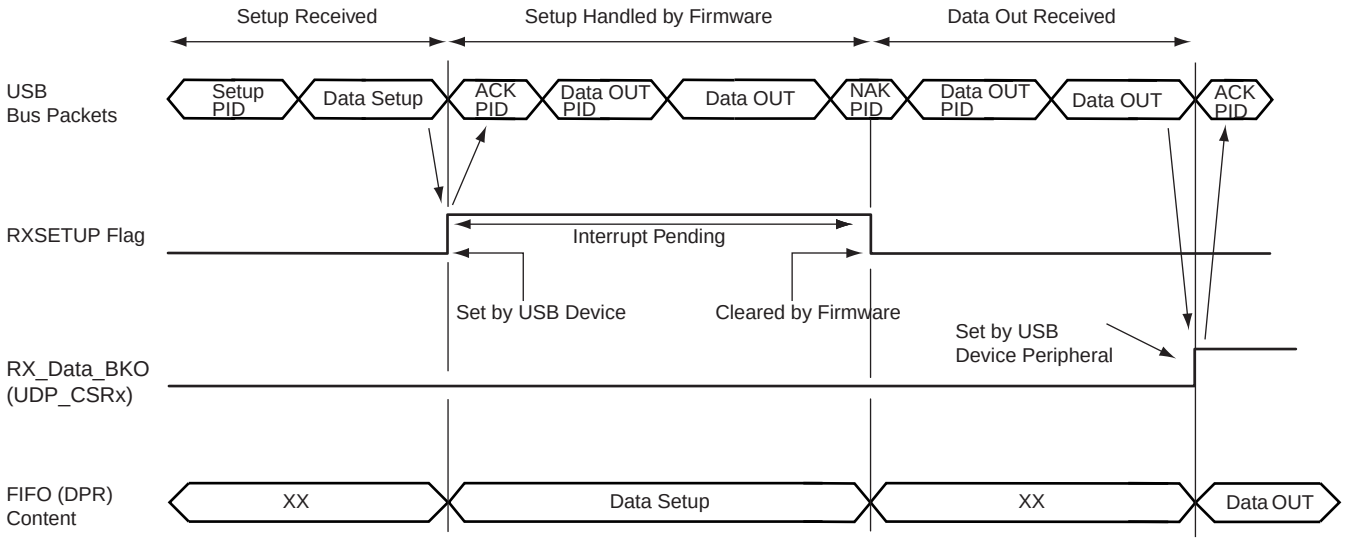
Setup is a special type of host-to-device transaction used during control transfers. Control transfers must be performed using endpoints with no ping-pong attributes. A setup transaction needs to be handled as soon as possible by the firmware. It is used to transmit requests from the host to the device. These requests are then handled by the USB device and may require more arguments. The arguments are sent to the device by a Data OUT transaction which follows the setup transaction. These requests may also return data. The data is carried out to the host by the next Data IN transaction which follows the setup transaction. A status transaction ends the control transfer.

When a setup transfer is received by the USB endpoint:

- The USB device automatically acknowledges the setup packet
- RXSETUP is set in the UDP_CSRx
- An endpoint interrupt is generated while the RXSETUP is not cleared. This interrupt is carried out to the microcontroller if interrupts are enabled for this endpoint.

Thus, firmware must detect the RXSETUP polling the UDP_CSRx or catching an interrupt, read the setup packet in the FIFO, then clear the RXSETUP. RXSETUP cannot be cleared before the setup packet has been read in the FIFO. Otherwise, the USB device would accept the next Data OUT transfer and overwrite the setup packet in the FIFO.

Figure 41-5. Setup Transaction Followed by a Data OUT Transaction



41.6.2.2 Data IN Transaction

Data IN transactions are used in control, isochronous, bulk and interrupt transfers and conduct the transfer of data from the device to the host. Data IN transactions in isochronous transfer must be done using endpoints with ping-pong attributes.

Using Endpoints Without Ping-pong Attributes

To perform a Data IN transaction using a non ping-pong endpoint:

1. The application checks if it is possible to write in the FIFO by polling TXPKTRDY in the endpoint's UDP_CSRx (TXPKTRDY must be cleared).
2. The application writes the first packet of data to be sent in the endpoint's FIFO, writing zero or more byte values in the endpoint's UDP_FDRx.
3. The application notifies the USB peripheral it has finished by setting the TXPKTRDY in the endpoint's UDP_CSRx.
4. The application is notified that the endpoint's FIFO has been released by the USB device when TXCOMP in the endpoint's UDP_CSRx has been set. Then an interrupt for the corresponding endpoint is pending while TXCOMP is set.
5. The microcontroller writes the second packet of data to be sent in the endpoint's FIFO, writing zero or more byte values in the endpoint's UDP_FDRx.
6. The microcontroller notifies the USB peripheral it has finished by setting the TXPKTRDY in the endpoint's UDP_CSRx.
7. The application clears the TXCOMP in the endpoint's UDP_CSRx.

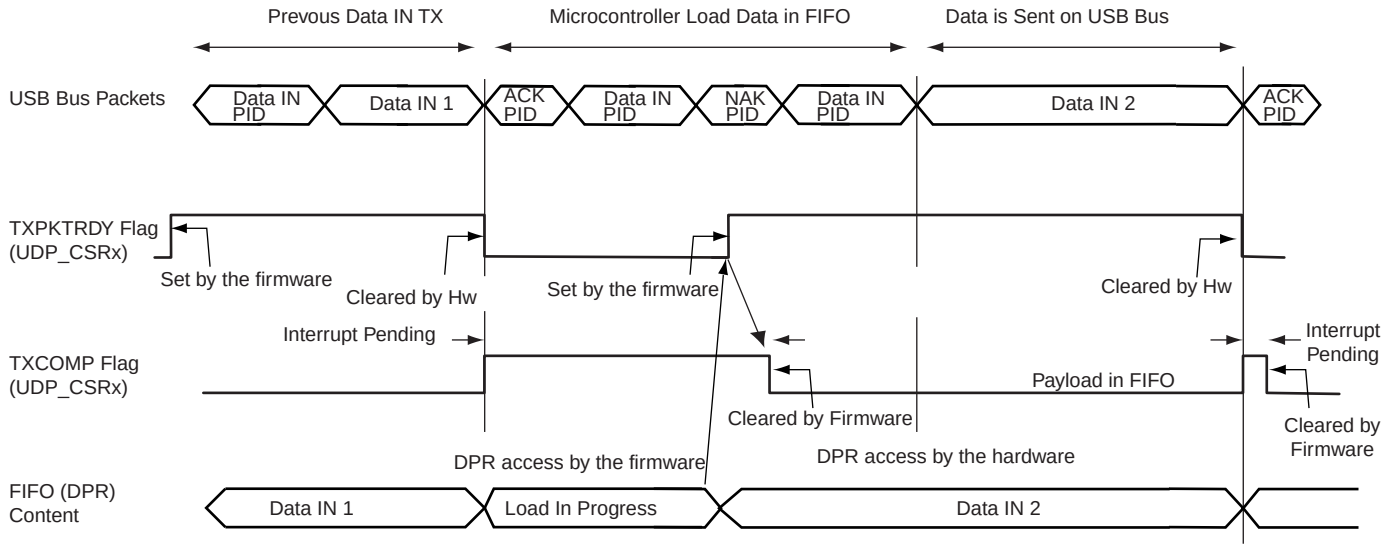
After the last packet has been sent, the application must clear TXCOMP once this has been set.

TXCOMP is set by the USB device when it has received an ACK PID signal for the Data IN packet. An interrupt is pending while TXCOMP is set.

Warning: TX_COMP must be cleared after TX_PKTRDY has been set.

Note: Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev 2.0,* for more information on the Data IN protocol layer.

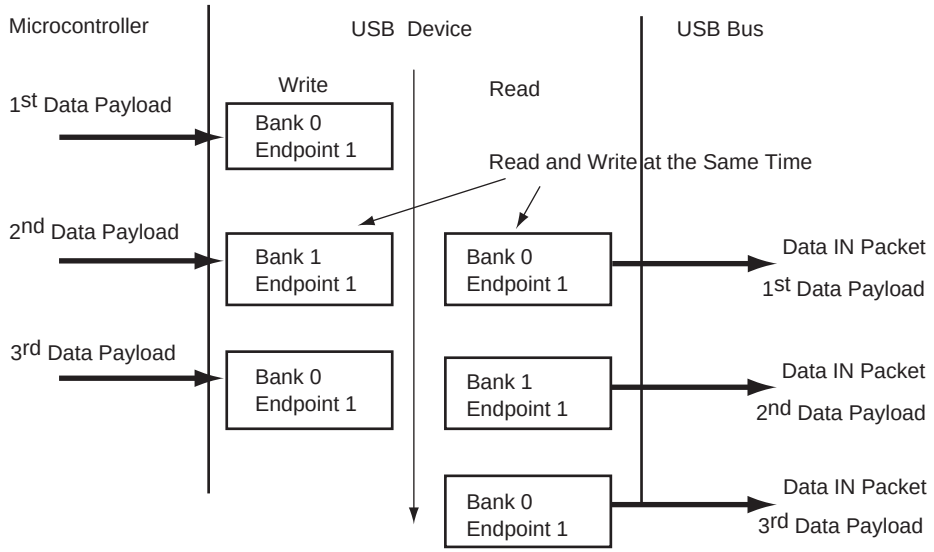
Figure 41-6. Data IN Transfer for Non Ping-pong Endpoint



Using Endpoints With Ping-pong Attribute

The use of an endpoint with ping-pong attributes is necessary during isochronous transfer. This also allows handling the maximum bandwidth defined in the USB specification during bulk transfer. To be able to guarantee a constant or the maximum bandwidth, the microcontroller must prepare the next data payload to be sent while the current one is being sent by the USB device. Thus two banks of memory are used. While one is available for the microcontroller, the other one is locked by the USB device.

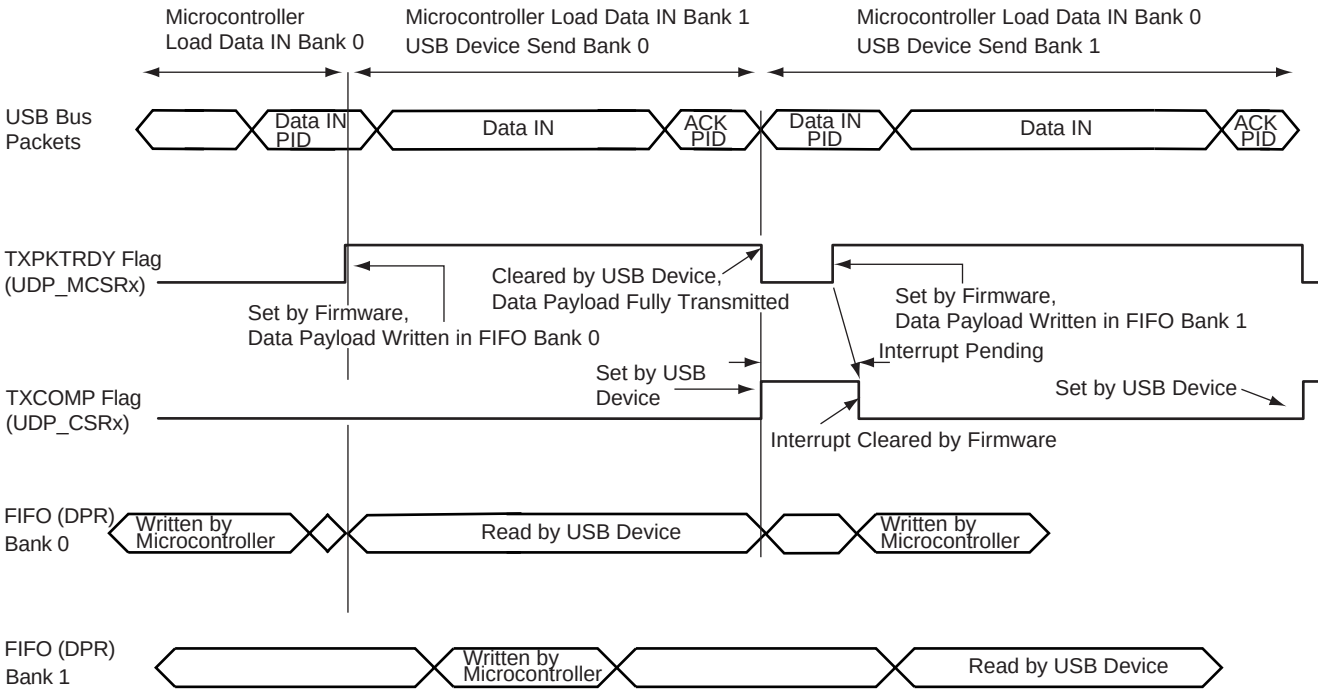
Figure 41-7. Bank Swapping Data IN Transfer for Ping-pong Endpoints



When using a ping-pong endpoint, the following procedures are required to perform Data IN transactions:

1. The microcontroller checks if it is possible to write in the FIFO by polling TXPKTRDY to be cleared in the endpoint's UDP_CSRx.
2. The microcontroller writes the first data payload to be sent in the FIFO (Bank 0), writing zero or more byte values in the endpoint's UDP_FDRx.
3. The microcontroller notifies the USB peripheral it has finished writing in Bank 0 of the FIFO by setting the TXPKTRDY in the endpoint's UDP_CSRx.
4. Without waiting for TXPKTRDY to be cleared, the microcontroller writes the second data payload to be sent in the FIFO (Bank 1), writing zero or more byte values in the endpoint's UDP_FDRx.
5. The microcontroller is notified that the first Bank has been released by the USB device when TXCOMP in the endpoint's UDP_CSRx is set. An interrupt is pending while TXCOMP is being set.
6. Once the microcontroller has received TXCOMP for the first Bank, it notifies the USB device that it has prepared the second Bank to be sent, raising TXPKTRDY in the endpoint's UDP_CSRx.
7. At this step, Bank 0 is available and the microcontroller can prepare a third data payload to be sent.

Figure 41-8. Data IN Transfer for Ping-pong Endpoint



Warning: There is software critical path due to the fact that once the second bank is filled, the driver has to wait for TX_COMP to set TX_PKTRDY. If the delay between receiving TX_COMP is set and TX_PKTRDY is set too long, some Data IN packets may be NACKed, reducing the bandwidth.

Warning: TX_COMP must be cleared after TX_PKTRDY has been set.

41.6.2.3 Data OUT Transaction

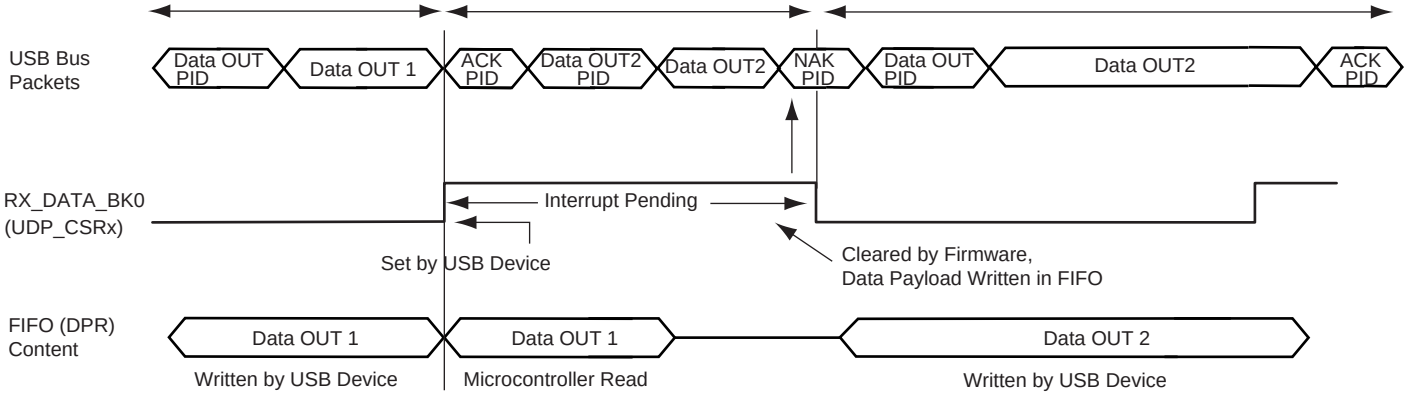
Data OUT transactions are used in control, isochronous, bulk and interrupt transfers and conduct the transfer of data from the host to the device. Data OUT transactions in isochronous transfers must be done using endpoints with ping-pong attributes.

Data OUT Transaction Without Ping-pong Attributes

To perform a Data OUT transaction, using a non ping-pong endpoint:

1. The host generates a Data OUT packet.
2. This packet is received by the USB device endpoint. While the FIFO associated to this endpoint is being used by the microcontroller, a NAK PID is returned to the host. Once the FIFO is available, data are written to the FIFO by the USB device and an ACK is automatically carried out to the host.
3. The microcontroller is notified that the USB device has received a data payload polling RX_DATA_BK0 in the endpoint's UDP_CSRx. An interrupt is pending for this endpoint while RX_DATA_BK0 is set.
4. The number of bytes available in the FIFO is made available by reading RXBYTECNT in the endpoint's UDP_CSRx.
5. The microcontroller carries out data received from the endpoint's memory to its memory. Data received is available by reading the endpoint's UDP_FDRx.
6. The microcontroller notifies the USB device that it has finished the transfer by clearing RX_DATA_BK0 in the endpoint's UDP_CSRx.
7. A new Data OUT packet can be accepted by the USB device.

Figure 41-9. Data OUT Transfer for Non Ping-pong Endpoints

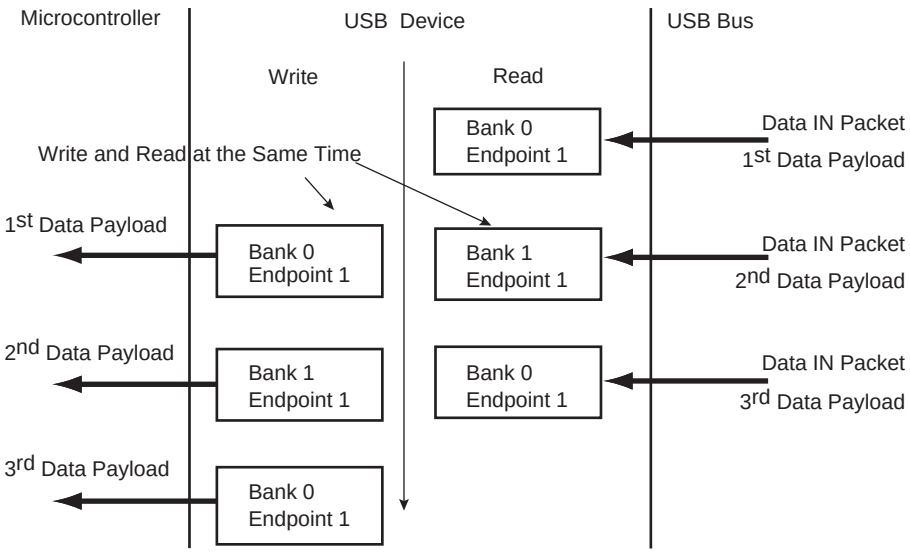


An interrupt is pending while the flag RX_DATA_BK0 is set. Memory transfer between the USB device, the FIFO and microcontroller memory is not possible after RX_DATA_BK0 has been cleared. Otherwise, the USB device would accept the next Data OUT transfer and overwrite the current Data OUT packet in the FIFO.

Using Endpoints With Ping-pong Attributes

During isochronous transfer, using an endpoint with ping-pong attributes is obligatory. To be able to guarantee a constant bandwidth, the microcontroller must read the previous data payload sent by the host, while the current data payload is received by the USB device. Thus two banks of memory are used. While one is available for the microcontroller, the other one is locked by the USB device.

Figure 41-10. Bank Swapping in Data OUT Transfers for Ping-pong Endpoints

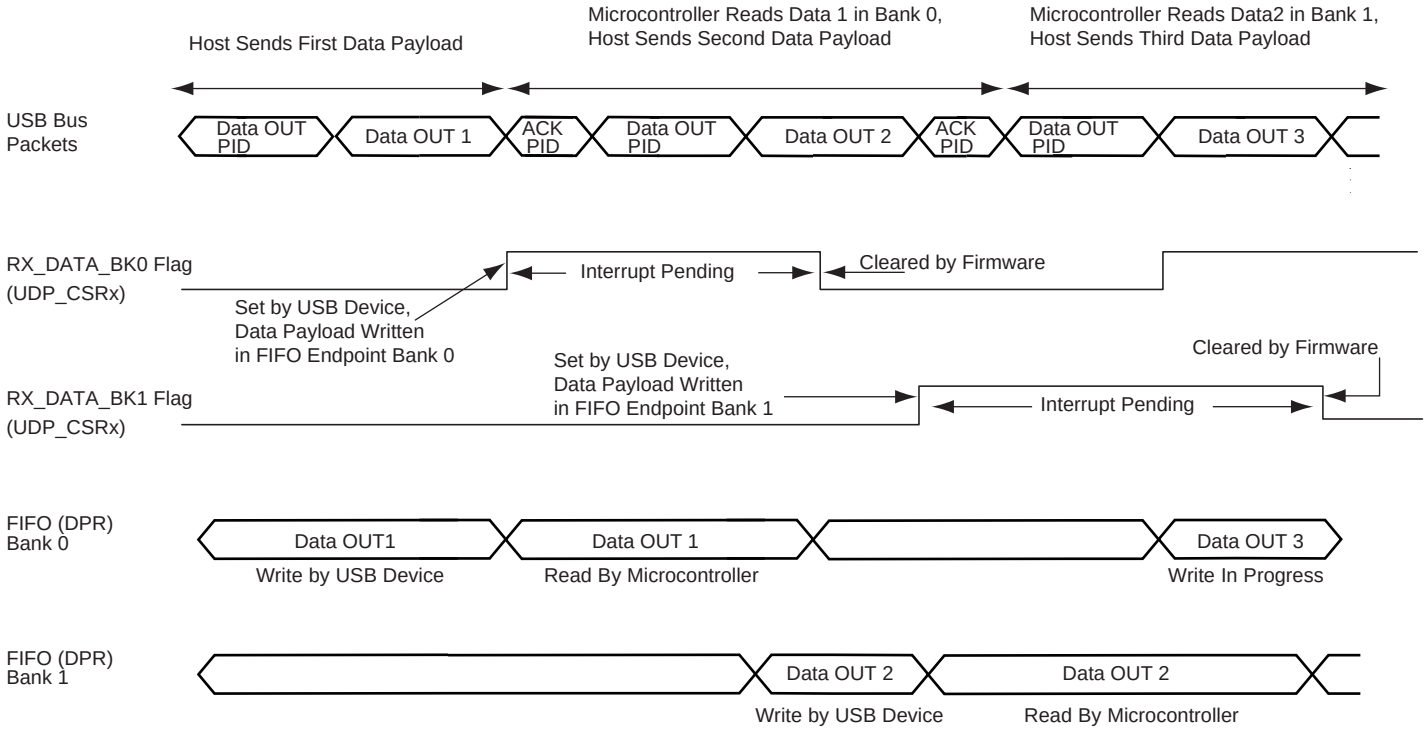


When using a ping-pong endpoint, the following procedures are required to perform Data OUT transactions:

1. The host generates a Data OUT packet.
2. This packet is received by the USB device endpoint. It is written in the endpoint's FIFO Bank 0.
3. The USB device sends an ACK PID packet to the host. The host can immediately send a second Data OUT packet. It is accepted by the device and copied to FIFO Bank 1.
4. The microcontroller is notified that the USB device has received a data payload, polling RX_DATA_BK0 in the endpoint's UDP_CSRx. An interrupt is pending for this endpoint while RX_DATA_BK0 is set.

5. The number of bytes available in the FIFO is made available by reading RXBYTECNT in the endpoint's UDP_CSRx.
6. The microcontroller transfers out data received from the endpoint's memory to the microcontroller's memory. Data received is made available by reading the endpoint's UDP_FDRx.
7. The microcontroller notifies the USB peripheral device that it has finished the transfer by clearing RX_DATA_BK0 in the endpoint's UDP_CSRx.
8. A third Data OUT packet can be accepted by the USB peripheral device and copied in the FIFO Bank 0.
9. If a second Data OUT packet has been received, the microcontroller is notified by the flag RX_DATA_BK1 set in the endpoint's UDP_CSRx. An interrupt is pending for this endpoint while RX_DATA_BK1 is set.
10. The microcontroller transfers out data received from the endpoint's memory to the microcontroller's memory. Data received is available by reading the endpoint's UDP_FDRx.
11. The microcontroller notifies the USB device it has finished the transfer by clearing RX_DATA_BK1 in the endpoint's UDP_CSRx.
12. A fourth Data OUT packet can be accepted by the USB device and copied in the FIFO Bank 1.

Figure 41-11. Data OUT Transfer for Ping-pong Endpoint



Note: An interrupt is pending while the RX_DATA_BK0 or RX_DATA_BK1 flag is set.

Warning: When RX_DATA_BK0 and RX_DATA_BK1 are both set, there is no way to determine which one to clear first. Thus the software must keep an internal counter to be sure to clear alternatively RX_DATA_BK0 then RX_DATA_BK1. This situation may occur when the software application is busy elsewhere and the two banks are filled by the USB host. Once the application comes back to the USB driver, the two flags are set.

41.6.2.4 Stall Handshake

A stall handshake can be used in one of two distinct occasions. (For more information on the stall handshake, refer to Chapter 8 of the *Universal Serial Bus Specification, Rev 2.0.*)

- A functional stall is used when the halt feature associated with the endpoint is set. (Refer to Chapter 9 of the *Universal Serial Bus Specification, Rev 2.0,* for more information on the halt feature.)

- To abort the current request, a protocol stall is used, but uniquely with control transfer.

The following procedure generates a stall packet:

1. The microcontroller sets the FORCESTALL flag in the UDP_CSRx endpoint's register.
2. The host receives the stall packet.
3. The microcontroller is notified that the device has sent the stall by polling the STALLSENT to be set. An endpoint interrupt is pending while STALLSENT is set. The microcontroller must clear STALLSENT to clear the interrupt.

When a setup transaction is received after a stall handshake, STALLSENT must be cleared in order to prevent interrupts due to STALLSENT being set.

Figure 41-12. Stall Handshake (Data IN Transfer)

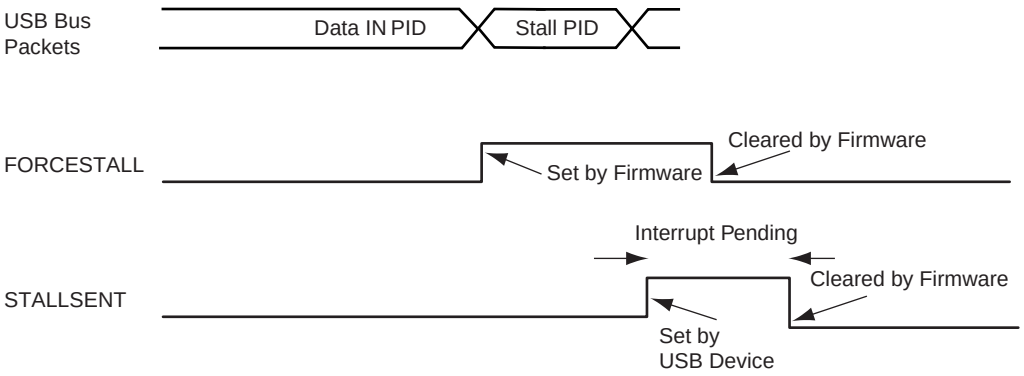
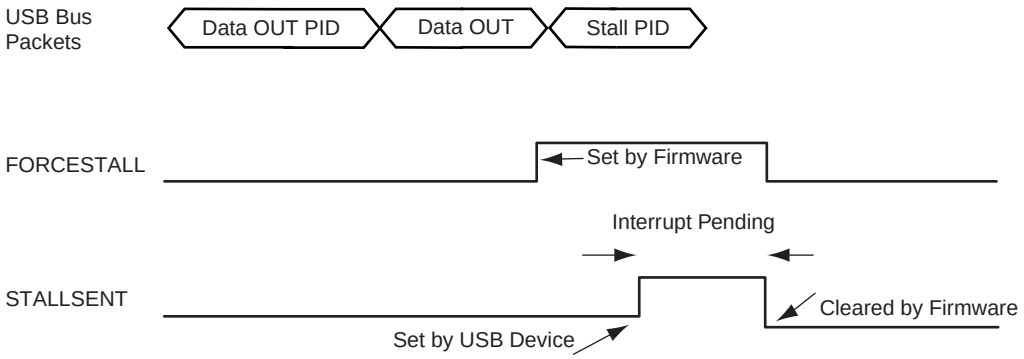


Figure 41-13. Stall Handshake (Data OUT Transfer)



41.6.2.5 Transmit Data Cancellation

Some endpoints have dual-banks whereas some endpoints have only one bank. The procedure to cancel transmission data held in these banks is described below.

To see the organization of dual-bank availability refer to Table 41-1 ”USB Endpoint Description”.

Endpoints Without Dual-Banks

The cancellation procedure depends on the TXPKTRDY flag value in the UDP_CSR:

- TXPKTRDY is not set:
  - Reset the endpoint to clear the FIFO (pointers). (See Section 41.7.9 ”UDP Reset Endpoint Register”.)
- TXPKTRDY has already been set:
  - Clear TXPKTRDY so that no packet is ready to be sent
  - Reset the endpoint to clear the FIFO (pointers). (See Section 41.7.9 ”UDP Reset Endpoint Register”.)

Endpoints With Dual-Banks

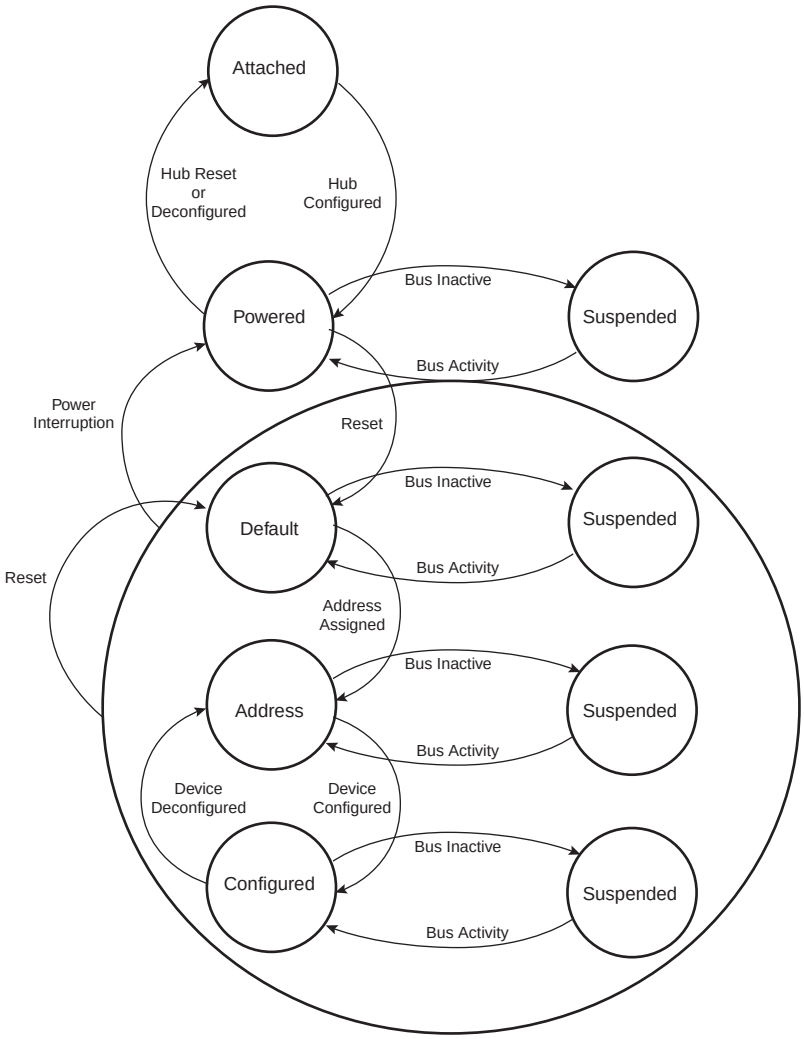
The cancellation procedure depends on the TXPKTRDY flag value in the UDP_CSR:

- TXPKTRDY is not set:
  - Reset the endpoint to clear the FIFO (pointers). (See Section 41.7.9 ”UDP Reset Endpoint Register”.)
- TXPKTRDY has already been set:
  - Clear TXPKTRDY and read it back until actually read at 0.
  - Set TXPKTRDY and read it back until actually read at 1.
  - Clear TXPKTRDY so that no packet is ready to be sent.
  - Reset the endpoint to clear the FIFO (pointers). (See Section 41.7.9 ”UDP Reset Endpoint Register”.)

41.6.3 Controlling Device States

A USB device has several possible states. Refer to Chapter 9 of the *Universal Serial Bus Specification, Rev 2.0*.

Figure 41-14. USB Device State Diagram



Movement from one state to another depends on the USB bus state or on standard requests sent through control transactions via the default endpoint (endpoint 0).

After a period of bus inactivity, the USB device enters Suspend Mode. Accepting Suspend/Resume requests from the USB host is mandatory. Constraints in Suspend Mode are very strict for bus-powered applications; devices must not consume more than 2.5 mA on the USB bus.

While in Suspend Mode, the host may wake up a device by sending a resume signal (bus activity) or a USB device may send a wakeup request to the host, e.g., waking up a PC by moving a USB mouse.

The wakeup feature is not mandatory for all devices and must be negotiated with the host.

41.6.3.1 Not Powered State

Self powered devices can detect 5V VBUS using a PIO as described in the typical connection section. When the device is not connected to a host, device power consumption can be reduced by disabling MCK for the UDP, disabling UDPCK and disabling the transceiver. DDP and DDM lines are pulled down by 330 KΩ resistors.

41.6.3.2 Entering Attached State

To enable integrated pull-up, the PUON bit in the UDP_TXVC register must be set.

Warning: To write to the UDP_TXVC register, MCK clock must be enabled on the UDP. This is done in the Power Management Controller.

After pull-up connection, the device enters the powered state. In this state, the UDPCK and MCK must be enabled in the Power Management Controller. The transceiver can remain disabled.

41.6.3.3 From Powered State to Default State

After its connection to a USB host, the USB device waits for an end-of-bus reset. The unmaskable flag ENDBUSRES is set in the UDP_ISR and an interrupt is triggered.

Once the ENDBUSRES interrupt has been triggered, the device enters Default State. In this state, the UDP software must:

- Enable the default endpoint, setting the EPEDS flag in the UDP_CSR0 and, optionally, enabling the interrupt for endpoint 0 by writing 1 to the UDP_IER. The enumeration then begins by a control transfer.
- Configure the interrupt mask register which has been reset by the USB reset detection
- Enable the transceiver clearing the TXVDIS flag in the UDP_TXVC register.

In this state UDPCK and MCK must be enabled.

Warning: Each time an ENDBUSRES interrupt is triggered, the Interrupt Mask Register and UDP_CSRs have been reset.

41.6.3.4 From Default State to Address State

After a set address standard device request, the USB host peripheral enters the address state.

Warning: Before the device enters in address state, it must achieve the Status IN transaction of the control transfer, i.e., the UDP device sets its new address once the TXCOMP flag in the UDP_CSR0 has been received and cleared.

To move to address state, the driver software sets the FADDEN flag in the UDP_GLB_STAT register, sets its new address, and sets the FEN bit in the UDP_FADDR register.

41.6.3.5 From Address State to Configured State

Once a valid Set Configuration standard request has been received and acknowledged, the device enables endpoints corresponding to the current configuration. This is done by setting the EPEDS and EPTYPE fields in the UDP_CSRx and, optionally, enabling corresponding interrupts in the UDP_IER.

41.6.3.6 Entering in Suspend State

When a Suspend (no bus activity on the USB bus) is detected, the RXSUSP signal in the UDP_ISR is set. This triggers an interrupt if the corresponding bit is set in the UDP_IMR. This flag is cleared by writing to the UDP_ICR. Then the device enters Suspend Mode.

In this state bus powered devices must drain no more than 2.5 mA from the 5V VBUS. As an example, the microcontroller switches to slow clock, disables the PLL and main oscillator, and goes into Idle Mode. It may also switch off other devices on the board.

The USB device peripheral clocks can be switched off. Resume event is asynchronously detected. MCK and UDPCK can be switched off in the Power Management controller and the USB transceiver can be disabled by setting the TXVDIS bit in the UDP_TXVC register.

Warning: Read, write operations to the UDP registers are allowed only if MCK is enabled for the UDP peripheral. Switching off MCK for the UDP peripheral must be one of the last operations after writing to the UDP_TXVC register and acknowledging the RXSUSP.

41.6.3.7 Receiving a Host Resume

In suspend mode, a resume event on the USB bus line is detected asynchronously, transceiver and clocks are disabled (however the pull-up shall not be removed).

Once the resume is detected on the bus, the WAKEUP signal in the UDP_ISR is set. It may generate an interrupt if the corresponding bit in the UDP_IMR is set. This interrupt may be used to wake up the core, enable PLL and main oscillators and configure clocks.

Warning: Read, write operations to the UDP registers are allowed only if MCK is enabled for the UDP peripheral. MCK for the UDP must be enabled before clearing the WAKEUP bit in the UDP_ICR and clearing TXVDIS in the UDP_TXVC register.

41.6.3.8 Sending a Device Remote Wakeup Request

In Suspend state it is possible to wake up the host sending an external resume.

- The device must wait at least 5 ms after being entered in suspend before sending an external resume.
- The device has 10 ms from the moment it starts to drain current and it forces a K state to resume the host.
- The device must force a K state from 1 to 15 ms to resume the host

Before sending a K state to the host, MCK, UDPCK and the transceiver must be enabled. Then to enable the remote wakeup feature, the RMWUPE bit in the UDP_GLB_STAT register must be enabled. To force the K state on the line, a transition of the ESR bit from 0 to 1 has to be done in the UDP_GLB_STAT register by first writing a 0 in the ESR bit and then writing a 1.

The K state is automatically generated and released according to the USB 2.0 specification.

41.7 USB Device Port (UDP) User Interface

WARNING: The UDP peripheral clock in the Power Management Controller (PMC) must be enabled before any read/write operations to the UDP registers, including the UDP_TXVC register.

Table 41-6. Register Mapping

Offset	Register	Name	Access	Reset
0x000	Frame Number Register	UDP_FRM_NUM	Read-only	0x0000_0000
0x004	Global State Register	UDP_GLB_STAT	Read/Write	0x0000_0010
0x008	Function Address Register	UDP_FADDR	Read/Write	0x0000_0100
0x00C	Reserved	–	–	–
0x010	Interrupt Enable Register	UDP_IER	Write-only	
0x014	Interrupt Disable Register	UDP_IDR	Write-only	
0x018	Interrupt Mask Register	UDP_IMR	Read-only	0x0000_1200
0x01C	Interrupt Status Register	UDP_ISR	Read-only	–(1)
0x020	Interrupt Clear Register	UDP_ICR	Write-only	
0x024	Reserved	–	–	–
0x028	Reset Endpoint Register	UDP_RST_EP	Read/Write	0x0000_0000
0x02C	Reserved	–	–	–
0x030	Endpoint Control and Status Register 0	UDP_CSR0	Read/Write	0x0000_0000
...	...	...	...	...
0x030 + 0x4 * 7	Endpoint Control and Status Register 7	UDP_CSR7	Read/Write	0x0000_0000
0x050	Endpoint FIFO Data Register 0	UDP_FDR0	Read/Write	–(1)
...	...	...	...	...
0x050 + 0x4 * 7	Endpoint FIFO Data Register 7	UDP_FDR7	Read/Write	–(1)
0x070	Reserved	–	–	–
0x074	Transceiver Control Register	UDP_TXVC(2)	Read/Write	0x0000_0100
0x078–0xFC	Reserved	–	–	–

Notes: 1. Reset values are not defined for UDP_ISR or UDP_FDRx. UDP_FDRs reflect Dual Port RAM memory locations which are not affected by any reset signals.

2. See Warning above the "Register Mapping" on this page.

41.7.1 UDP Frame Number Register

Name: UDP_FRM_NUM

Address: 0x40084000

Access: Read-only

31	30	29	28	27	26	25	24
---	---	---	---	---	---	---	---

23	22	21	20	19	18	17	16
–	–	–	–	–	–	FRM_OK	FRM_ERR

15	14	13	12	11	10	9	8
–	–	–	–	–	FRM_NUM		

7	6	5	4	3	2	1	0
FRM_NUM							

- **FRM_NUM[10:0]: Frame Number as Defined in the Packet Field Formats**

This 11-bit value is incremented by the host on a per frame basis. This value is updated at each start of frame.

Value updated at the SOF_EOP (Start of Frame End of Packet).

- **FRM_ERR: Frame Error**

This bit is set at SOF_EOP when the SOF packet is received containing an error.

This bit is reset upon receipt of SOF_PID.

- **FRM_OK: Frame OK**

This bit is set at SOF_EOP when the SOF packet is received without any error.

This bit is reset upon receipt of SOF_PID (Packet Identification).

In the Interrupt Status Register, the SOF interrupt is updated upon receiving SOF_PID. This bit is set without waiting for EOP.

Note: In the 8-bit Register Interface, FRM_OK is bit 4 of FRM_NUM_H and FRM_ERR is bit 3 of FRM_NUM_L.

41.7.2 UDP Global State Register

Name: UDP_GLB_STAT

Address: 0x40084004

Access: Read/Write

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

This register is used to get and set the device state as specified in Chapter 9 of the *USB Serial Bus Specification, Rev.2.0*.

- **FADDEN: Function Address Enable**

Read:

0: Device is not in address state

1: Device is in address state

Write:

0: No effect, only a reset can bring back a device to the default state.

1: Sets device in address state. This occurs after a successful Set Address request. Beforehand, the UDP_FADDR register must have been initialized with Set Address parameters. Set Address must complete the Status Stage before setting FADDEN. Refer to chapter 9 of the *Universal Serial Bus Specification, Rev. 2.0* for more details.

- **CONFG: Configured**

Read:

0: Device is not in configured state

1: Device is in configured state

Write:

0: Sets device in a non configured state

1: Sets device in configured state

The device is set in configured state when it is in address state and receives a successful Set Configuration request. Refer to Chapter 9 of the *Universal Serial Bus Specification, Rev. 2.0* for more details.

- **ESR: Enable Send Resume**

0: Mandatory value prior to starting any Remote Wakeup procedure

1: Starts the Remote Wakeup procedure if this bit value was 0 and if RMWUPE is enabled

- **RMWUPE: Remote Wakeup Enable**

0: The Remote Wakeup feature of the device is disabled.

1: The Remote Wakeup feature of the device is enabled.

41.7.3 UDP Function Address Register

Name: UDP_FADDR

Address: 0x40084008

Access: Read/Write

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

- **FADD[6:0]: Function Address Value**

The Function Address Value must be programmed by firmware once the device receives a set address request from the host, and has achieved the status stage of the no-data control sequence. Refer to the *Universal Serial Bus Specification, Rev. 2.0* for more information. After power up or reset, the function address value is set to 0.

- **FEN: Function Enable**

Read:

0: Function endpoint disabled

1: Function endpoint enabled

Write:

0: Disables function endpoint

1: Default value

The Function Enable bit (FEN) allows the microcontroller to enable or disable the function endpoints. The microcontroller sets this bit after receipt of a reset from the host. Once this bit is set, the USB device is able to accept and transfer data packets from and to the host.

41.7.4 UDP Interrupt Enable Register

Name: UDP_IER

Address: 0x40084010

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	WAKEUP	–	SOFINT	EXTRSM	RXRSM	RXSUSP
7	6	5	4	3	2	1	0
EP7INT	EP6INT	EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Enable Endpoint 0 Interrupt**
- **EP1INT: Enable Endpoint 1 Interrupt**
- **EP2INT: Enable Endpoint 2Interrupt**
- **EP3INT: Enable Endpoint 3 Interrupt**
- **EP4INT: Enable Endpoint 4 Interrupt**
- **EP5INT: Enable Endpoint 5 Interrupt**
- **EP6INT: Enable Endpoint 6 Interrupt**
- **EP7INT: Enable Endpoint 7 Interrupt**

0: No effect

1: Enables corresponding Endpoint Interrupt

- **RXSUSP: Enable UDP Suspend Interrupt**

0: No effect

1: Enables UDP Suspend Interrupt

- **RXRSM: Enable UDP Resume Interrupt**

0: No effect

1: Enables UDP Resume Interrupt

- **SOFINT: Enable Start Of Frame Interrupt**

0: No effect

1: Enables Start Of Frame Interrupt

- **WAKEUP: Enable UDP Bus Wakeup Interrupt**

0: No effect

1: Enables USB bus Interrupt

41.7.5 UDP Interrupt Disable Register

Name: UDP_IDR

Address: 0x40084014

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	WAKEUP	–	SOFINT	EXTRSM	RXRSM	RXSUSP

7	6	5	4	3	2	1	0
EP7INT	EP6INT	EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Disable Endpoint 0 Interrupt**
- **EP1INT: Disable Endpoint 1 Interrupt**
- **EP2INT: Disable Endpoint 2 Interrupt**
- **EP3INT: Disable Endpoint 3 Interrupt**
- **EP4INT: Disable Endpoint 4 Interrupt**
- **EP5INT: Disable Endpoint 5 Interrupt**
- **EP6INT: Disable Endpoint 6 Interrupt**
- **EP7INT: Disable Endpoint 7 Interrupt**

0: No effect

1: Disables corresponding Endpoint Interrupt

- **RXSUSP: Disable UDP Suspend Interrupt**

0: No effect

1: Disables UDP Suspend Interrupt

- **RXRSM: Disable UDP Resume Interrupt**

0: No effect

1: Disables UDP Resume Interrupt

- **SOFINT: Disable Start Of Frame Interrupt**

0: No effect

1: Disables Start Of Frame Interrupt

- **WAKEUP: Disable USB Bus Interrupt**

0: No effect

1: Disables USB Bus Wakeup Interrupt

41.7.6 UDP Interrupt Mask Register

Name: UDP_IMR

Address: 0x40084018

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	WAKEUP	BIT12	SOFINT	EXTRSM	RXRSM	RXSUSP
7	6	5	4	3	2	1	0
EP7INT	EP6INT	EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Mask Endpoint 0 Interrupt**
- **EP1INT: Mask Endpoint 1 Interrupt**
- **EP2INT: Mask Endpoint 2 Interrupt**
- **EP3INT: Mask Endpoint 3 Interrupt**
- **EP4INT: Mask Endpoint 4 Interrupt**
- **EP5INT: Mask Endpoint 5 Interrupt**
- **EP6INT: Mask Endpoint 6 Interrupt**
- **EP7INT: Mask Endpoint 7 Interrupt**

0: Corresponding Endpoint Interrupt is disabled

1: Corresponding Endpoint Interrupt is enabled

- **RXSUSP: Mask UDP Suspend Interrupt**

0: UDP Suspend Interrupt is disabled

1: UDP Suspend Interrupt is enabled

- **RXRSM: Mask UDP Resume Interrupt.**

0: UDP Resume Interrupt is disabled

1: UDP Resume Interrupt is enabled

- **SOFINT: Mask Start Of Frame Interrupt**

0: Start of Frame Interrupt is disabled

1: Start of Frame Interrupt is enabled

- **BIT12: UDP_IMR Bit 12**

Bit 12 of UDP_IMR cannot be masked and is always read at 1.

- **WAKEUP: USB Bus Wakeup Interrupt**

0: USB Bus Wakeup Interrupt is disabled

1: USB Bus Wakeup Interrupt is enabled

Note: When the USB block is in suspend mode, the application may power down the USB logic. In this case, any USB HOST resume request that is made must be taken into account and, thus, the reset value of the RXRSM bit of the register UDP_IMR is enabled.

41.7.7 UDP Interrupt Status Register

Name: UDP_ISR

Address: 0x4008401C

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	WAKEUP	ENDBUSRES	SOFINT	EXTRSM	RXRSM	RXSUSP
7	6	5	4	3	2	1	0
EP7INT	EP6INT	EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Endpoint 0 Interrupt Status**
- **EP1INT: Endpoint 1 Interrupt Status**
- **EP2INT: Endpoint 2 Interrupt Status**
- **EP3INT: Endpoint 3 Interrupt Status**
- **EP4INT: Endpoint 4 Interrupt Status**
- **EP5INT: Endpoint 5 Interrupt Status**
- **EP6INT: Endpoint 6 Interrupt Status**
- **EP7INT: Endpoint 7Interrupt Status**

0: No Endpoint0 Interrupt pending

1: Endpoint0 Interrupt has been raised

Several signals can generate this interrupt. The reason can be found by reading UDP_CSR0:

RXSETUP set to 1

RX_DATA_BK0 set to 1

RX_DATA_BK1 set to 1

TXCOMP set to 1

STALLSENT set to 1

EP0INT is a sticky bit. Interrupt remains valid until EP0INT is cleared by writing in the corresponding UDP_CSR0 bit.

- **RXSUSP: UDP Suspend Interrupt Status**

0: No UDP Suspend Interrupt pending

1: UDP Suspend Interrupt has been raised

The USB device sets this bit when it detects no activity for 3 ms. The USB device enters Suspend mode.

- **RXRSM: UDP Resume Interrupt Status**

0: No UDP Resume Interrupt pending

1: UDP Resume Interrupt has been raised

The USB device sets this bit when a UDP resume signal is detected at its port.

After reset, the state of this bit is undefined, the application must clear this bit by setting the RXRSM flag in the UDP_ICR.

- **SOFINT: Start of Frame Interrupt Status**

0: No Start of Frame Interrupt pending

1: Start of Frame Interrupt has been raised

This interrupt is raised each time a SOF token has been detected. It can be used as a synchronization signal by using isochronous endpoints.

- **ENDBUSRES: End of BUS Reset Interrupt Status**

0: No End of Bus Reset Interrupt pending

1: End of Bus Reset Interrupt has been raised

This interrupt is raised at the end of a UDP reset sequence. The USB device must prepare to receive requests on the endpoint 0. The host starts the enumeration, then performs the configuration.

- **WAKEUP: UDP Resume Interrupt Status**

0: No Wakeup Interrupt pending

1: A Wakeup Interrupt (USB Host Sent a RESUME or RESET) occurred since the last clear.

After reset the state of this bit is undefined; the application must clear this bit by setting the WAKEUP flag in the UDP_ICR.

41.7.8 UDP Interrupt Clear Register

Name: UDP_ICR

Address: 0x40084020

Access: Write-only

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

- **RXSUSP: Clear UDP Suspend Interrupt**

0: No effect

1: Clears UDP Suspend Interrupt

- **RXRSM: Clear UDP Resume Interrupt**

0: No effect

1: Clears UDP Resume Interrupt

- **SOFINT: Clear Start Of Frame Interrupt**

0: No effect

1: Clears Start Of Frame Interrupt

- **ENDBUSRES: Clear End of Bus Reset Interrupt**

0: No effect

1: Clears End of Bus Reset Interrupt

- **WAKEUP: Clear Wakeup Interrupt**

0: No effect

1: Clears Wakeup Interrupt

41.7.9 UDP Reset Endpoint Register

Name: UDP_RST_EP

Address: 0x40084028

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
EP7	EP6	EP5	EP4	EP3	EP2	EP1	EP0

- **EP0: Reset Endpoint 0**
- **EP1: Reset Endpoint 1**
- **EP2: Reset Endpoint 2**
- **EP3: Reset Endpoint 3**
- **EP4: Reset Endpoint 4**
- **EP5: Reset Endpoint 5**
- **EP6: Reset Endpoint 6**
- **EP7: Reset Endpoint 7**

This flag is used to reset the FIFO associated with the endpoint and the bit RXBYTECOUNT in the UDP_CSRx. It also resets the data toggle to DATA0. It is useful after removing a HALT condition on a BULK endpoint. Refer to Chapter 5.8.5 in the *USB Serial Bus Specification, Rev.2.0*.

Warning: This flag must be cleared at the end of the reset. It does not clear UDP_CSRx flags.

0: No reset

1: Forces the corresponding endpoint FIF0 pointers to 0, therefore RXBYTECNT field is read at 0 in UDP_CSRx

Resetting the endpoint is a two-step operation:

1. Set the corresponding EPx field.
2. Clear the corresponding EPx field.

41.7.10 UDP Endpoint Control and Status Register (CONTROL_BULK)

Name: UDP_CSRx [x = 0..7] (CONTROL_BULK)

Address: 0x40084030

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	RXBYTECNT		

23	22	21	20	19	18	17	16
RXBYTECNT							

15	14	13	12	11	10	9	8
EPEDS	–	–	–	DTGLE	EPTYPE		

7	6	5	4	3	2	1	0
DIR	RX_DATA_BK1	FORCESTALL	TXPKTRDY	STALLSENT	RXSETUP	RX_DATA_BK0	TXCOMP

WARNING: Due to synchronization between MCK and UDPCK, the software application must wait for the end of the write operation before executing another write by polling the bits which must be set/cleared.

As an example, to perform a control operation on the endpoint without modifying the status flags while accessing the control bits and fields of this register, the status flag bits must first be defined with the “No effect” value ‘1’. Once the overall UDP_CSR value is defined, the register can be written and then the synchronization wait procedure must be executed.

- **TXCOMP: Generates an IN Packet with Data Previously Written in the DPR**

This flag generates an interrupt while it is set to one.

Write (cleared by the firmware):

0: Clear the flag, clear the interrupt

1: No effect

Read (Set by the USB peripheral):

0: Data IN transaction has not been acknowledged by the Host

1: Data IN transaction is achieved, acknowledged by the Host

After having issued a Data IN transaction setting TXPKTRDY, the device firmware waits for TXCOMP to be sure that the host has acknowledged the transaction.

- **RX_DATA_BK0: Receive Data Bank 0**

This flag generates an interrupt while it is set to one.

Write (cleared by the firmware):

0: Notify USB peripheral device that data have been read in the FIFO’s Bank 0.

1: To leave the read value unchanged.

Read (Set by the USB peripheral):

0: No data packet has been received in the FIFO’s Bank 0.

1: A data packet has been received, it has been stored in the FIFO’s Bank 0.

When the device firmware has polled this bit or has been interrupted by this signal, it must transfer data from the FIFO to the microcontroller memory. The number of bytes received is available in RXBYTCENT field. Bank 0 FIFO values are read

through the UDP_FDRx. Once a transfer is done, the device firmware must release Bank 0 to the USB peripheral device by clearing RX_DATA_BK0.

After setting or clearing this bit, a wait time of 3 UDPCK clock cycles and 3 peripheral clock cycles is required before accessing DPR.

- **RXSETUP: Received Setup**

This flag generates an interrupt while it is set to one.

Read:

0: No setup packet available.

1: A setup data packet has been sent by the host and is available in the FIFO.

Write:

0: Device firmware notifies the USB peripheral device that it has read the setup data in the FIFO.

1: No effect.

This flag is used to notify the USB device firmware that a valid Setup data packet has been sent by the host and successfully received by the USB device. The USB device firmware may transfer Setup data from the FIFO by reading the UDP_FDRx to the microcontroller memory. Once a transfer has been done, RXSETUP must be cleared by the device firmware.

Ensuing Data OUT transaction is not accepted while RXSETUP is set.

- **STALLSENT: Stall Sent**

This flag generates an interrupt while it is set to one.

This ends a STALL handshake.

Read:

0: Host has not acknowledged a stall

1: Host has acknowledged the stall

Write:

0: Resets the STALLSENT flag, clears the interrupt

1: No effect

This is mandatory for the device firmware to clear this flag. Otherwise the interrupt remains.

Refer to chapters 8.4.5 and 9.4.5 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the STALL handshake.

- **TXPKTRDY: Transmit Packet Ready**

This flag is cleared by the USB device.

This flag is set by the USB device firmware.

Read:

0: There is no data to send.

1: The data is waiting to be sent upon reception of token IN.

Write:

0: Can be used in the procedure to cancel transmission data. (See Section 41.6.2.5 “Transmit Data Cancellation” on page 1153)

1: A new data payload has been written in the FIFO by the firmware and is ready to be sent.

This flag is used to generate a Data IN transaction (device to host). Device firmware checks that it can write a data payload in the FIFO, checking that TXPKTRDY is cleared. Transfer to the FIFO is done by writing in the UDP_FDRx. Once the data payload has been transferred to the FIFO, the firmware notifies the USB device setting TXPKTRDY to one. USB bus transactions can start. TXCOMP is set once the data payload has been received by the host.

After setting or clearing this bit, a wait time of 3 UDPCK clock cycles and 3 peripheral clock cycles is required before accessing DPR.

- **FORCESTALL: Force Stall (used by Control, Bulk and Isochronous Endpoints)**

Read:

0: Normal state

1: Stall state

Write:

0: Return to normal state

1: Send STALL to the host

Refer to chapters 8.4.5 and 9.4.5 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the STALL handshake.

Control endpoints: During the data stage and status stage, this bit indicates that the microcontroller cannot complete the request.

Bulk and interrupt endpoints: This bit notifies the host that the endpoint is halted.

The host acknowledges the STALL, device firmware is notified by the STALLSENT flag.

- **RX_DATA_BK1: Receive Data Bank 1 (only used by endpoints with ping-pong attributes)**

This flag generates an interrupt while it is set to one.

Write (cleared by the firmware):

0: Notifies USB device that data have been read in the FIFO’s Bank 1.

1: To leave the read value unchanged.

Read (Set by the USB peripheral):

0: No data packet has been received in the FIFO’s Bank 1.

1: A data packet has been received, it has been stored in FIFO’s Bank 1.

When the device firmware has polled this bit or has been interrupted by this signal, it must transfer data from the FIFO to microcontroller memory. The number of bytes received is available in RXBYTECNT field. Bank 1 FIFO values are read through UDP_FDRx. Once a transfer is done, the device firmware must release Bank 1 to the USB device by clearing RX_DATA_BK1.

After setting or clearing this bit, a wait time of 3 UDPCK clock cycles and 3 peripheral clock cycles is required before accessing DPR.

- **DIR: Transfer Direction (only available for control endpoints) (Read/Write)**

0: Allows Data OUT transactions in the control data stage.

1: Enables Data IN transactions in the control data stage.

Refer to Chapter 8.5.3 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the control data stage.

This bit must be set before UDP_CSRx/RXSETUP is cleared at the end of the setup stage. According to the request sent in the setup data packet, the data stage is either a device to host (DIR = 1) or host to device (DIR = 0) data transfer. It is not necessary to check this bit to reverse direction for the status stage.

- **EPTYPE[2:0]: Endpoint Type (Read/Write)**

Value	Name	Description
0	CTRL	Control
1	ISO_OUT	Isochronous OUT
5	ISO_IN	Isochronous IN
2	BULK_OUT	Bulk OUT
6	BULK_IN	Bulk IN
3	INT_OUT	Interrupt OUT
7	INT_IN	Interrupt IN

- **DTGLE: Data Toggle (Read-only)**

0: Identifies DATA0 packet

1: Identifies DATA1 packet

Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on DATA0, DATA1 packet definitions.

- **EPEDS: Endpoint Enable Disable**

Read:

0: Endpoint disabled

1: Endpoint enabled

Write:

0: Disables endpoint

1: Enables endpoint

Control endpoints are always enabled. Reading or writing this field has no effect on control endpoints.

Note: After reset, all endpoints are configured as control endpoints (zero).

- **RXBYTECNT[10:0]: Number of Bytes Available in the FIFO (Read-only)**

When the host sends a data packet to the device, the USB device stores the data in the FIFO and notifies the microcontroller. The microcontroller can load the data from the FIFO by reading RXBYTECENT bytes in the UDP_FDRx.

41.7.11 UDP Endpoint Control and Status Register (ISOCHRONOUS)

Name: UDP_CSRx [x = 0..7] (ISOCHRONOUS)

Address: 0x40084030

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	RXBYTECNT		

23	22	21	20	19	18	17	16
RXBYTECNT							

15	14	13	12	11	10	9	8
EPEDS	–	–	–	DTGLE	EPTYPE		

7	6	5	4	3	2	1	0
DIR	RX_DATA_BK1	FORCESTALL	TXPKTRDY	ISOERROR	RXSETUP	RX_DATA_BK0	TXCOMP

- **TXCOMP: Generates an IN Packet with Data Previously Written in the DPR**

This flag generates an interrupt while it is set to one.

Write (cleared by the firmware):

0: Clear the flag, clear the interrupt.

1: No effect.

Read (Set by the USB peripheral):

0: Data IN transaction has not been acknowledged by the Host.

1: Data IN transaction is achieved, acknowledged by the Host.

After having issued a Data IN transaction setting TXPKTRDY, the device firmware waits for TXCOMP to be sure that the host has acknowledged the transaction.

- **RX_DATA_BK0: Receive Data Bank 0**

This flag generates an interrupt while it is set to one.

Write (cleared by the firmware):

0: Notify USB peripheral device that data have been read in the FIFO's Bank 0.

1: To leave the read value unchanged.

Read (Set by the USB peripheral):

0: No data packet has been received in the FIFO's Bank 0.

1: A data packet has been received, it has been stored in the FIFO's Bank 0.

When the device firmware has polled this bit or has been interrupted by this signal, it must transfer data from the FIFO to the microcontroller memory. The number of bytes received is available in RXBYTCENT field. Bank 0 FIFO values are read through the UDP_FDRx. Once a transfer is done, the device firmware must release Bank 0 to the USB peripheral device by clearing RX_DATA_BK0.

After setting or clearing this bit, a wait time of 3 UDPCK clock cycles and 3 peripheral clock cycles is required before accessing DPR.

- **RXSETUP: Received Setup**

This flag generates an interrupt while it is set to one.

Read:

0: No setup packet available.

1: A setup data packet has been sent by the host and is available in the FIFO.

Write:

0: Device firmware notifies the USB peripheral device that it has read the setup data in the FIFO.

1: No effect.

This flag is used to notify the USB device firmware that a valid Setup data packet has been sent by the host and successfully received by the USB device. The USB device firmware may transfer Setup data from the FIFO by reading the UDP_FDRx to the microcontroller memory. Once a transfer has been done, RXSETUP must be cleared by the device firmware.

Ensuing Data OUT transaction is not accepted while RXSETUP is set.

- **ISOERROR: A CRC error has been detected in an isochronous transfer**

This flag generates an interrupt while it is set to one.

Read:

0: No error in the previous isochronous transfer.

1: CRC error has been detected, data available in the FIFO are corrupted.

Write:

0: Resets the ISOERROR flag, clears the interrupt.

1: No effect.

- **TXPKTRDY: Transmit Packet Ready**

This flag is cleared by the USB device.

This flag is set by the USB device firmware.

Read:

0: There is no data to send.

1: The data is waiting to be sent upon reception of token IN.

Write:

0: Can be used in the procedure to cancel transmission data. (See Section 41.6.2.5 “Transmit Data Cancellation” on page 1153)

1: A new data payload has been written in the FIFO by the firmware and is ready to be sent.

This flag is used to generate a Data IN transaction (device to host). Device firmware checks that it can write a data payload in the FIFO, checking that TXPKTRDY is cleared. Transfer to the FIFO is done by writing in the UDP_FDRx. Once the data payload has been transferred to the FIFO, the firmware notifies the USB device setting TXPKTRDY to one. USB bus transactions can start. TXCOMP is set once the data payload has been received by the host.

After setting or clearing this bit, a wait time of 3 UDPCK clock cycles and 3 peripheral clock cycles is required before accessing DPR.

- **FORCESTALL: Force Stall (used by Control, Bulk and Isochronous Endpoints)**

Read:

0: Normal state.

1: Stall state.

Write:

0: Return to normal state.

1: Send STALL to the host.

Refer to chapters 8.4.5 and 9.4.5 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the STALL handshake.

Control endpoints: During the data stage and status stage, this bit indicates that the microcontroller cannot complete the request.

Bulk and interrupt endpoints: This bit notifies the host that the endpoint is halted.

The host acknowledges the STALL, device firmware is notified by the STALLSENT flag.

- **RX_DATA_BK1: Receive Data Bank 1 (only used by endpoints with ping-pong attributes)**

This flag generates an interrupt while it is set to one.

Write (cleared by the firmware):

0: Notifies USB device that data have been read in the FIFO's Bank 1.

1: To leave the read value unchanged.

Read (set by the USB peripheral):

0: No data packet has been received in the FIFO's Bank 1.

1: A data packet has been received, it has been stored in FIFO's Bank 1.

When the device firmware has polled this bit or has been interrupted by this signal, it must transfer data from the FIFO to microcontroller memory. The number of bytes received is available in RXBYTECNT field. Bank 1 FIFO values are read through UDP_FDRx. Once a transfer is done, the device firmware must release Bank 1 to the USB device by clearing RX_DATA_BK1.

After setting or clearing this bit, a wait time of 3 UDPCK clock cycles and 3 peripheral clock cycles is required before accessing DPR.

- **DIR: Transfer Direction (only available for control endpoints) (Read/Write)**

0: Allows Data OUT transactions in the control data stage.

1: Enables Data IN transactions in the control data stage.

Refer to Chapter 8.5.3 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the control data stage.

This bit must be set before UDP_CSRx/RXSETUP is cleared at the end of the setup stage. According to the request sent in the setup data packet, the data stage is either a device to host (DIR = 1) or host to device (DIR = 0) data transfer. It is not necessary to check this bit to reverse direction for the status stage.

- **EPTYPE[2:0]: Endpoint Type (Read/Write)**

Value	Name	Description
0	CTRL	Control
1	ISO_OUT	Isochronous OUT
5	ISO_IN	Isochronous IN
2	BULK_OUT	Bulk OUT

Value	Name	Description
6	BULK_IN	Bulk IN
3	INT_OUT	Interrupt OUT
7	INT_IN	Interrupt IN

- **DTGLE: Data Toggle (Read-only)**

0: Identifies DATA0 packet

1: Identifies DATA1 packet

Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on DATA0, DATA1 packet definitions.

- **EPEDS: Endpoint Enable Disable**

Read:

0: Endpoint disabled

1: Endpoint enabled

Write:

0: Disables endpoint

1: Enables endpoint

Control endpoints are always enabled. Reading or writing this field has no effect on control endpoints.

Note: After reset, all endpoints are configured as control endpoints (zero).

- **RXBYTECNT[10:0]: Number of Bytes Available in the FIFO (Read-only)**

When the host sends a data packet to the device, the USB device stores the data in the FIFO and notifies the microcontroller. The microcontroller can load the data from the FIFO by reading RXBYTECENT bytes in the UDP_FDRx.

41.7.12 UDP FIFO Data Register

Name: UDP_FDRx [x = 0..7]

Address: 0x40084050

Access: Read/Write

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

- **FIFO_DATA[7:0]: FIFO Data Value**

The microcontroller can push or pop values in the FIFO through this register.

RXBYTECNT in the corresponding UDP_CSRx is the number of bytes to be read from the FIFO (sent by the host).

The maximum number of bytes to write is fixed by the Max Packet Size in the Standard Endpoint Descriptor. It can not be more than the physical memory size associated to the endpoint. Refer to the *Universal Serial Bus Specification, Rev. 2.0* for more information.

41.7.13 UDP Transceiver Control Register

Name: UDP_TXVC

Address: 0x40084074

Access: Read/Write

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

WARNING: The UDP peripheral clock in the Power Management Controller (PMC) must be enabled before any read/write operations to the UDP registers including the UDP_TXVC register.

- **TXVDIS: Transceiver Disable**

When UDP is disabled, power consumption can be reduced significantly by disabling the embedded transceiver. This can be done by setting TXVDIS bit.

To enable the transceiver, TXVDIS must be cleared.

- **PUON: Pull-up On**

0: The 1.5KΩ integrated pull-up on DDP is disconnected.

1: The 1.5 KΩ integrated pull-up on DDP is connected.

NOTE: If the USB pull-up is not connected on DDP, the user should not write in any UDP register other than the UDP_TXVC register. This is because if DDP and DDM are floating at 0, or pulled down, then SE0 is received by the device with the consequence of a USB Reset.

42. Ethernet MAC (GMAC)

42.1 Description

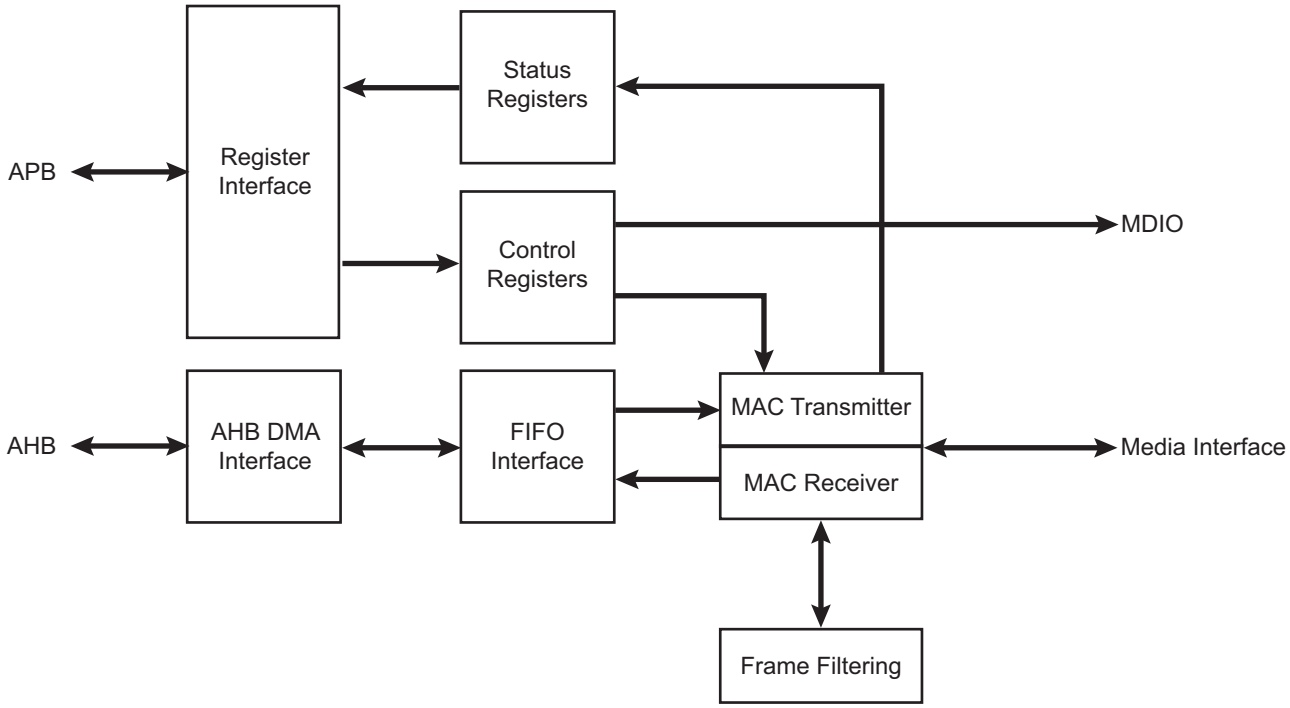
The Ethernet MAC (GMAC) module implements a 10/100 Mbps Ethernet MAC compatible with the IEEE 802.3 standard. The GMAC can operate in either half or full duplex mode at all supported speeds. The GMAC Network Configuration Register is used to select the speed, duplex mode and interface type (MII).

42.2 Embedded Characteristics

- Compatible with IEEE Standard 802.3
- 10, 100 Mbps Operation
- Full and Half Duplex Operation at all Supported Speeds of Operation
- MII Interface to the Physical Layer
- Integrated Physical Coding
- Direct Memory Access (DMA) Interface to External Memory
- Programmable Burst Length and Endianism for DMA
- Interrupt Generation to Signal Receive and Transmit Completion, Errors or Other Events
- Automatic Pad and Cyclic Redundancy Check (CRC) Generation on Transmitted Frames
- Automatic Discard of Frames Received with Errors
- Receive and Transmit IP, TCP and UDP Checksum Offload. Both IPv4 and IPv6 Packet Types Supported
- Address Checking Logic for Four Specific 48-bit Addresses, Four Type IDs, Promiscuous Mode, Hash Matching of Unicast and Multicast Destination Addresses and Wake-on-LAN
- Management Data Input/Output (MDIO) Interface for Physical Layer Management
- Support for Jumbo Frames up to 10240 Bytes
- Full Duplex Flow Control with Recognition of Incoming Pause Frames and Hardware Generation of Transmitted Pause Frames
- Half Duplex Flow Control by Forcing Collisions on Incoming Frames
- Support for 802.1Q VLAN Tagging with Recognition of Incoming VLAN and Priority Tagged Frames
- Support for 802.1Qbb Priority-based Flow Control
- Programmable Inter Packet Gap (IPG) Stretch
- Recognition of IEEE 1588 PTP Frames
- IEEE 1588 Time Stamp Unit (TSU)
- Support for 802.1AS Timing and Synchronization

42.3 Block Diagram

Figure 42-1. Block Diagram



42.4 Signal Interfaces

The GMAC includes the following signal interfaces:

- MII to an external PHY
- MDIO interface for external PHY management
- Slave APB interface for accessing GMAC registers
- Master AHB interface for memory access

Table 42-1. GMAC Connections in Different Modes

Signal Name	Function	MII
GTXCK	Transmit Clock or Reference Clock	TXCK
GTXEN	Transmit Enable	TXEN
GTX[3..0]	Transmit Data	TXD[3:0]
GTXER	Transmit Coding Error	TXER
GRXCK	Receive Clock	RXCK
GRXDV	Receive Data Valid	RXDV
GRX[3..0]	Receive Data	RXD[3:0]
GRXER	Receive Error	RXER
GCRS	Carrier Sense and Data Valid	CRS

Table 42-1. GMAC Connections in Different Modes (Continued)

Signal Name	Function	MII
GCOL	Collision Detect	COL
GMDC	Management Data Clock	MDC
GMDIO	Management Data Input/Output	MDIO

42.5 Product Dependencies

42.5.1 I/O Lines

The pins used for interfacing the GMAC may be multiplexed with PIO lines. The programmer must first program the PIO Controller to assign the pins to their peripheral function. If I/O lines of the GMAC are not used by the application, they can be used for other purposes by the PIO Controller.

Table 42-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
GMAC	GCOL	PD13	A
GMAC	GCRS	PD10	A
GMAC	GCRSDV/GRXDV	PD4	A
GMAC	GMDC	PD8	A
GMAC	GMDIO	PD9	A
GMAC	GRXCK	PD14	A
GMAC	GRXER	PD7	A
GMAC	GRX0	PD5	A
GMAC	GRX0	PD6	A
GMAC	GRX2	PD11	A
GMAC	GRX3	PD12	A
GMAC	GTXCK/GREFCK	PD0	A
GMAC	GTXEN	PD1	A
GMAC	GTXER	PD17	A
GMAC	GTX0	PD2	A
GMAC	GTX1	PD3	A
GMAC	GTX2	PD15	A
GMAC	GTX3	PD16	A

42.5.2 Power Management

The GMAC is not continuously clocked. The user must first enable the GMAC clock in the Power Management Controller before using it.

42.5.3 Interrupt Sources

The GMAC interrupt line is connected to one of the internal sources of the interrupt controller. Using the GMAC interrupt requires prior programming of the interrupt controller.

Table 42-3. Peripheral IDs

Instance	ID
GMAC	44

42.6 Functional Description

42.6.1 Media Access Controller

The Media Access Controller (MAC) transmit block takes data from FIFO, adds preamble and, if necessary, pad and frame check sequence (FCS). Both half duplex and full duplex Ethernet modes of operation are supported. When operating in half duplex mode, the MAC transmit block generates data according to the carrier sense multiple access with collision detect (CSMA/CD) protocol. The start of transmission is deferred if carrier sense (CRS) is active. If collision (COL) becomes active during transmission, a jam sequence is asserted and the transmission is retried after a random back off. The CRS and COL signals have no effect in full duplex mode.

The MAC receive block checks for valid preamble, FCS, alignment and length, and presents received frames to the MAC address checking block and FIFO. Software can configure the GMAC to receive jumbo frames up to 10240 bytes. It can optionally strip CRC from the received frame prior to transfer to FIFO.

The address checker recognizes four specific 48-bit addresses, can recognize four different type ID values, and contains a 64-bit Hash register for matching multicast and unicast addresses as required. It can recognize the broadcast address of all ones and copy all frames. The MAC can also reject all frames that are not VLAN tagged and recognize Wake on LAN events.

The MAC receive block supports offloading of IP, TCP and UDP checksum calculations (both IPv4 and IPv6 packet types supported), and can automatically discard bad checksum frames.

42.6.2 1588 Time Stamp Unit

The 1588 time stamp unit (TSU) is a timer implemented as a 62-bit timer comprising two registers (GMAC_TSL and GMAC_TN).

The 32 upper bits count seconds and are accessible in the “GMAC 1588 Timer Seconds Low Register” (GMAC_TSL).

The 30 lower bits count nanoseconds and are accessible in the “GMAC 1588 Timer Nanoseconds Register” (GMAC_TN).

The 30 lower bits roll over when they have counted to one second. The timer increments by a programmable number of nanoseconds with each MCK period and can be adjusted (incremented or decremented) through APB register accesses.

42.6.3 AHB Direct Memory Access Interface

The GMAC DMA controller performs six types of operations on the AHB bus. The order of priority of these operations is:

1. Receive buffer manager write
2. Receive buffer manager read
3. Transmit buffer manager write
4. Transmit buffer manager read
5. Receive data DMA write
6. Transmit data DMA read

42.6.3.1 Receive AHB Buffers

Received frames, optionally including FCS, are written to receive AHB buffers stored in memory. The receive buffer depth is programmable in the range of 64 bytes to 16 Kbytes through the DMA Configuration register, with the default being 128 bytes.

The start location for each receive AHB buffer is stored in memory in a list of receive buffer descriptors at an address location pointed to by the receive buffer queue pointer. The base address for the receive buffer queue pointer is configured in software using the Receive Buffer Queue Base Address register.

Each list entry consists of two words. The first is the address of the receive AHB buffer and the second the receive status. If the length of a receive frame exceeds the AHB buffer length, the status word for the used buffer is written with zeroes except for the “start of frame” bit, which is always set for the first buffer in a frame. Bit zero of the address field is written to 1 to show the buffer has been used. The receive buffer manager then reads the location of the next receive AHB buffer and fills that with the next part of the received frame data. AHB buffers are filled until the frame is complete and the final buffer descriptor status word contains the complete frame status. Refer to Table 42-4 for details of the receive buffer descriptor list.

Each receive AHB buffer start location is a word address. The start of the first AHB buffer in a frame can be offset by up to three bytes, depending on the value written to bits 14 and 15 of the Network Configuration register. If the start location of the AHB buffer is offset, the available length of the first AHB buffer is reduced by the corresponding number of bytes.

Table 42-4. 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 GMAC to write data to the receive buffer. The GMAC sets this to one once it has successfully written a frame to memory. Software has to clear this bit before the buffer can be used again.
Word 1	
31	Global all ones broadcast address detected
30	Multicast hash match
29	Unicast hash match
28	–
27	Specific Address Register match found, bit 25 and bit 26 indicate which Specific Address Register causes the match.

Table 42-4. Receive Buffer Descriptor Entry (Continued)

Bit	Function
26:25	Specific Address Register match. Encoded as follows: 00: Specific Address Register 1 match 01: Specific Address Register 2 match 10: Specific Address Register 3 match 11: Specific Address Register 4 match If more than one specific address is matched only one is indicated with priority 4 down to 1.
24	This bit has a different meaning depending on whether RX checksum offloading is enabled. **With RX checksum offloading disabled:** (bit 24 clear in Network Configuration Register) Type ID register match found, bit 22 and bit 23 indicate which type ID register causes the match. **With RX checksum offloading enabled:** (bit 24 set in Network Configuration Register) 0: The frame was not SNAP encoded and/or had a VLAN tag with the Canonical Format Indicator (CFI) bit set. 1: The frame was SNAP encoded and had either no VLAN tag or a VLAN tag with the CFI bit not set.
23:22	This bit has a different meaning depending on whether RX checksum offloading is enabled. **With RX checksum offloading disabled:** (bit 24 clear in Network Configuration) Type ID register match. Encoded as follows: 00: Type ID register 1 match 01: Type ID register 2 match 10: Type ID register 3 match 11: Type ID register 4 match If more than one Type ID is matched only one is indicated with priority 4 down to 1. **With RX checksum offloading enabled:** (bit 24 set in Network Configuration Register) 00: Neither the IP header checksum nor the TCP/UDP checksum was checked. 01: The IP header checksum was checked and was correct. Neither the TCP nor UDP checksum was checked. 10: Both the IP header and TCP checksum were checked and were correct. 11: Both the IP header and UDP checksum were checked and were correct.
21	VLAN tag detected—type ID of 0x8100. For packets incorporating the stacked VLAN processing feature, this bit will be set if the second VLAN tag has a type ID of 0x8100
20	Priority tag detected—type ID of 0x8100 and null VLAN identifier. For packets incorporating the stacked VLAN processing feature, this bit will be set if the second VLAN tag has a type ID of 0x8100 and a null VLAN identifier.
19:17	VLAN priority—only valid if bit 21 is set.
16	Canonical format indicator (CFI) bit (only valid if bit 21 is set).
15	End of frame—when set the buffer contains the end of a frame. If end of frame is not set, then the only valid status bit is start of frame (bit 14).
14	Start of frame—when set the buffer contains the start of a frame. If both bits 15 and 14 are set, the buffer contains a whole frame.

Table 42-4. Receive Buffer Descriptor Entry (Continued)

Bit	Function
13	This bit has a different meaning depending on whether jumbo frames and ignore FCS modes are enabled. If neither mode is enabled this bit will be zero. **With jumbo frame mode enabled:** (bit 3 set in Network Configuration Register) Additional bit for length of frame (bit[13]), that is concatenated with bits[12:0] **With ignore FCS mode enabled and jumbo frames disabled:** (bit 26 set in Network Configuration Register and bit 3 clear in Network Configuration Register) This indicates per frame FCS status as follows: 0: Frame had good FCS 1: Frame had bad FCS, but was copied to memory as ignore FCS enabled.
12:0	These bits represent the length of the received frame which may or may not include FCS depending on whether FCS discard mode is enabled. **With FCS discard mode disabled:** (bit 17 clear in Network Configuration Register) Least significant 12 bits for length of frame including FCS. If jumbo frames are enabled, these 12 bits are concatenated with bit[13] of the descriptor above. **With FCS discard mode enabled:** (bit 17 set in Network Configuration Register) Least significant 12 bits for length of frame excluding FCS. If jumbo frames are enabled, these 12 bits are concatenated with bit[13] of the descriptor above.

To receive frames, the AHB buffer descriptors must be initialized by writing an appropriate address to bits 31:2 in the first word of each list entry. Bit 0 must be written with zero. Bit 1 is the wrap bit and indicates the last entry in the buffer descriptor list.

The start location of the receive buffer descriptor list must be written with the receive buffer queue base address before reception is enabled (receive enable in the Network Control register). Once reception is enabled, any writes to the Receive Buffer Queue Base Address register are ignored. When read, it will return the current pointer position in the descriptor list, though this is only valid and stable when receive is disabled.

If the filter block indicates that a frame should be copied to memory, the receive data DMA operation starts writing data into the receive buffer. If an error occurs, the buffer is recovered.

An internal counter within the GMAC represents the receive buffer queue pointer and it is not visible through the CPU interface. The receive buffer queue pointer increments by two words after each buffer has been used. It re-initializes to the receive buffer queue base address if any descriptor has its wrap bit set.

As receive AHB buffers are used, the receive AHB buffer manager sets bit zero of the first word of the descriptor to logic one indicating the AHB buffer has been used.

Software should search through the “used” bits in the AHB buffer descriptors to find out how many frames have been received, checking the start of frame and end of frame bits.

To function properly, a 10/100 Ethernet system should have no excessive length frames or frames greater than 128 bytes with CRC errors. Collision fragments will be less than 128 bytes long, therefore it will be a rare occurrence to find a frame fragment in a receive AHB buffer, when using the default value of 128 bytes for the receive buffers size.

If bit zero of the receive buffer descriptor is already set when the receive buffer manager reads the location of the receive AHB buffer, then the buffer has been already used and cannot be used again until software has processed the frame and cleared bit zero. In this case, the “buffer not available” bit in the Receive Status register is set and an interrupt triggered.

42.6.3.2 Transmit AHB Buffers

Frames to transmit are stored in one or more transmit AHB buffers. Transmit frames can be between 1 and 16384 bytes long, so it is possible to transmit frames longer than the maximum length specified in the IEEE 802.3 standard. It should be noted that zero length AHB buffers are allowed and that the maximum number of buffers permitted for each transmit frame is 128.

The start location for each transmit AHB buffer is stored in memory in a list of transmit buffer descriptors at a location pointed to by the transmit buffer queue pointer. The base address for this queue pointer is set in software using the Transmit Buffer Queue Base Address register. Each list entry consists of two words. The first is the byte address of the transmit buffer and the second containing the transmit control and status. For the FIFO-based DMA configured with a 32-bit data path the address of the buffer is a byte address.

Frames can be transmitted with or without automatic CRC generation. If CRC is automatically generated, pad will also be automatically generated to take frames to a minimum length of 64 bytes. When CRC is not automatically generated (as defined in word 1 of the transmit buffer descriptor), the frame is assumed to be at least 64 bytes long and pad is not generated.

An entry in the transmit buffer descriptor list is described in Table 42-5.

To transmit frames, the buffer descriptors must be initialized by writing an appropriate byte address to bits [31:0] in the first word of each descriptor list entry.

The second word of the transmit buffer descriptor is initialized with control information that indicates the length of the frame, whether or not the MAC is to append CRC and whether the buffer is the last buffer in the frame.

After transmission the status bits are written back to the second word of the first buffer along with the used bit. Bit 31 is the used bit which must be zero when the control word is read if transmission is to take place. It is written to one once the frame has been transmitted. Bits[29:20] indicate various transmit error conditions. Bit 30 is the wrap bit which can be set for any buffer within a frame. If no wrap bit is encountered the queue pointer continues to increment.

The Transmit Buffer Queue Base Address register can only be updated while transmission is disabled or halted; otherwise any attempted write will be ignored. When transmission is halted the transmit buffer queue pointer will maintain its value. Therefore when transmission is restarted the next descriptor read from the queue will be from immediately after the last successfully transmitted frame. while transmit is disabled (bit 3 of the Network Control register set low), the transmit buffer queue pointer resets to point to the address indicated by the Transmit Buffer Queue Base Address register. Note that disabling receive does not have the same effect on the receive buffer queue pointer.

Once the transmit queue is initialized, transmit is activated by writing to the transmit start bit (bit 9) of the Network Control register. Transmit is halted when a buffer descriptor with its used bit set is read, a transmit error occurs, or by writing to the transmit halt bit of the Network Control register. Transmission is suspended if a pause frame is received while the pause enable bit is set in the Network Configuration register. Rewriting the start bit while transmission is active is allowed. This is implemented with TXGO variable which is readable in the Transmit Status register at bit location 3. The TXGO variable is reset when:

- Transmit is disabled.
- A buffer descriptor with its ownership bit set is read.
- Bit 10, THALT, of the Network Control register is written.
- There is a transmit error such as too many retries or a transmit underrun.

To set TXGO, write TSTART to the bit 9 of the Network Control register. Transmit halt does not take effect until any ongoing transmit finishes.

The DMA transmission will automatically restart 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, GTXER is asserted and the FCS will be bad.

If transmission stops due to a transmit error or a used bit being read, transmission restarts from the first buffer descriptor of the frame being transmitted when the transmit start bit is rewritten.

Table 42-5. Transmit Buffer Descriptor Entry

Bit	Function
Word 0	
31:0	Byte address of buffer
Word 1	
31	Used—must be zero for the GMAC to read data to the transmit buffer. The GMAC sets this to one for the first buffer of a frame once it has been successfully transmitted. Software must clear this bit before the buffer can be used again.
30	Wrap—marks last descriptor in transmit buffer descriptor list. This can be set for any buffer within the frame.
29	Retry limit exceeded, transmit error detected
28	Transmit underrun—occurs when the start of packet data has been written into the FIFO and either HRESP is not OK, or the transmit data could not be fetched in time, or when buffers are exhausted.
27	Transmit frame corruption due to AHB error—set if an error occurs while midway through reading transmit frame from the AHB, including HRESP errors and buffers exhausted mid frame (if the buffers run out during transmission of a frame then transmission stops, FCS shall be bad and GTXER asserted).
26	Late collision, transmit error detected.
25:23	Reserved
22:20	Transmit IP/TCP/UDP checksum generation offload errors: 000: No Error. 001: The Packet was identified as a VLAN type, but the header was not fully complete, or had an error in it. 010: The Packet was identified as a SNAP type, but the header was not fully complete, or had an error in it. 011: The Packet was not of an IP type, or the IP packet was invalidly short, or the IP was not of type IPv4/IPv6. 100: The Packet was not identified as VLAN, SNAP or IP. 101: Non supported packet fragmentation occurred. For IPv4 packets, the IP checksum was generated and inserted. 110: Packet type detected was not TCP or UDP. TCP/UDP checksum was therefore not generated. For IPv4 packets, the IP checksum was generated and inserted. 111: A premature end of packet was detected and the TCP/UDP checksum could not be generated.
19:17	Reserved
16	No CRC to be appended by MAC. When set, this implies that the data in the buffers already contains a valid CRC, hence no CRC or padding is to be appended to the current frame by the MAC. This control bit must be set for the first buffer in a frame and will be ignored for the subsequent buffers of a frame. Note that this bit must be clear when using the transmit IP/TCP/UDP checksum generation offload, otherwise checksum generation and substitution will not occur.
15	Last buffer, when set this bit will indicate the last buffer in the current frame has been reached.
14	Reserved
13:0	Length of buffer

42.6.3.3 DMA Bursting on the AHB

The DMA will always use SINGLE, or INCR type AHB accesses for buffer management operations. When performing data transfers, the AHB burst length used can be programmed using bits 4:0 of the DMA Configuration register so that either SINGLE, INCR or fixed length incrementing bursts (INCR4, INCR8 or INCR16) are used where possible.

When there is enough space and enough data to be transferred, the programmed fixed length bursts will be used. If there is not enough data or space available, for example when at the beginning or the end of a buffer, SINGLE type accesses are used. Also SINGLE type accesses are used at 1024 byte boundaries, so that the 1 Kbyte boundaries are not burst over as per AHB requirements.

The DMA will not terminate a fixed length burst early, unless an error condition occurs on the AHB or if receive or transmit are disabled in the Network Control register.

42.6.4 MAC Transmit Block

The MAC transmitter can operate in either half duplex or full duplex mode and transmits frames in accordance with the Ethernet IEEE 802.3 standard. In half duplex mode, the CSMA/CD protocol of the IEEE 802.3 specification is followed.

A small input buffer receives data through the FIFO interface which will extract data in 32-bit form. All subsequent processing prior to the final output is performed in bytes.

Transmit data can be output using the MII interface.

Frame assembly starts by adding preamble and the start frame delimiter. Data is taken from the transmit FIFO interface a word at a time.

If necessary, padding is added to take the frame length to 60 bytes. CRC is calculated using an order 32-bit polynomial. This is inverted and appended to the end of the frame taking the frame length to a minimum of 64 bytes. If the no CRC bit is set in the second word of the last buffer descriptor of a transmit frame, neither pad nor CRC are appended. The no CRC bit can also be set through the FIFO interface.

In full duplex mode (at all data rates), frames are transmitted immediately. Back to back frames are transmitted at least 96 bit times apart to guarantee the interframe gap.

In half duplex mode, the transmitter checks carrier sense. If asserted, the transmitter waits for the signal to become inactive, and then starts transmission after the interframe gap of 96 bit times. If the collision signal is asserted during transmission, the transmitter will transmit a jam sequence of 32 bits taken from the data register and then retry transmission after the back off time has elapsed. If the collision occurs during either the preamble or Start Frame Delimiter (SFD), then these fields will be completed prior to generation of the jam sequence.

The back off time is based on an XOR of the 10 least significant bits of the data coming from the transmit FIFO interface and a 10-bit pseudo random number generator. The number of bits used depends on the number of collisions seen. After the first collision 1 bit is used, then the second 2 bits and so on up to the maximum of 10 bits. All 10 bits are used above ten collisions. An error will be indicated and no further attempts will be made if 16 consecutive attempts cause collision. This operation is compliant with the description in Clause 4.2.3.2.5 of the IEEE 802.3 standard which refers to the truncated binary exponential back off algorithm.

In 10/100 mode, both collisions and late collisions are treated identically, and back off and retry will be performed up to 16 times. This condition is reported in the transmit buffer descriptor word 1 (late collision, bit 26) and also in the Transmit Status register (late collision, bit 7). An interrupt can also be generated (if enabled) when this exception occurs, and bit 5 in the Interrupt Status register will be set.

In all modes of operation, if the transmit DMA underruns, a bad CRC is automatically appended using the same mechanism as jam insertion and the GTXER signal is asserted. For a properly configured system this should never happen.

By setting when bit 28 is set in the Network Configuration register, the Inter Packet Gap (IPG) may be stretched beyond 96 bits depending on the length of the previously transmitted frame and the value written to the IPG Stretch register (GMAC_IPGS). The least significant 8 bits of the IPG Stretch register multiply the previous frame length (including preamble). The next significant 8 bits (+1 so as not to get a divide by zero) divide the frame length to generate the IPG. IPG stretch only works in full duplex mode and when bit 28 is set in the Network Configuration register. The IPG Stretch register cannot be used to shrink the IPG below 96 bits.

If the back pressure bit is set in the Network Control register, or if the HDFC configuration bit is set in the GMAC_UR register (10M or 100M half duplex mode), the transmit block transmits 64 bits of data, which can consist of 16 nibbles of 1011 or in bit rate mode 64 1s, whenever it sees an incoming frame to force a collision. This provides a way of implementing flow control in half duplex mode.

42.6.5 MAC Receive Block

All processing within the MAC receive block is implemented using a 16-bit data path. The MAC receive block checks for valid preamble, FCS, alignment and length, presents received frames to the FIFO interface and stores the frame destination address for use by the address checking block.

If, during the frame reception, the frame is found to be too long, a bad frame indication is sent to the FIFO interface. The receiver logic ceases to send data to memory as soon as this condition occurs.

At end of frame reception the receive block indicates to the DMA block whether the frame is good or bad. The DMA block will recover the current receive buffer if the frame was bad.

Ethernet frames are normally stored in DMA memory complete with the FCS. Setting the FCS remove bit in the network configuration (bit 17) causes frames to be stored without their corresponding FCS. The reported frame length field is reduced by four bytes to reflect this operation.

The receive block signals to the register block to increment the alignment, CRC (FCS), short frame, long frame, jabber or receive symbol errors when any of these exception conditions occur.

If bit 26 is set in the network configuration, CRC errors will be ignored and CRC errored frames will not be discarded, though the Frame Check Sequence Errors statistic register will still be incremented. Additionally, if not enabled for jumbo frames mode, then bit[13] of the receiver descriptor word 1 will be updated to indicate the FCS validity for the particular frame. This is useful for applications such as EtherCAT whereby individual frames with FCS errors must be identified.

Received frames can be checked for length field error by setting the length field error frame discard bit of the Network Configuration register (bit-16). When this bit is set, the receiver compares a frame's measured length with the length field (bytes 13 and 14) extracted from the frame. The frame is discarded if the measured length is shorter. This checking procedure is for received frames between 64 bytes and 1518 bytes in length.

Frames where the length field is greater than or equal to 0x0600 hex will not be checked.

42.6.6 Checksum Offload for IP, TCP and UDP

The GMAC can be programmed to perform IP, TCP and UDP checksum offloading in both receive and transmit directions, which is enabled by setting bit 24 in the Network Configuration register for receive.

IPv4 packets contain a 16-bit checksum field, which is the 16-bit 1’s complement of the 1’s complement sum of all 16-bit words in the header. TCP and UDP packets contain a 16-bit checksum field, which is the 16-bit 1’s complement of the 1’s complement sum of all 16-bit words in the header, the data and a conceptual IP pseudo header.

To calculate these checksums in software requires each byte of the packet to be processed. For TCP and UDP this can use a large amount of processing power. Offloading the checksum calculation to hardware can result in significant performance improvements.

For IP, TCP or UDP checksum offload to be useful, the operating system containing the protocol stack must be aware that this offload is available so that it can make use of the fact that the hardware can either generate or verify the checksum.

42.6.6.1 Receiver Checksum Offload

When receive checksum offloading is enabled in the GMAC, the IPv4 header checksum is checked as per RFC 791, where the packet meets the following criteria:

- If present, the VLAN header must be four octets long and the CFI bit must not be set.

- Encapsulation must be RFC 894 Ethernet Type Encoding or RFC 1042 SNAP Encoding.
- IPv4 packet
- IP header is of a valid length

The GMAC also checks the TCP checksum as per RFC 793, or the UDP checksum as per RFC 768, if the following criteria are met:

- IPv4 or IPv6 packet
- Good IP header checksum (if IPv4)
- No IP fragmentation
- TCP or UDP packet

When an IP, TCP or UDP frame is received, the receive buffer descriptor gives an indication if the GMAC was able to verify the checksums. There is also an indication if the frame had SNAP encapsulation. These indication bits will replace the type ID match indication bits when the receive checksum offload is enabled. For details of these indication bits refer to Table 42-4 “Receive Buffer Descriptor Entry”.

42.6.7 MAC Filtering Block

The filter block determines which frames should be written to the FIFO interface and on to the DMA.

Whether a frame is passed depends on what is enabled in the Network Configuration register, the state of the external matching pins, the contents of the specific address, type and Hash registers and the frame's destination address and type field.

If bit 25 of the Network Configuration register is not set, a frame will not be copied to memory if the GMAC is transmitting in half duplex mode at the time a destination address is received.

Ethernet frames are transmitted a byte at a time, least significant bit first. The first six bytes (48 bits) of an Ethernet frame make up the destination address. The first bit of the destination address, which is the LSB of the first byte of the frame, is the group or individual bit. This is one for multicast addresses and zero for unicast. The all ones address is the broadcast address and a special case of multicast.

The GMAC supports recognition of four specific addresses. Each specific address requires two registers, Specific Address Bottom register and Specific Address Top register. Specific Address Bottom register stores the first four bytes of the destination address and Specific Address Top register 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 Bottom register is written. They are activated when Specific Address Top register is written. If a receive frame address matches an active address, the frame is written to the FIFO interface and on to DMA memory.

Frames may be filtered using the type ID field for matching. Four type ID registers exist in the register address space and each can be enabled for matching by writing a one to the MSB (bit 31) of the respective register. When a frame is received, the matching is implemented as an OR function of the various types of match.

The contents of each type ID register (when enabled) are compared against the length/type ID of the frame being received (e.g., bytes 13 and 14 in non-VLAN and non-SNAP encapsulated frames) and copied to memory if a match is found. The encoded type ID match bits (Word 0, Bit 22 and Bit 23) in the receive buffer descriptor status are set indicating which type ID register generated the match, if the receive checksum offload is disabled.

The reset state of the type ID registers is zero, hence each is initially disabled.

The following example illustrates the use of the address and type ID match registers for a MAC address of 21:43:65:87:A9:CB:

Preamble	55
SFD	D5
DA (Octet 0 - LSB)	21
DA (Octet 1)	43
DA (Octet 2)	65
DA (Octet 3)	87
DA (Octet 4)	A9
DA (Octet 5 - MSB)	CB
SA (LSB)	00(1)
SA	00(1)
SA	00(1)
SA	00(1)
SA	00(1)
SA (MSB)	00(1)
Type ID (MSB)	43
Type ID (LSB)	21

Note: 1. Contains the address of the transmitting device

The sequence above shows the beginning of an Ethernet frame. Byte order of transmission is from top to bottom as shown. For a successful match to specific address 1, the following address matching registers must be set up:

Specific Address 1 Bottom register (GMAC_SAB1) (Address 0x088) 0x87654321

Specific Address 1 Top register (GMAC_SAT1) (Address 0x08C) 0x0000CBA9

For a successful match to the type ID, the following Type ID Match 1 register must be set up:

Type ID Match 1 register (GMAC_TIDM1) (Address 0x0A8) 0x80004321

42.6.8 Broadcast Address

Frames with the broadcast address of 0xFFFFFFFFFFFF are stored to memory only if the 'no broadcast' bit in the Network Configuration register is set to zero.

42.6.9 Hash Addressing

The hash address register is 64 bits long and takes up two locations in the memory map. The least significant bits are stored in Hash Register Bottom and the most significant bits in Hash Register Top.

The unicast hash enable and the multicast hash enable bits in the Network Configuration register enable the reception of hash matched frames. The destination address is reduced to a 6-bit index into the 64-bit Hash register using the following hash function: The hash function is an XOR of every sixth bit of the destination address.

```
hash_index[05] = da[05] ^ da[11] ^ da[17] ^ da[23] ^ da[29] ^ da[35] ^ da[41] ^
da[47]
hash_index[04] = da[04] ^ da[10] ^ da[16] ^ da[22] ^ da[28] ^ da[34] ^ da[40] ^
da[46]
hash_index[03] = da[03] ^ da[09] ^ da[15] ^ da[21] ^ da[27] ^ da[33] ^ da[39] ^
da[45]
```

```
hash_index[02] = da[02] ^ da[08] ^ da[14] ^ da[20] ^ da[26] ^ da[32] ^ da[38] ^
da[44]
hash_index[01] = da[01] ^ da[07] ^ da[13] ^ da[19] ^ da[25] ^ da[31] ^ da[37] ^
da[43]
hash_index[00] = da[00] ^ da[06] ^ da[12] ^ da[18] ^ da[24] ^ da[30] ^ da[36] ^
da[42]
```

`da[0]`

represents the least significant bit of the first byte received, that is, the multicast/unicast indicator, and `da[47]` represents the most significant bit of the last byte received.

If the hash index points to a bit that is set in the Hash register then the frame will be matched according to whether the frame is multicast or unicast.

A multicast match will be signalled if the multicast hash enable bit is set, `da[0]` is logic 1 and the hash index points to a bit set in the Hash register.

A unicast match will be signalled if the unicast hash enable bit is set, `da[0]` is logic 0 and the hash index points to a bit set in the Hash register.

To receive all multicast frames, the Hash register should be set with all ones and the multicast hash enable bit should be set in the Network Configuration register.

42.6.10 Copy all Frames (Promiscuous Mode)

If the Copy All Frames bit is set in the Network Configuration register then all frames (except those that are too long, too short, have FCS errors or have GRXER asserted during reception) will be copied to memory. Frames with FCS errors will be copied if bit 26 is set in the Network Configuration register.

42.6.11 Disable Copy of Pause Frames

Pause frames can be prevented from being written to memory by setting the disable copying of pause frames control bit 23 in the Network Configuration register. When set, pause frames are not copied to memory regardless of the Copy All Frames bit, whether a hash match is found, a type ID match is identified or if a destination address match is found.

42.6.12 VLAN Support

The following table describes an Ethernet encoded 802.1Q VLAN tag.

Table 42-6. 802.1Q VLAN Tag

TPID (Tag Protocol Identifier) 16 bits	TCI (Tag Control Information) 16 bits
0x8100	First 3 bits priority, then CFI bit, last 12 bits VID

The VLAN tag is inserted at the 13th byte of the frame adding an extra four bytes to the frame. To support these extra four bytes, the GMAC can accept frame lengths up to 1536 bytes by setting bit 8 in the Network Configuration register.

If the VID (VLAN identifier) is null (0x000) this indicates a priority-tagged frame.

The following bits in the receive buffer descriptor status word give information about VLAN tagged frames:-

- Bit 21 set if receive frame is VLAN tagged (i.e., type ID of 0x8100).
- Bit 20 set if receive frame is priority tagged (i.e., type ID of 0x8100 and null VID). (If bit 20 is set, bit 21 will be set also.)
- Bit 19, 18 and 17 set to priority if bit 21 is set.
- Bit 16 set to CFI if bit 21 is set.

The GMAC can be configured to reject all frames except VLAN tagged frames by setting the discard non-VLAN frames bit in the Network Configuration register.

42.6.13 IEEE 1588 Support

IEEE 1588 is a standard for precision time synchronization in local area networks. It works with the exchange of special Precision Time Protocol (PTP) frames. The PTP messages can be transported over IEEE 802.3/Ethernet, over Internet Protocol Version 4 or over Internet Protocol Version 6 as described in the annex of IEEE P1588.D2.1.

The GMAC indicates the message time-stamp point (asserted on the start packet delimiter and de-asserted at end of frame) for all frames and the passage of PTP event frames (asserted when a PTP event frame is detected and de-asserted at end of frame).

IEEE 802.1AS is a subset of IEEE 1588. One difference is that IEEE 802.1AS uses the Ethernet multicast address 0180C200000E for sync frame recognition whereas IEEE 1588 does not. GMAC is designed to recognize sync frames with both IEEE 802.1AS and IEEE 1588 addresses and so can support both 1588 and 802.1AS frame recognition simultaneously.

Synchronization between master and slave clocks is a two stage process.

First, the offset between the master and slave clocks is corrected by the master sending a sync frame to the slave with a follow up frame containing the exact time the sync frame was sent. Hardware assist modules at the master and slave side detect exactly when the sync frame was sent by the master and received by the slave. The slave then corrects its clock to match the master clock.

Second, the transmission delay between the master and slave is corrected. The slave sends a delay request frame to the master which sends a delay response frame in reply. Hardware assist modules at the master and slave side detect exactly when the delay request frame was sent by the slave and received by the master. The slave will now have enough information to adjust its clock to account for delay. For example, if the slave was assuming zero delay, the actual delay will be half the difference between the transmit and receive time of the delay request frame (assuming equal transmit and receive times) because the slave clock will be lagging the master clock by the delay time already.

The time-stamp is taken when the message time-stamp point passes the clock time-stamp point. This can generate an interrupt if enabled (GMAC_IER). However, MAC Filtering configuration is needed to actually ‘copy’ the message to memory. For Ethernet, the message time-stamp point is the SFD and the clock time-stamp point is the MII interface. (The IEEE 1588 specification refers to sync and delay_req messages as event messages as these require time-stamping. These events are captured in the registers GMAC_EFTx and GMAC_EFRx, respectively. Follow up, delay response and management messages do not require time-stamping and are referred to as general messages.)

1588 version 2 defines two additional PTP event messages. These are the peer delay request (Pdelay_Req) and peer delay response (Pdelay_Resp) messages. These events are captured in the registers GMAC_PEFTx and GMAC_PEFRx, respectively. These messages are used to calculate the delay on a link. Nodes at both ends of a link send both types of frames (regardless of whether they contain a master or slave clock). The Pdelay_Resp message contains the time at which a Pdelay_Req was received and is itself an event message. The time at which a Pdelay_Resp message is received is returned in a Pdelay_Resp_Follow_Up message.

1588 version 2 introduces transparent clocks of which there are two kinds, peer-to-peer (P2P) and end-to-end (E2E). Transparent clocks measure the transit time of event messages through a bridge and amend a correction field within the message to allow for the transit time. P2P transparent clocks additionally correct for the delay in the receive path of the link using the information gathered from the peer delay frames. With P2P transparent clocks delay_req messages are not used to measure link delay. This simplifies the protocol and makes larger systems more stable.

The GMAC recognizes four different encapsulations for PTP event messages:

1. 1588 version 1 (UDP/IPv4 multicast)
2. 1588 version 2 (UDP/IPv4 multicast)
3. 1588 version 2 (UDP/IPv6 multicast)
4. 1588 version 2 (Ethernet multicast)

Table 42-7. Example of Sync Frame in 1588 Version 1 Format

Frame Segment	Value
Preamble/SFD	55555555555555D5
DA (Octets 0–5)	—
SA (Octets 6–11)	—
Type (Octets 12–13)	0800
IP stuff (Octets 14–22)	—
UDP (Octet 23)	11
IP stuff (Octets 24–29)	—
IP DA (Octets 30–32)	E00001
IP DA (Octet 33)	81 or 82 or 83 or 84
Source IP port (Octets 34–35)	—
Dest IP port (Octets 36–37)	013F
Other stuff (Octets 38–42)	—
Version PTP (Octet 43)	01
Other stuff (Octets 44–73)	—
Control (Octet 74)	00
Other stuff (Octets 75–168)	—

Table 42-8. Example of Delay Request Frame in 1588 Version 1 Format

Frame Segment	Value
Preamble/SFD	55555555555555D5
DA (Octets 0–5)	—
SA (Octets 6–11)	—
Type (Octets 12–13)	0800
IP stuff (Octets 14–22)	—
UDP (Octet 23)	11
IP stuff (Octets 24–29)	—
IP DA (Octets 30–32)	E00001
IP DA (Octet 33)	81 or 82 or 83 or 84
Source IP port (Octets 34–35)	—
Dest IP port (Octets 36–37)	013F
Other stuff (Octets 38–42)	—

Table 42-8. Example of Delay Request Frame in 1588 Version 1 Format (Continued)

Frame Segment	Value
Version PTP (Octet 43)	01
Other stuff (Octets 44–73)	—
Control (Octet 74)	01
Other stuff (Octets 75–168)	—

For 1588 version 1 messages, sync and delay request frames are indicated by the GMAC if the frame type field indicates TCP/IP, UDP protocol is indicated, the destination IP address is 224.0.1.129/130/131 or 132, the destination UDP port is 319 and the control field is correct.

The control field is 0x00 for sync frames and 0x01 for delay request frames.

For 1588 version 2 messages, the type of frame is determined by looking at the message type field in the first byte of the PTP frame. Whether a frame is version 1 or version 2 can be determined by looking at the version PTP field in the second byte of both version 1 and version 2 PTP frames.

In version 2 messages sync frames have a message type value of 0x0, delay_req have 0x1, Pdelay_Req have 0x2 and Pdelay_Resp have 0x3.

Table 42-9. Example of Sync Frame in 1588 Version 2 (UDP/IPv4) Format

Frame Segment	Value
Preamble/SFD	55555555555555D5
DA (Octets 0–5)	—
SA (Octets 6–11)	—
Type (Octets 12–13)	0800
IP stuff (Octets 14–22)	—
UDP (Octet 23)	11
IP stuff (Octets 24–29)	—
IP DA (Octets 30–33)	E0000181
Source IP port (Octets 34–35)	—
Dest IP port (Octets 36–37)	013F
Other stuff (Octets 38–41)	—
Message type (Octet 42)	00
Version PTP (Octet 43)	02

Table 42-10. Example of Pdelay_Req Frame in 1588 Version 2 (UDP/IPv4) Format

Frame Segment	Value
Preamble/SFD	55555555555555D5
DA (Octets 0–5)	—
SA (Octets 6–11)	—
Type (Octets 12–13)	0800
IP stuff (Octets 14–22)	—
UDP (Octet 23)	11

Table 42-10. Example of Pdelay_Req Frame in 1588 Version 2 (UDP/IPv4) Format (Continued)

Frame Segment	Value
IP stuff (Octets 24–29)	—
IP DA (Octets 30–33)	E000006B
Source IP port (Octets 34–35)	—
Dest IP port (Octets 36–37)	013F
Other stuff (Octets 38–41)	—
Message type (Octet 42)	02
Version PTP (Octet 43)	02

Table 42-11. Example of Sync Frame in 1588 Version 2 (UDP/IPv6) Format

Frame Segment	Value
Preamble/SFD	55555555555555D5
DA (Octets 0–5)	—
SA (Octets 6–11)	—
Type (Octets 12–13)	86dd
IP stuff (Octets 14–19)	—
UDP (Octet 20)	11
IP stuff (Octets 21–37)	—
IP DA (Octets 38–53)	FF0X00000000018
Source IP port (Octets 54–55)	—
Dest IP port (Octets 56–57)	013F
Other stuff (Octets 58–61)	—
Message type (Octet 62)	00
Other stuff (Octets 63–93)	—
Version PTP (Octet 94)	02

Table 42-12. Example of Pdelay_Resp Frame in 1588 Version 2 (UDP/IPv6) Format

Frame Segment	Value
Preamble/SFD	55555555555555D5
DA (Octets 0–5)	—
SA (Octets 6–11)	—
Type (Octets 12–13)	86dd
IP stuff (Octets 14–19)	—
UDP (Octet 20)	11
IP stuff (Octets 21–37)	—
IP DA (Octets 38–53)	FF0200000000006B
Source IP port (Octets 54–55)	—

Table 42-12. Example of Pdelay_Resp Frame in 1588 Version 2 (UDP/IPv6) Format (Continued)

Frame Segment	Value
Dest IP port (Octets 56–57)	013F
Other stuff (Octets 58–61)	—
Message type (Octet 62)	03
Other stuff (Octets 63–93)	—
Version PTP (Octet 94)	02

For the multicast address 011B19000000 sync and delay request frames are recognized depending on the message type field, 00 for sync and 01 for delay request.

Table 42-13. Example of Sync Frame in 1588 Version 2 (Ethernet Multicast) Format

Frame Segment	Value
Preamble/SFD	55555555555555D5
DA (Octets 0–5)	011B19000000
SA (Octets 6–11)	—
Type (Octets 12–13)	88F7
Message type (Octet 14)	00
Version PTP (Octet 15)	02

Pdelay request frames need a special multicast address so they can pass through ports blocked by the spanning tree protocol. For the multicast address 0180C200000E sync, Pdelay_Req and Pdelay_Resp frames are recognized depending on the message type field, 00 for sync, 02 for pdelay request and 03 for pdelay response.

Table 42-14. Example of Pdelay_Req Frame in 1588 Version 2 (Ethernet Multicast) Format

Frame Segment	Value
Preamble/SFD	55555555555555D5
DA (Octets 0–5)	0180C200000E
SA (Octets 6–11)	—
Type (Octets 12–13)	88F7
Message type (Octet 14)	00
Version PTP (Octet 15)	02

42.6.14 Time Stamp Unit

The TSU consists of a timer and registers to capture the time at which PTP event frames cross the message timestamp point. An interrupt is issued when a capture register is updated.

The timer is implemented as a 62-bit register with the upper 32 bits counting seconds and the lower 30 bits counting nanoseconds. The lower 30 bits roll over when they have counted to one second. An interrupt is generated when the seconds increment. The timer value can be read, written and adjusted through the APB interface.

The amount by which the timer increments each clock cycle is controlled by the timer increment registers (GMAC_TI). Bits 7:0 are the default increment value in nanoseconds and an additional 16 bits of sub-nanosecond resolution are available using the Timer Increment Sub-nanoseconds register (GMAC_TISUBN). If the rest of the

register is written with zero, the timer increments by the value in [7:0], plus the value of GMAC_TISUBN, each clock cycle.

The GMAC_TISUBN register allows a resolution of approximately 15 femtoseconds.

Bits 15:8 of the increment register are the alternative increment value in nanoseconds and bits 23:16 are the number of increments after which the alternative increment value is used. If 23:16 are zero then the alternative increment value will never be used.

Taking the example of 10.2 MHz, there are 102 cycles every ten microseconds or 51 every five microseconds. So a timer with a 10.2 MHz clock source is constructed by incrementing by 98 ns for fifty cycles and then incrementing by 100 ns ($98 \times 50 + 100 = 5000$). This is programmed by setting the 1588 Timer Increment register to 0x00326462.

For a 49.8 MHz clock source it would be 20 ns for 248 cycles followed by an increment of 40 ns ($20 \times 248 + 40 = 5000$) programmed as 0x00F82814.

Having eight bits for the “number of increments” field allows frequencies up to 50 MHz to be supported with 200 kHz resolution.

Without the alternative increment field the period of the clock would be limited to an integer number of nanoseconds, resulting in supported clock frequencies of 8, 10, 20, 25, 40, 50, 100, 125, 200 and 250 MHz.

There are six additional 62-bit registers that capture the time at which PTP event frames are transmitted and received. An interrupt is issued when these registers are updated. The TSU timer count value can be compared to a programmable comparison value. For the comparison, the 32 bits of the seconds value and the upper 22 bits of the nanoseconds value are used. An interrupt can also be generated (if enabled) when the TSU timer count value and comparison value are equal, mapped to bit 29 of the Interrupt Status register.

42.6.15 MAC 802.3 Pause Frame Support

Note: See Clause 31, and Annex 31A and 31B of the IEEE standard 802.3 for a full description of MAC 802.3 pause operation.

The following table shows the start of a MAC 802.3 pause frame.

Table 42-15. Start of an 802.3 Pause Frame

Address		Type (MAC Control Frame)	Pause	
Destination	Source		Opcode	Time
0x0180C2000001	6 bytes	0x8808	0x0001	2 bytes

The GMAC supports both hardware controlled pause of the transmitter, upon reception of a pause frame, and hardware generated pause frame transmission.

42.6.15.1 802.3 Pause Frame Reception

Bit 13 of the Network Configuration register is the pause enable control for reception. If this bit is set, transmission pauses if a non zero pause quantum frame is received.

If a valid pause frame is received, then the Pause Time register is updated with the new frame's pause time, regardless of whether a previous pause frame is active or not. An interrupt (either bit 12 or bit 13 of the Interrupt Status register) is triggered when a pause frame is received, but only if the interrupt has been enabled (bit 12 and bit 13 of the Interrupt Mask register). Pause frames received with non zero quantum are indicated through the interrupt bit 12 of the Interrupt Status register. Pause frames received with zero quantum are indicated on bit 13 of the Interrupt Status register.

Once the Pause Time register is loaded and the frame currently being transmitted has been sent, no new frames are transmitted until the pause time reaches zero. The loading of a new pause time, and hence the pausing of transmission, only occurs when the GMAC is configured for full duplex operation. If the GMAC is configured for

half duplex there will be no transmission pause, but the pause frame received interrupt will still be triggered. A valid pause frame is defined as having a destination address that matches either the address stored in Specific Address 1 register or if it matches the reserved address of 0x0180C2000001. It must also have the MAC control frame type ID of 0x8808 and have the pause opcode of 0x0001.

Pause frames that have frame check sequence (FCS) or other errors will be treated as invalid and will be discarded. 802.3 Pause frames that are received after Priority-based Flow Control (PFC) has been negotiated will also be discarded. Valid pause frames received will increment the Pause Frames Received statistic register.

The Pause Time register decrements every 512 bit times once transmission has stopped. For test purposes, the retry test bit can be set (bit 12 in the Network Configuration register) which causes the Pause Time register to decrement every GTXCK cycle once transmission has stopped.

The interrupt (bit 13 in the Interrupt Status register) is asserted whenever the Pause Time register decrements to zero (assuming it has been enabled by bit 13 in the Interrupt Mask register). This interrupt is also set when a zero quantum pause frame is received.

42.6.15.2 802.3 Pause Frame Transmission

Automatic transmission of pause frames is supported through the transmit pause frame bits of the Network Control register. If either bit 11 or bit 12 of the Network Control register is written with logic 1, an 802.3 pause frame will be transmitted, providing full duplex is selected in the Network Configuration register and the transmit block is enabled in the Network Control register.

Pause frame transmission will happen immediately if transmit is inactive or if transmit is active between the current frame and the next frame due to be transmitted.

Transmitted pause frames comprise the following:

- A destination address of 01-80-C2-00-00-01
- A source address taken from Specific Address 1 register
- A type ID of 88-08 (MAC control frame)
- A pause opcode of 00-01
- A Pause Quantum register
- Fill of 00 to take the frame to minimum frame length
- Valid FCS

The pause quantum used in the generated frame will depend on the trigger source for the frame as follows:

- If bit 11 is written with a one, the pause quantum will be taken from the Transmit Pause Quantum register. The Transmit Pause Quantum register resets to a value of 0xFFFF giving maximum pause quantum as default.
- If bit 12 is written with a one, the pause quantum will be zero.

After transmission, a pause frame transmitted interrupt will be generated (bit 14 of the Interrupt Status register).

Pause frames can also be transmitted by the MAC using normal frame transmission methods.

42.6.16 MAC PFC Priority-based Pause Frame Support

Note: Refer to the 802.1Qbb standard for a full description of priority-based pause operation.

The following table shows the start of a Priority-based Flow Control (PFC) pause frame.

Table 42-16. Start of a PFC Pause Frame

Address		Type (Mac Control Frame)	Pause Opcode	Priority Enable Vector	Pause Time
Destination	Source				
0x0180C2000001	6 bytes	0x8808	0x1001	2 bytes	8 × 2 bytes

The GMAC supports PFC priority-based pause transmission and reception. Before PFC pause frames can be received, bit 16 of the Network Control register must be set.

42.6.16.1 PFC Pause Frame Reception

The ability to receive and decode priority-based pause frames is enabled by setting bit 16 of the Network Control register. When this bit is set, the GMAC will match either classic 802.3 pause frames or PFC priority-based pause frames. Once a priority-based pause frame has been received and matched, then from that moment on the GMAC will only match on priority-based pause frames (this is an 802.1Qbb requirement, known as PFC negotiation). Once priority-based pause has been negotiated, any received 802.3x format pause frames will not be acted upon.

If a valid priority-based pause frame is received then the GMAC will decode the frame and determine which, if any, of the eight priorities require to be paused. Up to eight Pause Time registers are then updated with the eight pause times extracted from the frame regardless of whether a previous pause operation is active or not. An interrupt (either bit 12 or bit 13 of the Interrupt Status register) is triggered when a pause frame is received, but only if the interrupt has been enabled (bit 12 and bit 13 of the Interrupt Mask register). Pause frames received with non zero quantum are indicated through the interrupt bit 12 of the Interrupt Status register. Pause frames received with zero quantum are indicated on bit 13 of the Interrupt Status register. The loading of a new pause time only occurs when the GMAC is configured for full duplex operation. If the GMAC is configured for half duplex, the pause time counters will not be loaded, but the pause frame received interrupt will still be triggered. A valid pause frame is defined as having a destination address that matches either the address stored in Specific Address 1 register or if it matches the reserved address of 0x0180C2000001. It must also have the MAC control frame type ID of 0x8808 and have the pause opcode of 0x0101.

Pause frames that have frame check sequence (FCS) or other errors will be treated as invalid and will be discarded. Valid pause frames received will increment the Pause Frames Received Statistic register.

The Pause Time registers decrement every 512 bit times immediately following the PFC frame reception. For test purposes, the retry test bit can be set (bit 12 in the Network Configuration register) which causes the Pause Time register to decrement every GRXCK cycle once transmission has stopped.

The interrupt (bit 13 in the Interrupt Status register) is asserted whenever the Pause Time register decrements to zero (assuming it has been enabled by bit 13 in the Interrupt Mask register). This interrupt is also set when a zero quantum pause frame is received.

42.6.16.2 PFC Pause Frame Transmission

Automatic transmission of pause frames is supported through the transmit priority-based pause frame bit of the Network Control register. If bit 17 of the Network Control register is written with logic 1, a PFC pause frame will be transmitted providing full duplex is selected in the Network Configuration register and the transmit block is enabled in the Network Control register. When bit 17 of the Network Control register is set, the fields of the priority-based pause frame will be built using the values stored in the Transmit PFC Pause register.

Pause frame transmission will happen immediately if transmit is inactive or if transmit is active between the current frame and the next frame due to be transmitted.

Transmitted pause frames comprise the following:

- A destination address of 01-80-C2-00-00-01
- A source address taken from Specific Address 1 register
- A type ID of 88-08 (MAC control frame)
- A pause opcode of 01-01
- A priority enable vector taken from Transmit PFC Pause register
- 8 Pause Quantum registers
- Fill of 00 to take the frame to minimum frame length
- Valid FCS

The Pause Quantum registers used in the generated frame will depend on the trigger source for the frame as follows:

- If bit 17 of the Network Control register is written with a one, then the priority enable vector of the priority-based pause frame will be set equal to the value stored in the Transmit PFC Pause register [7:0]. For each entry equal to zero in the Transmit PFC Pause register [15:8], the pause quantum field of the pause frame associated with that entry will be taken from the Transmit Pause Quantum register. For each entry equal to one in the Transmit PFC Pause register [15:8], the pause quantum associated with that entry will be zero.
- The Transmit Pause Quantum register resets to a value of 0xFFFF giving maximum pause quantum as default.

After transmission, a pause frame transmitted interrupt will be generated (bit 14 of the Interrupt Status register).

PFC Pause frames can also be transmitted by the MAC using normal frame transmission methods.

42.6.17 PHY Interface

Different PHY interfaces are supported by the Ethernet MAC:

- MII

The MII interface is provided for 10/100 operation and uses txd[3:0] and rxd[3:0].

42.6.18 10/100 Operation

The 10/100 Mbps speed bit in the Network Configuration register is used to select between 10 Mbps and 100 Mbps.

42.6.19 Jumbo Frames

The jumbo frames enable bit in the Network Configuration register allows the GMAC, in its default configuration, to receive jumbo frames up to 10240 bytes in size. This operation does not form part of the IEEE 802.3 specification and is normally disabled. When jumbo frames are enabled, frames received with a frame size greater than 10240 bytes are discarded.

42.7 Programming Interface

42.7.1 Initialization

42.7.1.1 Configuration

Initialization of the GMAC configuration (e.g., loop back mode, frequency ratios) must be done while the transmit and receive circuits are disabled. See the description of the Network Control register and Network Configuration register earlier in this document.

To change loop back mode, the following sequence of operations must be followed:

1. Write to Network Control register to disable transmit and receive circuits.
2. Write to Network Control register to change loop back mode.
3. Write to Network Control register to re-enable transmit or receive circuits.

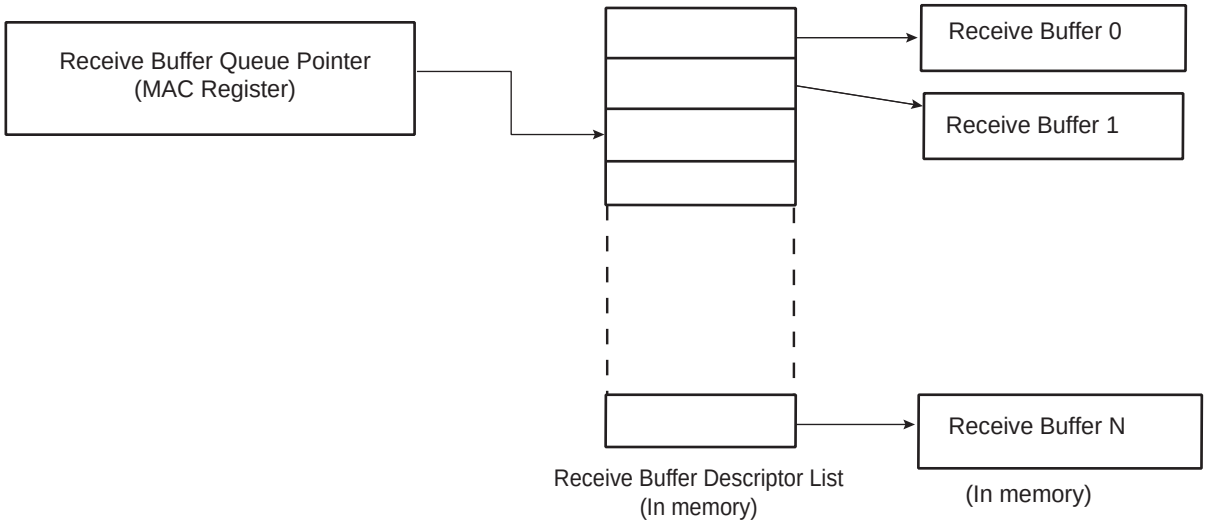
Note: These writes to the Network Control register cannot be combined in any way.

42.7.1.2 Receive Buffer List

Receive data is written to areas of data (i.e., buffers) in system memory. These buffers are listed in another data structure that also resides in main memory. This data structure (receive buffer queue) is a sequence of descriptor entries as defined in Table 42-4 “Receive Buffer Descriptor Entry”.

The Receive Buffer Queue Pointer register points to this data structure.

Figure 42-2. Receive Buffer List



To create the list of buffers:

1. Allocate a number (N) of buffers of X bytes in system memory, where X is the DMA buffer length programmed in the DMA Configuration register.
2. Allocate an area 8N bytes for the receive buffer descriptor list in system memory and create N entries in this list. Mark all entries in this list as owned by GMAC, i.e., bit 0 of word 0 set to 0.
3. Mark the last descriptor in the queue with the wrap bit (bit 1 in word 0 set to 1).
4. Write address of receive buffer descriptor list and control information to GMAC register receive buffer queue pointer
5. The receive circuits can then be enabled by writing to the address recognition registers and the Network Control register.

Note: The queue pointers must be initialized and point to USED descriptors for all queues including those not intended for use.

42.7.1.3 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 42-5 “Transmit Buffer Descriptor Entry”.

The Transmit Buffer Queue Pointer register points to this data structure.

To create this list of buffers:

1. Allocate a number (N) of buffers of between 1 and 2047 bytes of data to be transmitted in system memory. Up to 128 buffers per frame are allowed.
2. Allocate an area 8N bytes for the transmit buffer descriptor list in system memory and create N entries in this list. Mark all entries in this list as owned by GMAC, i.e., bit 31 of word 1 set to 0.
3. Mark the last descriptor in the queue with the wrap bit (bit 30 in word 1 set to 1).
4. Write address of transmit buffer descriptor list and control information to GMAC register transmit buffer queue pointer.
5. The transmit circuits can then be enabled by writing to the Network Control register.

Note: The queue pointers must be initialized and point to USED descriptors for all queues including those not intended for use.

42.7.1.4 Address Matching

The GMAC Hash register pair and the four Specific Address register pairs must be written with the required values. Each register pair comprises of a bottom register and top register, with the bottom register being written first. The address matching is disabled for a particular register pair after the bottom register has been written and re-enabled when the top register is written. Each register pair may be written at any time, regardless of whether the receive circuits are enabled or disabled.

As an example, to set Specific Address 1 register to recognize destination address 21:43:65:87:A9:CB, the following values are written to Specific Address 1 Bottom register and Specific Address 1 Top register:

- Specific Address 1 Bottom register bits 31:0 (0x98): 0x8765_4321.
- Specific Address 1 Top register bits 31:0 (0x9C): 0x0000_CBA9.

42.7.1.5 PHY Maintenance

The PHY Maintenance register is implemented as a shift register. Writing to the register starts a shift operation which is signalled as complete when bit two is set in the Network Status register (about 2000 MCK cycles later when bits 18:16 are set to 010 in the Network Configuration register). An interrupt is generated as this bit is set.

During this time, the MSB of the register is output on the MDIO pin and the LSB updated from the MDIO pin with each Management Data Clock (MDC) cycle. This causes the transmission of a PHY management frame on MDIO. See section 22.2.4.5 of the IEEE 802.3 standard.

Reading during the shift operation will return the current contents of the shift register. At the end of the management operation the bits will have shifted back to their original locations. For a read operation the data bits are updated with data read from the PHY. It is important to write the correct values to the register to ensure a valid PHY management frame is produced.

The Management Data Clock (MDC) should not toggle faster than 2.5 MHz (minimum period of 400 ns), as defined by the IEEE 802.3 standard. MDC is generated by dividing down MCK. Three bits in the Network Configuration register determine by how much MCK should be divided to produce MDC.

42.7.1.6 Interrupts

There are 18 interrupt conditions that are detected within the GMAC. The conditions are ORed to make a single interrupt. Depending on the overall system design this may be passed through a further level of interrupt collection (interrupt controller). On receipt of the interrupt signal, the CPU enters the interrupt handler. Refer to the device interrupt controller documentation to identify that it is the GMAC that is generating the interrupt. To ascertain which interrupt, read the Interrupt Status register. Note that in the default configuration this register will clear itself after being read, though this may be configured to be write-one-to-clear if desired.

At reset all interrupts are disabled. To enable an interrupt, write to Interrupt Enable register with the pertinent interrupt bit set to 1. To disable an interrupt, write to Interrupt Disable register with the pertinent interrupt bit set to 1. To check whether an interrupt is enabled or disabled, read Interrupt Mask register. If the bit is set to 1, the interrupt is disabled.

42.7.1.7 Transmitting Frames

The procedure to set up a frame for transmission is the following:

1. Enable transmit in the Network Control register.
2. Allocate an area of system memory for transmit data. This does not have to be contiguous, varying byte lengths can be used if they conclude on byte borders.
3. Set-up the transmit buffer list by writing buffer addresses to word zero of the transmit buffer descriptor entries and control and length to word one.
4. Write data for transmission into the buffers pointed to by the descriptors.
5. Write the address of the first buffer descriptor to transmit buffer descriptor queue pointer.
6. Enable appropriate interrupts.

7. Write to the transmit start bit (TSTART) in the Network Control register.

42.7.1.8 Receiving Frames

When a frame is received and the receive circuits are enabled, the GMAC checks the address and, in the following cases, the frame is written to system memory:

- If it matches one of the four Specific Address registers.
- If it matches one of the four Type ID registers.
- If it matches the hash address function.
- If it is a broadcast address (0xFFFFFFFFFFFF) and broadcasts are allowed.
- If the GMAC is configured to “copy all frames”.

The register receive buffer queue pointer points to the next entry in the receive buffer descriptor list and the GMAC uses this as the address in system memory to write the frame to.

Once the frame has been completely and successfully received and written to system memory, the GMAC then updates the receive buffer descriptor entry (see Table 42-4 “Receive Buffer Descriptor Entry”) with the reason for the address match and marks the area as being owned by software. Once this is complete, a receive complete interrupt is set. Software is then responsible for copying the data to the application area and releasing the buffer (by writing the ownership bit back to 0).

If the GMAC is unable to write the data at a rate to match the incoming frame, then a receive overrun interrupt is set. If there is no receive buffer available, i.e., the next buffer is still owned by software, a receive buffer not available interrupt is set.

42.8 Ethernet MAC (GMAC) User Interface

Table 42-17. Register Mapping

Offset[1] [2]	Register	Name	Access	Reset
0x000	Network Control Register	GMAC_NCR	Read/Write	0x0000_0000
0x004	Network Configuration Register	GMAC_NCFGR	Read/Write	0x0008_0000
0x008	Network Status Register	GMAC_NSR	Read-only	0b01x0
0x00C	User Register	GMAC_UR	Read/Write	0x0000_0000
0x010	DMA Configuration Register	GMAC_DCFGR	Read/Write	0x0002_0004
0x014	Transmit Status Register	GMAC_TSR	Read/Write	0x0000_0000
0x018	Receive Buffer Queue Base Address Register	GMAC_RBQB	Read/Write	0x0000_0000
0x01C	Transmit Buffer Queue Base Address Register	GMAC_TBQB	Read/Write	0x0000_0000
0x020	Receive Status Register	GMAC_RSR	Read/Write	0x0000_0000
0x024	Interrupt Status Register	GMAC_ISR	Read-only	0x0000_0000
0x028	Interrupt Enable Register	GMAC_IER	Write-only	–
0x02C	Interrupt Disable Register	GMAC_IDR	Write-only	–
0x030	Interrupt Mask Register	GMAC_IMR	Read/Write	0x07FF_FFFF
0x034	PHY Maintenance Register	GMAC_MAN	Read/Write	0x0000_0000
0x038	Received Pause Quantum Register	GMAC_RPQ	Read-only	0x0000_0000
0x03C	Transmit Pause Quantum Register	GMAC_TPQ	Read/Write	0x0000_FFFF
0x040–0x07C	Reserved	–	–	–
0x080	Hash Register Bottom	GMAC_HRB	Read/Write	0x0000_0000
0x084	Hash Register Top	GMAC_HRT	Read/Write	0x0000_0000
0x088	Specific Address 1 Bottom Register	GMAC_SAB1	Read/Write	0x0000_0000
0x08C	Specific Address 1 Top Register	GMAC_SAT1	Read/Write	0x0000_0000
0x090	Specific Address 2 Bottom Register	GMAC_SAB2	Read/Write	0x0000_0000
0x094	Specific Address 2 Top Register	GMAC_SAT2	Read/Write	0x0000_0000
0x098	Specific Address 3 Bottom Register	GMAC_SAB3	Read/Write	0x0000_0000
0x09C	Specific Address 3 Top Register	GMAC_SAT3	Read/Write	0x0000_0000
0x0A0	Specific Address 4 Bottom Register	GMAC_SAB4	Read/Write	0x0000_0000
0x0A4	Specific Address 4 Top Register	GMAC_SAT4	Read/Write	0x0000_0000
0x0A8	Type ID Match 1 Register	GMAC_TIDM1	Read/Write	0x0000_0000
0x0AC	Type ID Match 2 Register	GMAC_TIDM2	Read/Write	0x0000_0000
0x0B0	Type ID Match 3 Register	GMAC_TIDM3	Read/Write	0x0000_0000
0x0B4	Type ID Match 4 Register	GMAC_TIDM4	Read/Write	0x0000_0000
0x0B8	Reserved	–	–	–
0x0BC	IPG Stretch Register	GMAC_IPGS	Read/Write	0x0000_0000
0x0C0	Stacked VLAN Register	GMAC_SVLAN	Read/Write	0x0000_0000
0x0C4	Transmit PFC Pause Register	GMAC_TPFCP	Read/Write	0x0000_0000

Table 42-17. Register Mapping (Continued)

Offset[1] [2]	Register	Name	Access	Reset
0x0C8	Specific Address 1 Mask Bottom Register	GMAC_SAMB1	Read/Write	0x0000_0000
0x0CC	Specific Address 1 Mask Top Register	GMAC_SAMT1	Read/Write	0x0000_0000
0x0D0–0x0E4	Reserved	–	–	–
0x0E8–0x0FC	Reserved	–	–	–
0x100–0x1B0	Reserved	–	–	–
0x1B4–0x1CC	Reserved	–	–	–
0x1D0	1588 Timer Seconds Low Register	GMAC_TSL	Read/Write	0x0000_0000
0x1D4	1588 Timer Nanoseconds Register	GMAC_TN	Read/Write	0x0000_0000
0x1D8	1588 Timer Adjust Register	GMAC_TA	Write-only	–
0x1DC	1588 Timer Increment Register	GMAC_TI	Read/Write	0x0000_0000
0x1E0	PTP Event Frame Transmitted Seconds Low Register	GMAC_EFTSL	Read-only	0x0000_0000
0x1E4	PTP Event Frame Transmitted Nanoseconds Register	GMAC_EFTN	Read-only	0x0000_0000
0x1E8	PTP Event Frame Received Seconds Low Register	GMAC_EFRSL	Read-only	0x0000_0000
0x1EC	PTP Event Frame Received Nanoseconds Register	GMAC_EFRN	Read-only	0x0000_0000
0x1F0	PTP Peer Event Frame Transmitted Seconds Low Register	GMAC_PEFTSL	Read-only	0x0000_0000
0x1F4	PTP Peer Event Frame Transmitted Nanoseconds Register	GMAC_PEFTN	Read-only	0x0000_0000
0x1F8	PTP Peer Event Frame Received Seconds Low Register	GMAC_PEFRSL	Read-only	0x0000_0000
0x1FC	PTP Peer Event Frame Received Nanoseconds Register	GMAC_PEFRN	Read-only	0x0000_0000
0x200–0x7FC	Reserved	–	–	–

Notes: 1. If an offset is not listed in the Register Mapping, it must be considered as ‘reserved’.
2. Some register groups are not continuous in memory.

42.8.1 GMAC Network Control Register

Name: GMAC_NCR

Address: 0x40034000

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	FNP	TXPBPF	ENPBPR
15	14	13	12	11	10	9	8
SRTSM	–	–	TXZQPF	TXPF	THALT	TSTART	BP
7	6	5	4	3	2	1	0
–	–	–	MPE	TXEN	RXEN	LBL	–

- **LBL: Loop Back Local**

Connects GTX to GRX，GTXEN to GRXDV and forces full duplex mode. GRXCK and GTXCK may malfunction as the GMAC is switched into and out of internal loop back. It is important that receive and transmit circuits have already been disabled when making the switch into and out of internal loop back.

- **RXEN: Receive Enable**

When set, RXEN enables the GMAC to receive data. When reset frame reception stops immediately and the receive pipeline will be cleared. The Receive Queue Pointer Register is unaffected.

- **TXEN: Transmit Enable**

When set, TXEN enables the GMAC transmitter to send data. When reset transmission will stop immediately, the transmit pipeline and control registers will be cleared and the Transmit Queue Pointer Register will reset to point to the start of the transmit descriptor list.

- **MPE: Management Port Enable**

Set to one to enable the management port. When zero, forces MDIO to high impedance state and MDC low.

- **BP: Back pressure**

If set in 10M or 100M half duplex mode, forces collisions on all received frames.

- **TSTART: Start Transmission**

Writing one to this bit starts transmission.

- **THALT: Transmit Halt**

Writing one to this bit halts transmission as soon as any ongoing frame transmission ends.

- **TXPF: Transmit Pause Frame**

Writing one to this bit causes a pause frame to be transmitted.

- **TXZQPF: Transmit Zero Quantum Pause Frame**

Writing one to this bit causes a pause frame with zero quantum to be transmitted.

- **SRTSM: Store Receive Time Stamp to Memory**

0: Normal operation.

1: Causes the CRC of every received frame to be replaced with the value of the nanoseconds field of the 1588 timer that was captured as the receive frame passed the message time stamp point.

- **ENPBPR: Enable PFC Priority-based Pause Reception**

Enables PFC Priority Based Pause Reception capabilities. Setting this bit enables PFC negotiation and recognition of priority-based pause frames.

- **TXPBPF: Transmit PFC Priority-based Pause Frame**

Takes the values stored in the Transmit PFC Pause Register.

- **FNP: Flush Next Packet**

Flush the next packet from the external RX DPRAM. Writing one to this bit will only have an effect if the DMA is not currently writing a packet already stored in the DPRAM to memory.

42.8.2 GMAC Network Configuration Register

Name: GMAC_NCFGR

Address: 0x40034004

Access: Read/Write

31	30	29	28	27	26	25	24
–	IRXER	RXBP	IPGSEN	–	IRXFCS	EFRHD	RXCOEN
23	22	21	20	19	18	17	16
DCPF	DBW		CLK			RFCS	LFERD
15	14	13	12	11	10	9	8
RXBUFO		PEN	RTY	–	–	–	MAXFS
7	6	5	4	3	2	1	0
UNIHEN	MTI HEN	NBC	CAF	JFRAME	DNVLAN	FD	SPD

- **SPD: Speed**

Set to logic one to indicate 100 Mbps operation, logic zero for 10 Mbps.

- **FD: Full Duplex**

If set to logic one, the transmit block ignores the state of collision and carrier sense and allows receive while transmitting.

- **DNVLAN: Discard Non-VLAN FRAMES**

When set only VLAN tagged frames will be passed to the address matching logic.

- **JFRAME: Jumbo Frame Size**

Set to one to enable jumbo frames up to 10240 bytes to be accepted. The default length is 10240 bytes.

- **CAF: Copy All Frames**

When set to logic one, all valid frames will be accepted.

- **NBC: No Broadcast**

When set to logic one, frames addressed to the broadcast address of all ones will not be accepted.

- **MTIHEN: Multicast Hash Enable**

When set, multicast frames will be accepted when the 6-bit hash function of the destination address points to a bit that is set in the Hash Register.

- **UNIHEN: Unicast Hash Enable**

When set, unicast frames will be accepted when the 6-bit hash function of the destination address points to a bit that is set in the Hash Register.

- **MAXFS: 1536 Maximum Frame Size**

Setting this bit means the GMAC will accept frames up to 1536 bytes in length. Normally the GMAC would reject any frame above 1518 bytes.

- **RTY: Retry Test**

Must be set to zero for normal operation. If set to one the backoff between collisions will always be one slot time. Setting this bit to one helps test the too many retries condition. Also used in the pause frame tests to reduce the pause counter's decrement time from 512 bit times, to every GRXCK cycle.

- **PEN: Pause Enable**

When set, transmission will pause if a non-zero 802.3 classic pause frame is received and PFC has not been negotiated.

- **RXBUFO: Receive Buffer Offset**

Indicates the number of bytes by which the received data is offset from the start of the receive buffer

- **LFERD: Length Field Error Frame Discard**

Setting this bit causes frames with a measured length shorter than the extracted length field (as indicated by bytes 13 and 14 in a non-VLAN tagged frame) to be discarded. This only applies to frames with a length field less than 0x0600.

- **RFCS: Remove FCS**

Setting this bit will cause received frames to be written to memory without their frame check sequence (last 4 bytes). The frame length indicated will be reduced by four bytes in this mode.

- **CLK: MDC CLock Division**

Set according to MCK speed. These three bits determine the number MCK will be divided by to generate Management Data Clock (MDC). For conformance with the 802.3 specification, MDC must not exceed 2.5 MHz (MDC is only active during MDIO read and write operations).

Value	Name	Description
0	MCK_8	MCK divided by 8 (MCK up to 20 MHz)
1	MCK_16	MCK divided by 16 (MCK up to 40 MHz)
2	MCK_32	MCK divided by 32 (MCK up to 80 MHz)
3	MCK_48	MCK divided by 48 (MCK up to 120 MHz)
4	MCK_64	MCK divided by 64 (MCK up to 160 MHz)
5	MCK_96	MCK divided by 96 (MCK up to 240 MHz)

- **DBW: Data Bus Width**

Should always be written to ‘0’.

- **DCPF: Disable Copy of Pause Frames**

Set to one to prevent valid pause frames being copied to memory. When set, pause frames are not copied to memory regardless of the state of the Copy All Frames bit, whether a hash match is found or whether a type ID match is identified. If a destination address match is found, the pause frame will be copied to memory. Note that valid pause frames received will still increment pause statistics and pause the transmission of frames as required.

- **RXCOEN: Receive Checksum Offload Enable**

When set, the receive checksum engine is enabled. Frames with bad IP, TCP or UDP checksums are discarded.

- **EFRHD: Enable Frames Received in Half Duplex**

Enable frames to be received in half-duplex mode while transmitting.

- **IRXFCS: Ignore RX FCS**

When set, frames with FCS/CRC errors will not be rejected. FCS error statistics will still be collected for frames with bad FCS and FCS status will be recorded in frame’s DMA descriptor. For normal operation this bit must be set to zero.

- **IPGSEN: IP Stretch Enable**

When set, the transmit IPG can be increased above 96 bit times depending on the previous frame length using the IPG Stretch Register.

- **RXBP: Receive Bad Preamble**

When set, frames with non-standard preamble are not rejected.

- **IRXER: Ignore IPG GRXER**

When set, GRXER has no effect on the GMAC's operation when GRXDV is low.

42.8.3 GMAC Network Status Register

Name: GMAC_NSR

Address: 0x40034008

Access: Read-only

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

- **MDIO: MDIO Input Status**

Returns status of the MDIO pin.

- **IDLE: PHY Management Logic Idle**

The PHY management logic is idle (i.e., has completed).

42.8.4 GMAC User Register

Name: GMAC_UR

Address: 0x4003400C

Access: Read/Write

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

- **MII: MII Mode**

This bit must be set to 1.

Warning: The default value of this bit is 0.

42.8.5 GMAC DMA Configuration Register

Name: GMAC_DCFGR

Address: 0x40034010

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
DRBS							

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
ESPA	ESMA	–	FBLDO				

- **FBLDO: Fixed Burst Length for DMA Data Operations:**

Selects the burst length to attempt to use on the AHB when transferring frame data. Not used for DMA management operations and only used where space and data size allow. Otherwise SINGLE type AHB transfers are used.

Upper bits become non-writable if the configured DMA TX and RX FIFO sizes are smaller than required to support the selected burst size.

One-hot priority encoding enforced automatically on register writes as follows, where ‘x’ represents don’t care:

Value	Name	Description
0	–	Reserved
1	SINGLE	00001: Always use SINGLE AHB bursts
2	–	Reserved
4	INCR4	001xx: Attempt to use INCR4 AHB bursts (Default)
8	INCR8	01xxx: Attempt to use INCR8 AHB bursts
16	INCR16	1xxxx: Attempt to use INCR16 AHB bursts

- **ESMA: Endian Swap Mode Enable for Management Descriptor Accesses**

When set, selects swapped endianism for AHB transfers. When clear, selects little endian mode.

- **ESPA: Endian Swap Mode Enable for Packet Data Accesses**

When set, selects swapped endianism for AHB transfers. When clear, selects little endian mode.

- **DRBS: DMA Receive Buffer Size**

DMA receive buffer size in AHB system memory. The value defined by these bits determines the size of buffer to use in main AHB system memory when writing received data.

The value is defined in multiples of 64 bytes, thus a value of 0x01 corresponds to buffers of 64 bytes, 0x02 corresponds to 128 bytes etc. For example:

– 0x02: 128 bytes

– 0x18: 1536 bytes (1 × max length frame/buffer)

– 0xA0: 10240 bytes (1 × 10K jumbo frame/buffer)

Note that this value should never be written as zero.

42.8.6 GMAC Transmit Status Register

Name: GMAC_TSR

Address: 0x40034014

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	HRESP
7	6	5	4	3	2	1	0
–	UND	TXCOMP	TFC	TXGO	RLE	COL	UBR

- **UBR: Used Bit Read**

Set when a transmit buffer descriptor is read with its used bit set. Writing a one clears this bit.

- **COL: Collision Occurred**

Set by the assertion of collision. Writing a one clears this bit. When operating in 10/100 mode, this status indicates either a collision or a late collision.

- **RLE: Retry Limit Exceeded**

Writing a one clears this bit.

- **TXGO: Transmit Go**

Transmit go, if high transmit is active. When using the DMA interface this bit represents the TXGO variable as specified in the transmit buffer description.

- **TFC: Transmit Frame Corruption Due to AHB Error**

Transmit frame corruption due to AHB error. Set if an error occurs while midway through reading transmit frame from the AHB, including HRESP errors and buffers exhausted mid frame (if the buffers run out during transmission of a frame then transmission stops, FCS shall be bad and GTXER asserted).

Writing a one clears this bit.

- **TXCOMP: Transmit Complete**

Set when a frame has been transmitted. Writing a one clears this bit.

- **UND: Transmit Underrun**

This bit is set if the transmitter was forced to terminate a frame that it had already began transmitting due to further data being unavailable.

This bit is set if a transmitter status write back has not completed when another status write back is attempted.

When using the DMA interface configured for internal FIFO mode, this bit is also set when the transmit DMA has written the SOP data into the FIFO and either the AHB bus was not granted in time for further data, or because an AHB not OK response was returned, or because a used bit was read.

Writing a one clears this bit.

- **HRESP: HRESP Not OK**

Set when the DMA block sees HRESP not OK. Writing a one clears this bit.

42.8.7 GMAC Receive Buffer Queue Base Address Register

Name: GMAC_RBQB

Address: 0x40034018

Access: Read/Write

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR						–	–

This register holds the start address of the receive buffer queue (receive buffers descriptor list). The receive buffer queue base address must be initialized before receive is enabled through bit 2 of the Network Control Register. Once reception is enabled, any write to the Receive Buffer Queue Base Address Register is ignored. Reading this register returns the location of the descriptor currently being accessed. This value increments as buffers are used. Software should not use this register for determining where to remove received frames from the queue as it constantly changes as new frames are received. Software should instead work its way through the buffer descriptor queue checking the “used” bits.

In terms of AMBA AHB operation, the descriptors are read from memory using a single 32-bit AHB access. The descriptors should be aligned at 32-bit boundaries and the descriptors are written to using two individual non sequential accesses.

- **ADDR: Receive Buffer Queue Base Address**

Written with the address of the start of the receive queue.

42.8.8 GMAC Transmit Buffer Queue Base Address Register

Name: GMAC_TBQB

Address: 0x4003401C

Access: Read/Write

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR						–	–

This register holds the start address of the transmit buffer queue (transmit buffers descriptor list). The Transmit Buffer Queue Base Address Register must be initialized before transmit is started through bit 9 of the Network Control Register. Once transmission has started, any write to the Transmit Buffer Queue Base Address Register is illegal and therefore ignored.

Note that due to clock boundary synchronization, it takes a maximum of four MCK cycles from the writing of the transmit start bit before the transmitter is active. Writing to the Transmit Buffer Queue Base Address Register during this time may produce unpredictable results.

Reading this register returns the location of the descriptor currently being accessed. Since the DMA handles two frames at once, this may not necessarily be pointing to the current frame being transmitted.

In terms of AMBA AHB operation, the descriptors are written to memory using a single 32-bit AHB access. The descriptors should be aligned at 32-bit boundaries and the descriptors are read from memory using two individual non sequential accesses.

- **ADDR: Transmit Buffer Queue Base Address**

Written with the address of the start of the transmit queue.

42.8.9 GMAC Receive Status Register

Name: GMAC_RSR

Address: 0x40034020

Access: Read/Write

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

This register, when read, provides receive status details. Once read, individual bits may be cleared by writing a one to them. It is not possible to set a bit to 1 by writing to the register.

- **BNA: Buffer Not Available**

An attempt was made to get a new buffer and the pointer indicated that it was owned by the processor. The DMA will re-read the pointer each time an end of frame is received until a valid pointer is found. This bit is set following each descriptor read attempt that fails, even if consecutive pointers are unsuccessful and software has in the mean time cleared the status flag. Writing a one clears this bit.

- **REC: Frame Received**

One or more frames have been received and placed in memory. Writing a one clears this bit.

- **RXOVR: Receive Overrun**

This bit is set if RX FIFO is not able to store the receive frame due to a FIFO overflow, or if the receive status was not taken at the end of the frame. The buffer will be recovered if an overrun occurs. Writing a one clears this bit.

- **HNO: HRESP Not OK**

Set when the DMA block sees HRESP not OK. Writing a one clears this bit.

42.8.10 GMAC Interrupt Status Register

Name: GMAC_ISR

Address: 0x40034024

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	WOL	–	SRI	PDRSFT	PDRQFT
23	22	21	20	19	18	17	16
PDRSFR	PDRQFR	SFT	DRQFT	SFR	DRQFR	–	–
15	14	13	12	11	10	9	8
–	PFTR	PTZ	PFNZ	HRESP	ROVR	–	–
7	6	5	4	3	2	1	0
TCOMP	TFC	RLEX	TUR	TXUBR	RXUBR	RCOMP	MFS

This register indicates the source of the interrupt. In order that the bits of this register read 1, the corresponding interrupt source must be enabled in the mask register. If any bit is set in this register, the GMAC interrupt signal will be asserted in the system.

- **MFS: Management Frame Sent**

The PHY Maintenance Register has completed its operation. Cleared on read.

- **RCOMP: Receive Complete**

A frame has been stored in memory. Cleared on read.

- **RXUBR: RX Used Bit Read**

Set when a receive buffer descriptor is read with its used bit set. Cleared on read.

- **TXUBR: TX Used Bit Read**

Set when a transmit buffer descriptor is read with its used bit set. Cleared on read.

- **TUR: Transmit Underrun**

This interrupt is set if the transmitter was forced to terminate a frame that it has already began transmitting due to further data being unavailable.

This interrupt is set if a transmitter status write back has not completed when another status write back is attempted.

This interrupt is also set when the transmit DMA has written the SOP data into the FIFO and either the AHB bus was not granted in time for further data, or because an AHB not OK response was returned, or because the used bit was read.

- **RLEX: Retry Limit Exceeded**

Transmit error. Cleared on read.

- **TFC: Transmit Frame Corruption Due to AHB Error**

Transmit frame corruption due to AHB error. Set if an error occurs while midway through reading transmit frame from the AHB, including HRESP errors and buffers exhausted mid frame.

- **TCOMP: Transmit Complete**

Set when a frame has been transmitted. Cleared on read.

- **ROVR: Receive Overrun**

Set when the receive overrun status bit is set. Cleared on read.

- **HRESP: HRESP Not OK**

Set when the DMA block sees HRESP not OK. Cleared on read.

- **PFNZ: Pause Frame with Non-zero Pause Quantum Received**

Indicates a valid pause has been received that has a non-zero pause quantum field. Cleared on read.

- **PTZ: Pause Time Zero**

Set when either the Pause Time register at address 0x38 decrements to zero, or when a valid pause frame is received with a zero pause quantum field. Cleared on read.

- **PFTR: Pause Frame Transmitted**

Indicates a pause frame has been successfully transmitted after being initiated from the Network Control register. Cleared on read.

- **DRQFR: PTP Delay Request Frame Received**

Indicates a PTP delay_req frame has been received. Cleared on read.

- **SFR: PTP Sync Frame Received**

Indicates a PTP sync frame has been received. Cleared on read.

- **DRQFT: PTP Delay Request Frame Transmitted**

Indicates a PTP delay_req frame has been transmitted. Cleared on read.

- **SFT: PTP Sync Frame Transmitted**

Indicates a PTP sync frame has been transmitted. Cleared on read.

- **PDRQFR: PDelay Request Frame Received**

Indicates a PTP pdelay_req frame has been received. Cleared on read.

- **PDRSFR: PDelay Response Frame Received**

Indicates a PTP pdelay_resp frame has been received. Cleared on read.

- **PDRQFT: PDelay Request Frame Transmitted**

Indicates a PTP pdelay_req frame has been transmitted. Cleared on read.

- **PDRSFT: PDelay Response Frame Transmitted**

Indicates a PTP pdelay_resp frame has been transmitted. Cleared on read.

- **SRI: TSU Seconds Register Increment**

Indicates the register has incremented. Cleared on read.

- **WOL: Wake On LAN**

WOL interrupt. Indicates a WOL event has been received.

42.8.11 GMAC Interrupt Enable Register

Name: GMAC_IER

Address: 0x40034028

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	WOL	–	SRI	PDRSFT	PDRQFT
23	22	21	20	19	18	17	16
PDRSFR	PDRQFR	SFT	DRQFT	SFR	DRQFR	–	–
15	14	13	12	11	10	9	8
EXINT	PFTR	PTZ	PFNZ	HRESP	ROVR	–	–
7	6	5	4	3	2	1	0
TCOMP	TFC	RLEX	TUR	TXUBR	RXUBR	RCOMP	MFS

This register is write-only and when read will return zero.

The following values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **MFS: Management Frame Sent**
- **RCOMP: Receive Complete**
- **RXUBR: RX Used Bit Read**
- **TXUBR: TX Used Bit Read**
- **TUR: Transmit Underrun**
- **RLEX: Retry Limit Exceeded or Late Collision**
- **TFC: Transmit Frame Corruption Due to AHB Error**
- **TCOMP: Transmit Complete**
- **ROVR: Receive Overrun**
- **HRESP: HRESP Not OK**
- **PFNZ: Pause Frame with Non-zero Pause Quantum Received**
- **PTZ: Pause Time Zero**
- **PFTR: Pause Frame Transmitted**
- **EXINT: External Interrupt**
- **DRQFR: PTP Delay Request Frame Received**

- **SFR: PTP Sync Frame Received**
- **DRQFT: PTP Delay Request Frame Transmitted**
- **SFT: PTP Sync Frame Transmitted**
- **PDRQFR: PDelay Request Frame Received**
- **PDRSFR: PDelay Response Frame Received**
- **PDRQFT: PDelay Request Frame Transmitted**
- **PDRSFT: PDelay Response Frame Transmitted**
- **SRI: TSU Seconds Register Increment**
- **WOL: Wake On LAN**

42.8.12 GMAC Interrupt Disable Register

Name: GMAC_IDR

Address: 0x4003402C

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	WOL	–	SRI	PDRSFT	PDRQFT
23	22	21	20	19	18	17	16
PDRSFR	PDRQFR	SFT	DRQFT	SFR	DRQFR	–	–
15	14	13	12	11	10	9	8
EXINT	PFTR	PTZ	PFNZ	HRESP	ROVR	–	–
7	6	5	4	3	2	1	0
TCOMP	TFC	RLEX	TUR	TXUBR	RXUBR	RCOMP	MFS

This register is write-only and when read will return zero.

The following values are valid for all listed bit names of this register:

0: No effect.

1: Disables the corresponding interrupt.

- **MFS: Management Frame Sent**
- **RCOMP: Receive Complete**
- **RXUBR: RX Used Bit Read**
- **TXUBR: TX Used Bit Read**
- **TUR: Transmit Underrun**
- **RLEX: Retry Limit Exceeded or Late Collision**
- **TFC: Transmit Frame Corruption Due to AHB Error**
- **TCOMP: Transmit Complete**
- **ROVR: Receive Overrun**
- **HRESP: HRESP Not OK**
- **PFNZ: Pause Frame with Non-zero Pause Quantum Received**
- **PTZ: Pause Time Zero**
- **PFTR: Pause Frame Transmitted**
- **EXINT: External Interrupt**
- **DRQFR: PTP Delay Request Frame Received**

- **SFR: PTP Sync Frame Received**
- **DRQFT: PTP Delay Request Frame Transmitted**
- **SFT: PTP Sync Frame Transmitted**
- **PDRQFR: PDelay Request Frame Received**
- **PDRSFR: PDelay Response Frame Received**
- **PDRQFT: PDelay Request Frame Transmitted**
- **PDRSFT: PDelay Response Frame Transmitted**
- **SRI: TSU Seconds Register Increment**
- **WOL: Wake On LAN**

42.8.13 GMAC Interrupt Mask Register

Name: GMAC_IMR

Address: 0x40034030

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	PDRSFT	PDRQFT
23	22	21	20	19	18	17	16
PDRSFR	PDRQFR	SFT	DRQFT	SFR	DRQFR	–	–
15	14	13	12	11	10	9	8
EXINT	PFTR	PTZ	PFNZ	HRESP	ROVR	–	–
7	6	5	4	3	2	1	0
TCOMP	TFC	RLEX	TUR	TXUBR	RXUBR	RCOMP	MFS

The Interrupt Mask Register is a read-only register indicating which interrupts are masked. All bits are set at reset and can be reset individually by writing to the Interrupt Enable Register or set individually by writing to the Interrupt Disable Register. Having separate address locations for enable and disable saves the need for performing a read modify write when updating the Interrupt Mask Register.

For test purposes there is a write-only function to this register that allows the bits in the Interrupt Status Register to be set or cleared, regardless of the state of the mask register. A write to this register directly affects the state of the corresponding bit in the Interrupt Status Register, causing an interrupt to be generated if a 1 is written.

The following values are valid for all listed bit names of this register when read:

0: The corresponding interrupt is enabled.

1: The corresponding interrupt is not enabled.

- **MFS: Management Frame Sent**
- **RCOMP: Receive Complete**
- **RXUBR: RX Used Bit Read**
- **TXUBR: TX Used Bit Read**
- **TUR: Transmit Underrun**
- **RLEX: Retry Limit Exceeded**
- **TFC: Transmit Frame Corruption Due to AHB Error**
- **TCOMP: Transmit Complete**
- **ROVR: Receive Overrun**
- **HRESP: HRESP Not OK**
- **PFNZ: Pause Frame with Non-zero Pause Quantum Received**
- **PTZ: Pause Time Zero**

- **PFTR: Pause Frame Transmitted**
- **EXINT: External Interrupt**
- **DRQFR: PTP Delay Request Frame Received**
- **SFR: PTP Sync Frame Received**
- **DRQFT: PTP Delay Request Frame Transmitted**
- **SFT: PTP Sync Frame Transmitted**
- **PDRQFR: PDelay Request Frame Received**
- **PDRSFR: PDelay Response Frame Received**
- **PDRQFT: PDelay Request Frame Transmitted**
- **PDRSFT: PDelay Response Frame Transmitted**

42.8.14 GMAC PHY Maintenance Register

Name: GMAC_MAN

Address: 0x40034034

Access: Read/Write

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

The PHY Maintenance Register is implemented as a shift register. Writing to the register starts a shift operation which is signalled as complete when bit 2 is set in the Network Status Register. It takes about 2000 MCK cycles to complete, when MDC is set for MCK divide by 32 in the Network Configuration Register. An interrupt is generated upon completion.

During this time, the MSB of the register is output on the MDIO pin and the LSB updated from the MDIO pin with each MDC cycle. This causes transmission of a PHY management frame on MDIO. *See Section 22.2.4.5 of the IEEE 802.3 standard*.

Reading during the shift operation returns the current contents of the shift register. At the end of management operation, the bits will have shifted back to their original locations. For a read operation, the data bits 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, bit 30 should be written with a 0 rather than a 1. To write clause 45 PHYs, bits 31:28 should be written as 0x0001. See Table 42-18.

Table 42-18. Clause 22/Clause 45 PHYs Read/Write Access Configuration (GMAC_MAN Bits 31:28)

PHY	Access	Bit Value			
		WZO	CLTTO	OP[1]	OP[0]
Clause 22	Read	0	1	1	0
	Write	0	1	0	1
Clause 45	Read	0	0	1	1
	Write	0	0	0	1
	Read + Address	0	0	1	0

For a description of MDC generation, see Section 42.8.2 "GMAC Network Configuration Register".

- **DATA: PHY Data**

For a write operation this field is written with the data to be written to the PHY. After a read operation this field contains the data read from the PHY.

- **WTN: Write Ten**

Must be written to 10.

- **REGA: Register Address**

Specifies the register in the PHY to access.

- **PHYA: PHY Address**
- **OP: Operation**

01: Write

10: Read

- **CLTTO: Clause 22 Operation**

0: Clause 45 operation

1: Clause 22 operation

- **WZO: Write ZERO**

Must be written with 0.

42.8.15 GMAC Receive Pause Quantum Register

Name: GMAC_RPQ

Address: 0x40034038

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
RPQ							

7	6	5	4	3	2	1	0
RPQ							

- **RPQ: Received Pause Quantum**

Stores the current value of the Receive Pause Quantum Register which is decremented every 512 bit times.

42.8.16 GMAC Transmit Pause Quantum Register

Name: GMAC_TPQ

Address: 0x4003403C

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
TPQ							

7	6	5	4	3	2	1	0
TPQ							

- **TPQ: Transmit Pause Quantum**

Written with the pause quantum value for pause frame transmission.

42.8.17 GMAC Hash Register Bottom

Name: GMAC_HRB

Address: 0x40034080

Access: Read-only

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR							

The unicast hash enable (UNIHEN) and the multicast hash enable (MITIHEN) bits in the Network Configuration Register (Section 42.8.2 ”GMAC Network Configuration Register”) enable the reception of hash matched frames. See Section 42.6.9 ”Hash Addressing”.

- **ADDR: Hash Address**

The first 32 bits of the Hash Address Register.

42.8.18 GMAC Hash Register Top

Name: GMAC_HRT

Address: 0x40034084

Access: Read-only

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR							

The unicast hash enable (UNIHEN) and the multicast hash enable (MITIHEN) bits in the GMAC Network Configuration Register enable the reception of hash matched frames. See Section 42.6.9 ”Hash Addressing”.

- **ADDR: Hash Address**

Bits 63 to 32 of the Hash Address Register.

42.8.19 GMAC Specific Address 1 Bottom Register

Name: GMAC_SAB1

Address: 0x40034088

Access: Read/Write

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR							

The addresses stored in the Specific Address Registers are deactivated at reset or when their corresponding Specific Address Register Bottom is written. They are activated when Specific Address Register Top is written.

- **ADDR: Specific Address 1**

Least significant 32 bits of the destination address, that is, bits 31:0. Bit zero indicates whether the address is multicast or unicast and corresponds to the least significant bit of the first byte received.

42.8.20 GMAC Specific Address 1 Top Register

Name: GMAC_SAT1

Address: 0x4003408C

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR							

The addresses stored in the Specific Address Registers are deactivated at reset or when their corresponding Specific Address Register Bottom is written. They are activated when Specific Address Register Top is written.

- **ADDR: Specific Address 1**

The most significant bits of the destination address, that is, bits 47:32.

42.8.21 GMAC Specific Address 2 Bottom Register

Name: GMAC_SAB2

Address: 0x40034090

Access: Read/Write

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR							

The addresses stored in the Specific Address Registers are deactivated at reset or when their corresponding Specific Address Register Bottom is written. They are activated when Specific Address Register Top is written.

- **ADDR: Specific Address 2**

Least significant 32 bits of the destination address, that is, bits 31:0. Bit zero indicates whether the address is multicast or unicast and corresponds to the least significant bit of the first byte received.

42.8.22 GMAC Specific Address 2 Top Register

Name: GMAC_SAT2

Address: 0x40034094

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR							

The addresses stored in the Specific Address Registers are deactivated at reset or when their corresponding Specific Address Register Bottom is written. They are activated when Specific Address Register Top is written.

- **ADDR: Specific Address 2**

The most significant bits of the destination address, that is, bits 47:32.

42.8.23 GMAC Specific Address 3 Bottom Register

Name: GMAC_SAB3

Address: 0x40034098

Access: Read/Write

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR							

The addresses stored in the Specific Address Registers are deactivated at reset or when their corresponding Specific Address Register Bottom is written. They are activated when Specific Address Register Top is written.

- **ADDR: Specific Address 3**

Least significant 32 bits of the destination address, that is, bits 31:0. Bit zero indicates whether the address is multicast or unicast and corresponds to the least significant bit of the first byte received.

42.8.24 GMAC Specific Address 3 Top Register

Name: GMAC_SAT3

Address: 0x4003409C

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR							

The addresses stored in the Specific Address Registers are deactivated at reset or when their corresponding Specific Address Register Bottom is written. They are activated when Specific Address Register Top is written.

- **ADDR: Specific Address 3**

The most significant bits of the destination address, that is, bits 47:32.

42.8.25 GMAC Specific Address 4 Bottom Register

Name: GMAC_SAB4

Address: 0x400340A0

Access: Read/Write

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR							

The addresses stored in the Specific Address Registers are deactivated at reset or when their corresponding Specific Address Register Bottom is written. They are activated when Specific Address Register Top is written.

- **ADDR: Specific Address 4**

Least significant 32 bits of the destination address, that is, bits 31:0. Bit zero indicates whether the address is multicast or unicast and corresponds to the least significant bit of the first byte received.

42.8.26 GMAC Specific Address 4 Top Register

Name: GMAC_SAT4

Address: 0x400340A4

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR							

The addresses stored in the Specific Address Registers are deactivated at reset or when their corresponding Specific Address Register Bottom is written. They are activated when Specific Address Register Top is written.

- **ADDR: Specific Address 4**

The most significant bits of the destination address, that is, bits 47:32.

42.8.27 GMAC Type ID Match 1 Register

Name: GMAC_TIDM1

Address: 0x400340A8

Access: Read/Write

31	30	29	28	27	26	25	24
ENID1	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
TID							

7	6	5	4	3	2	1	0
TID							

- **TID: Type ID Match 1**

For use in comparisons with received frames type ID/length frames.

- **ENID1: Enable Copying of TID Matched Frames**

0: TID is not part of the comparison match.

1: TID is processed for the comparison match.

42.8.28 GMAC Type ID Match 2 Register

Name: GMAC_TIDM2

Address: 0x400340AC

Access: Read/Write

31	30	29	28	27	26	25	24
ENID2	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
TID							

7	6	5	4	3	2	1	0
TID							

- **TID: Type ID Match 2**

For use in comparisons with received frames type ID/length frames.

- **ENID2: Enable Copying of TID Matched Frames**

0: TID is not part of the comparison match.

1: TID is processed for the comparison match.

42.8.29 GMAC Type ID Match 3 Register

Name: GMAC_TIDM3

Address: 0x400340B0

Access: Read/Write

31	30	29	28	27	26	25	24
ENID3	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
TID							

7	6	5	4	3	2	1	0
TID							

- **TID: Type ID Match 3**

For use in comparisons with received frames type ID/length frames.

- **ENID3: Enable Copying of TID Matched Frames**

0: TID is not part of the comparison match.

1: TID is processed for the comparison match.

42.8.30 GMAC Type ID Match 4 Register

Name: GMAC_TIDM4

Address: 0x400340B4

Access: Read/Write

31	30	29	28	27	26	25	24
ENID4	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
TID							

7	6	5	4	3	2	1	0
TID							

- **TID: Type ID Match 4**

For use in comparisons with received frames type ID/length frames.

- **ENID4: Enable Copying of TID Matched Frames**

0: TID is not part of the comparison match.

1: TID is processed for the comparison match.

42.8.31 GMAC IPG Stretch Register

Name: GMAC_IPGS

Address: 0x400340BC

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
FL							

7	6	5	4	3	2	1	0
FL							

- **FL: Frame Length**

Bits 7:0 are multiplied with the previously transmitted frame length (including preamble). Bits 15:8 +1 divide the frame length. If the resulting number is greater than 96 and bit 28 is set in the Network Configuration Register then the resulting number is used for the transmit inter-packet-gap. 1 is added to bits 15:8 to prevent a divide by zero. See Section 42.6.4 ”MAC Transmit Block”.

42.8.32 GMAC Stacked VLAN Register

Name: GMAC_SVLAN

Address: 0x400340C0

Access: Read/Write

31	30	29	28	27	26	25	24
ESVLAN	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
VLAN_TYPE							

7	6	5	4	3	2	1	0
VLAN_TYPE							

- **VLAN_TYPE: User Defined VLAN_TYPE Field**

User defined VLAN_TYPE field. When Stacked VLAN is enabled, the first VLAN tag in a received frame will only be accepted if the VLAN type field is equal to this user defined VLAN_TYPE, OR equal to the standard VLAN type (0x8100). Note that the second VLAN tag of a Stacked VLAN packet will only be matched correctly if its VLAN_TYPE field equals 0x8100.

- **ESVLAN: Enable Stacked VLAN Processing Mode**

0: Disable the stacked VLAN processing mode

1: Enable the stacked VLAN processing mode

42.8.33 GMAC Transmit PFC Pause Register

Name: GMAC_TPFCP

Address: 0x400340C4

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
PQ							

7	6	5	4	3	2	1	0
PEV							

- **PEV: Priority Enable Vector**

If bit 17 of the Network Control Register is written with a one then the priority enable vector of the PFC priority based pause frame will be set equal to the value stored in this register [7:0].

- **PQ: Pause Quantum**

If bit 17 of the Network Control Register is written with a one then for each entry equal to zero in the Transmit PFC Pause Register[15:8], the PFC pause frame's pause quantum field associated with that entry will be taken from the Transmit Pause Quantum Register. For each entry equal to one in the Transmit PFC Pause Register [15:8], the pause quantum associated with that entry will be zero.

42.8.34 GMAC Specific Address 1 Mask Bottom Register

Name: GMAC_SAMB1

Address: 0x400340C8

Access: Read/Write

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR							

- **ADDR: Specific Address 1 Mask**

Setting a bit to one masks the corresponding bit in the Specific Address 1 Register.

42.8.35 GMAC Specific Address Mask 1 Top Register

Name: GMAC_SAMT1

Address: 0x400340CC

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR							

- **ADDR: Specific Address 1 Mask**

Setting a bit to one masks the corresponding bit in the Specific Address 1 Register.

42.8.36 GMAC 1588 Timer Seconds Low Register

Name: GMAC_TSL

Address: 0x400341D0

Access: Read/Write

31	30	29	28	27	26	25	24
TCS							

23	22	21	20	19	18	17	16
TCS							

15	14	13	12	11	10	9	8
TCS							

7	6	5	4	3	2	1	0
TCS							

- **TCS: Timer Count in Seconds**

This register is writable. It increments by one when the 1588 nanoseconds counter counts to one second. It may also be incremented when the Timer Adjust Register is written.

42.8.37 GMAC 1588 Timer Nanoseconds Register

Name: GMAC_TN

Address: 0x400341D4

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	TNS					

23	22	21	20	19	18	17	16
TNS							

15	14	13	12	11	10	9	8
TNS							

7	6	5	4	3	2	1	0
TNS							

- **TNS: Timer Count in Nanoseconds**

This register is writable. It can also be adjusted by writes to the 1588 Timer Adjust Register. It increments by the value of the 1588 Timer Increment Register each clock cycle.

42.8.38 GMAC 1588 Timer Adjust Register

Name: GMAC_TA

Address: 0x400341D8

Access: Write-only

31	30	29	28	27	26	25	24
ADJ	–	ITDT					

23	22	21	20	19	18	17	16
ITDT							

15	14	13	12	11	10	9	8
ITDT							

7	6	5	4	3	2	1	0
ITDT							

- **ITDT: Increment/Decrement**

The number of nanoseconds to increment or decrement the 1588 Timer Nanoseconds Register. If necessary, the 1588 Seconds Register will be incremented or decremented.

- **ADJ: Adjust 1588 Timer**

Write as one to subtract from the 1588 timer. Write as zero to add to it.

42.8.39 GMAC 1588 Timer Increment Register

Name: GMAC_TI

Address: 0x400341DC

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
NIT							

15	14	13	12	11	10	9	8
ACNS							

7	6	5	4	3	2	1	0
CNS							

- **CNS: Count Nanoseconds**

A count of nanoseconds by which the 1588 Timer Nanoseconds Register will be incremented each clock cycle.

- **ACNS: Alternative Count Nanoseconds**

Alternative count of nanoseconds by which the 1588 Timer Nanoseconds Register will be incremented each clock cycle.

- **NIT: Number of Increments**

The number of increments after which the alternative increment is used.

42.8.40 GMAC PTP Event Frame Transmitted Seconds Low Register

Name: GMAC_EFTSL

Address: 0x400341E0

Access: Read-only

31	30	29	28	27	26	25	24
RUD							

23	22	21	20	19	18	17	16
RUD							

15	14	13	12	11	10	9	8
RUD							

7	6	5	4	3	2	1	0
RUD							

- **RUD: Register Update**

The register is updated with the value that the 1588 Timer Seconds Register holds when the SFD of a PTP transmit primary event crosses the MII interface. An interrupt is issued when the register is updated.

42.8.41 GMAC PTP Event Frame Transmitted Nanoseconds Register

Name: GMAC_EFTN

Address: 0x400341E4

Access: Read-only

31	30	29	28	27	26	25	24
–	–	RUD					

23	22	21	20	19	18	17	16
RUD							

15	14	13	12	11	10	9	8
RUD							

7	6	5	4	3	2	1	0
RUD							

- **RUD: Register Update**

The register is updated with the value that the 1588 Timer Nanoseconds Register holds when the SFD of a PTP transmit primary event crosses the MII interface. An interrupt is issued when the register is updated.

42.8.42 GMAC PTP Event Frame Received Seconds Low Register

Name: GMAC_EFRSL

Address: 0x400341E8

Access: Read-only

31	30	29	28	27	26	25	24
RUD							

23	22	21	20	19	18	17	16
RUD							

15	14	13	12	11	10	9	8
RUD							

7	6	5	4	3	2	1	0
RUD							

- **RUD: Register Update**

The register is updated with the value that the 1588 Timer Seconds Register holds when the SFD of a PTP receive primary event crosses the MII interface. An interrupt is issued when the register is updated.

42.8.43 GMAC PTP Event Frame Received Nanoseconds Register

Name: GMAC_EFRN

Address: 0x400341EC

Access: Read-only

31	30	29	28	27	26	25	24
–	–	RUD					

23	22	21	20	19	18	17	16
RUD							

15	14	13	12	11	10	9	8
RUD							

7	6	5	4	3	2	1	0
RUD							

- **RUD: Register Update**

The register is updated with the value that the 1588 Timer Nanoseconds Register holds when the SFD of a PTP receive primary event crosses the MII interface. An interrupt is issued when the register is updated.

42.8.44 GMAC PTP Peer Event Frame Transmitted Seconds Low Register

Name: GMAC_PEFTSL

Address: 0x400341F0

Access: Read-only

31	30	29	28	27	26	25	24
RUD							

23	22	21	20	19	18	17	16
RUD							

15	14	13	12	11	10	9	8
RUD							

7	6	5	4	3	2	1	0
RUD							

- **RUD: Register Update**

The register is updated with the value that the 1588 Timer Seconds Register holds when the SFD of a PTP transmit peer event crosses the MII interface. An interrupt is issued when the register is updated.

42.8.45 GMAC PTP Peer Event Frame Transmitted Nanoseconds Register

Name: GMAC_PEFTN

Address: 0x400341F4

Access: Read-only

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

- **RUD: Register Update**

The register is updated with the value that the 1588 Timer Nanoseconds Register holds when the SFD of a PTP transmit peer event crosses the MII interface. An interrupt is issued when the register is updated.

42.8.46 GMAC PTP Peer Event Frame Received Seconds Low Register

Name: GMAC_PEFRSL

Address: 0x400341F8

Access: Read-only

31	30	29	28	27	26	25	24
RUD							

23	22	21	20	19	18	17	16
RUD							

15	14	13	12	11	10	9	8
RUD							

7	6	5	4	3	2	1	0
RUD							

- **RUD: Register Update**

The register is updated with the value that the 1588 Timer Seconds Register holds when the SFD of a PTP receive primary event crosses the MII interface. An interrupt is issued when the register is updated.

42.8.47 GMAC PTP Peer Event Frame Received Nanoseconds Register

Name: GMAC_PEFRN

Address: 0x400341FC

Access: Read-only

31	30	29	28	27	26	25	24
–	–	RUD					

23	22	21	20	19	18	17	16
RUD							

15	14	13	12	11	10	9	8
RUD							

7	6	5	4	3	2	1	0
RUD							

- **RUD: Register Update**

The register is updated with the value that the 1588 Timer Nanoseconds Register holds when the SFD of a PTP receive primary event crosses the MII interface. An interrupt is issued when the register is updated.

43. Analog Front-End Controller (AFEC)

43.1 Description

The Analog Front-End Controller (AFEC) is based on an Analog Front-End cell (AFE) integrating a 12-bit Analog-to-Digital Converter (ADC), a Programmable Gain Amplifier (PGA), a Digital-to-Analog Converter (DAC) and a 16-to-1 analog multiplexer, making possible the analog-to-digital conversions of 16 analog lines. The conversions extend from 0V to ADVREF. The AFEC supports an 10-bit or 12-bit resolution mode which can be extended up to a 16-bit resolution by digital averaging.

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 AFE_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 are fully configurable.

The AFEC internal fault output is directly connected to PWM Fault input. This input can be asserted by means of comparison circuitry in order to immediately put the PWM outputs in a safe state (pure combinational path).

The AFEC also integrates a Sleep mode and a conversion sequencer and connects with a PDC channel. These features reduce both power consumption and processor intervention.

The AFEC has a selectable single-ended or fully differential input and benefits from a 2-bit programmable gain. A 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.

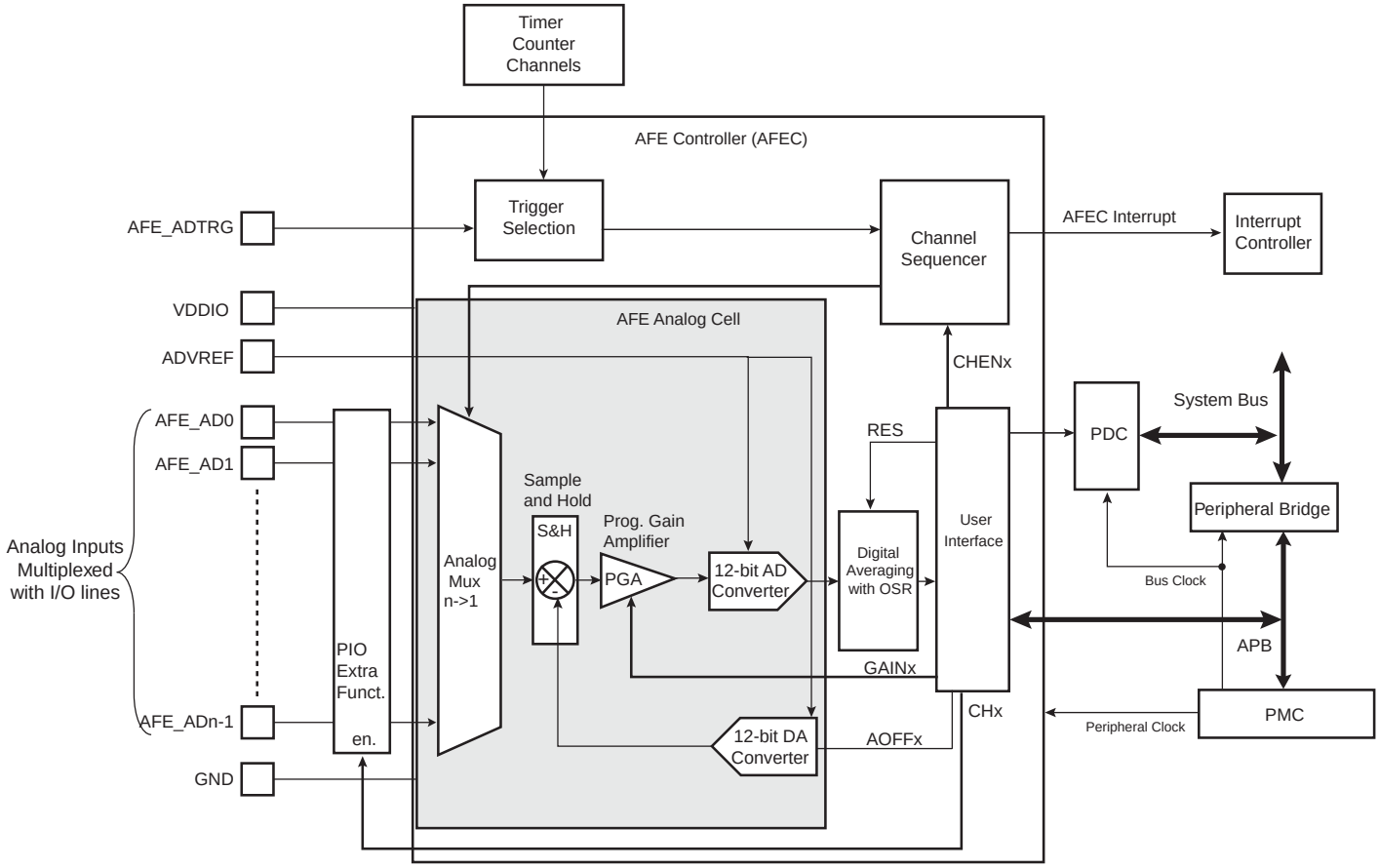
Finally, the user can configure AFE timings, such as startup time and tracking time.

43.2 Embedded Characteristics

- 12-bit Resolution up to 16-bit Resolution by Digital Averaging
- Wide Range of Power Supply Operation
- Selectable Single-ended or Differential Input Voltage
- Programmable Gain for Maximum Full-Scale Input Range 0–V_{DD}
- Programmable Offset Per Channel
- Integrated Multiplexer Offering Up to 16 Independent Analog Inputs
- Individual Enable and Disable of Each Channel
- Hardware or Software Trigger
  - External trigger pin
  - Timer counter outputs (corresponding TIOA trigger)
  - PWM event line
- Drive of PWM Fault Input
- PDC Support
- Possibility of AFE Timings Configuration
- Two Sleep Modes and Conversion Sequencer
  - Automatic wake-up on trigger and back to sleep mode after conversions of all enabled channels
  - Possibility of customized channel sequence
- Standby Mode for Fast Wake-up Time Response
  - Power-down capability
- Automatic Window Comparison of Converted Values
- Register Write Protection

43.3 Block Diagram

Figure 43-1. Analog Front-End Controller Block Diagram



43.4 Signal Description

Table 43-1. AFEC Signal Description

Pin Name	Description
ADVREF	Reference voltage
AFE_AD0—AFE_AD15[1]	Analog input channels
AFE_ADTRG	External trigger

Note: 1. AFE_AD15 is not an actual pin but is connected to a temperature sensor.

43.5 Product Dependencies

43.5.1 I/O Lines

The digital input AFE_ADTRG is multiplexed with digital functions on the I/O line and the selection of AFE_ADTRG is made using the PIO Controller.

The analog inputs AFE_ADx are multiplexed with digital functions on the I/O lines. AFE_ADx inputs are selected as inputs of the AFEC when writing a one in the corresponding CHx bit of AFEC_CHER and the digital functions are not selected.

Table 43-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
AFEC0	AFE0_ADTRG	PA8	B
AFEC0	AFE0_AD0	PA17	X1
AFEC0	AFE0_AD1	PA18	X1
AFEC0	AFE0_AD2/WKUP9	PA19	X1
AFEC0	AFE0_AD3/WKUP10	PA20	X1
AFEC0	AFE0_AD4/RTCOUT0	PB0	X1
AFEC0	AFE0_AD5/RTCOUT1	PB1	X1
AFEC0	AFE0_AD6	PC13	X1
AFEC0	AFE0_AD7	PC15	X1
AFEC0	AFE0_AD8	PC12	X1
AFEC0	AFE0_AD9	PC29	X1
AFEC0	AFE0_AD10	PC30	X1
AFEC0	AFE0_AD11	PC31	X1
AFEC0	AFE0_AD12	PC26	X1
AFEC0	AFE0_AD13	PC27	X1
AFEC0	AFE0_AD14	PC0	X1
AFEC1	AFE1_AD0/WKUP12	PB2	X1
AFEC1	AFE1_AD1	PB3	X1
AFEC1	AFE1_AD2	PA21	X1
AFEC1	AFE1_AD3	PA22	X1
AFEC1	AFE1_AD4	PC1	X1
AFEC1	AFE1_AD5	PC2	X1
AFEC1	AFE1_AD6	PC3	X1
AFEC1	AFE1_AD7	PC4	X1

43.5.2 Power Management

The AFEC is not continuously clocked. The programmer must first enable the AFEC peripheral clock in the Power Management Controller (PMC) before using the AFEC. However, if the application does not require AFEC operations, the peripheral clock can be stopped when not needed and restarted when necessary.

When the AFEC is in Sleep mode, the peripheral clock must always be enabled.

43.5.3 Interrupt Sources

The AFEC interrupt line is connected on one of the internal sources of the Interrupt Controller. Using the AFEC interrupt requires the interrupt controller to be programmed first.

Table 43-3. Peripheral IDs

Instance	ID
AFEC0	30
AFEC1	31

43.5.4 Temperature Sensor

The temperature sensor is connected to Channel 15 of the AFEC.

The temperature sensor provides an output voltage V_T that is proportional to the absolute temperature (PTAT).

43.5.5 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.

43.5.6 PWM Event Line

PWM event lines may or may not be used as hardware triggers depending on user requirements.

43.5.7 Fault Output

The AFEC has the Fault output connected to the FAULT input of PWM. Refer to Section 43.6.15 “Fault Output” and implementation of the PWM in the product.

43.5.8 Conversion Performances

For performance and electrical characteristics of the AFE, refer to the AFE Characteristics in the section “Electrical Characteristics”.

43.6 Functional Description

43.6.1 Analog Front-End Conversion

The AFE uses the AFE clock to perform conversions. In order to guarantee a conversion with minimum error, after any start of conversion, the AFEC waits a number of AFE clock cycles (called transfer time) before changing the channel selection again (and so starts a new tracking operation).

AFE conversions are sequenced by two operating times: the tracking time and the conversion time.

- The tracking time represents the time between the channel selection change and the time for the controller to start the AFEC. The AFEC allows a minimum tracking time of 15 AFE clock periods.
- The conversion time represents the time for the AFEC to convert the analog signal.

The AFE clock frequency is selected in the PRESCAL field of the AFEC_MR. The tracking phase starts during the conversion of the previous channel. If the tracking time is longer than the conversion time of the12-bit AD converter (t_{CONV}), the tracking phase is extended to the end of the previous conversion.

The AFE clock frequency ranges from $f_{peripheral\ clock}/2$ if PRESCAL is 0, and $f_{peripheral\ clock}/512$ if PRESCAL is set to 255 (0xFF). PRESCAL must be programmed to provide the AFE clock frequency given in the section “Electrical Characteristics”.

The AFE conversion time (t_{AFE_conv}) is applicable for all modes and is calculated as follows:

$$t_{AFE\_conv} = 21 \times t_{AFE\ Clock}$$

When the averager is activated, the AFE conversion time is multiplied by the OSR value.

In Free Run mode, the sampling frequency (f_S) is calculated as $1/t_{AFE\_conv}$.

Figure 43-2. Sequence of AFE Conversions when Tracking Time > Conversion Time

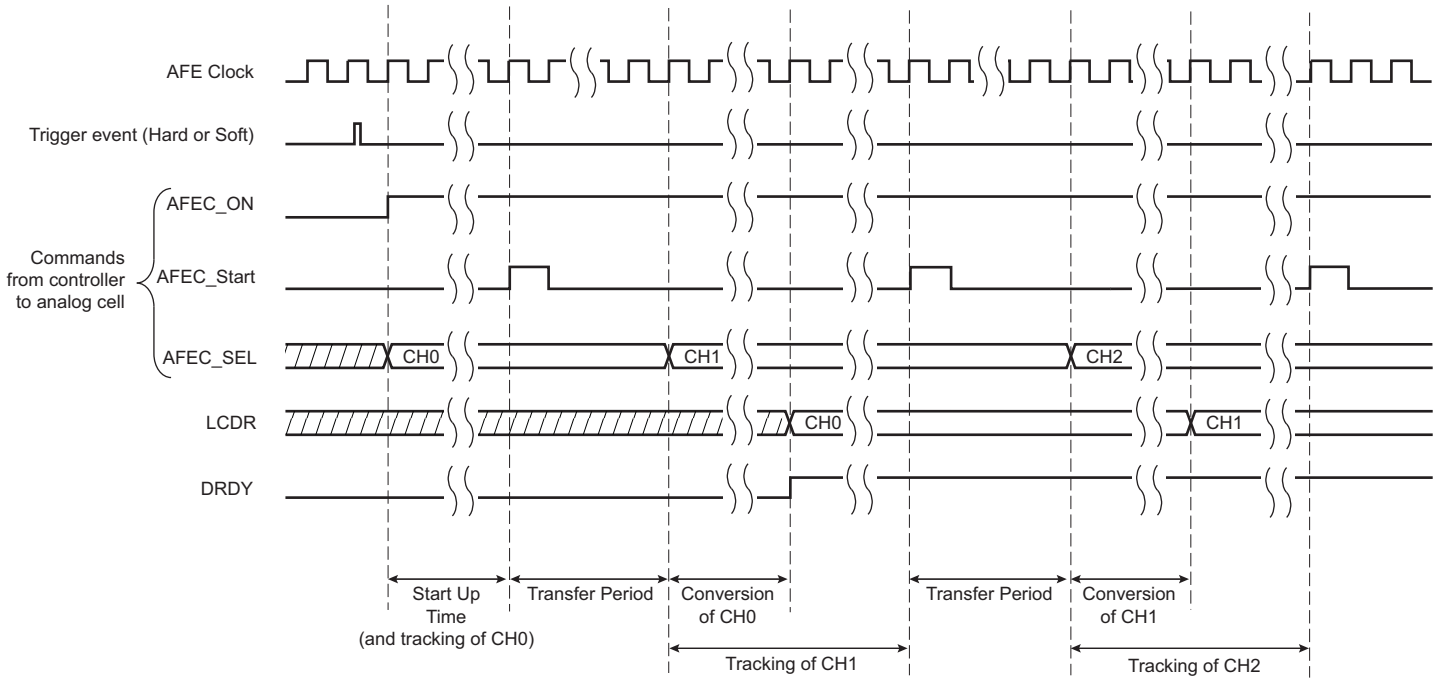
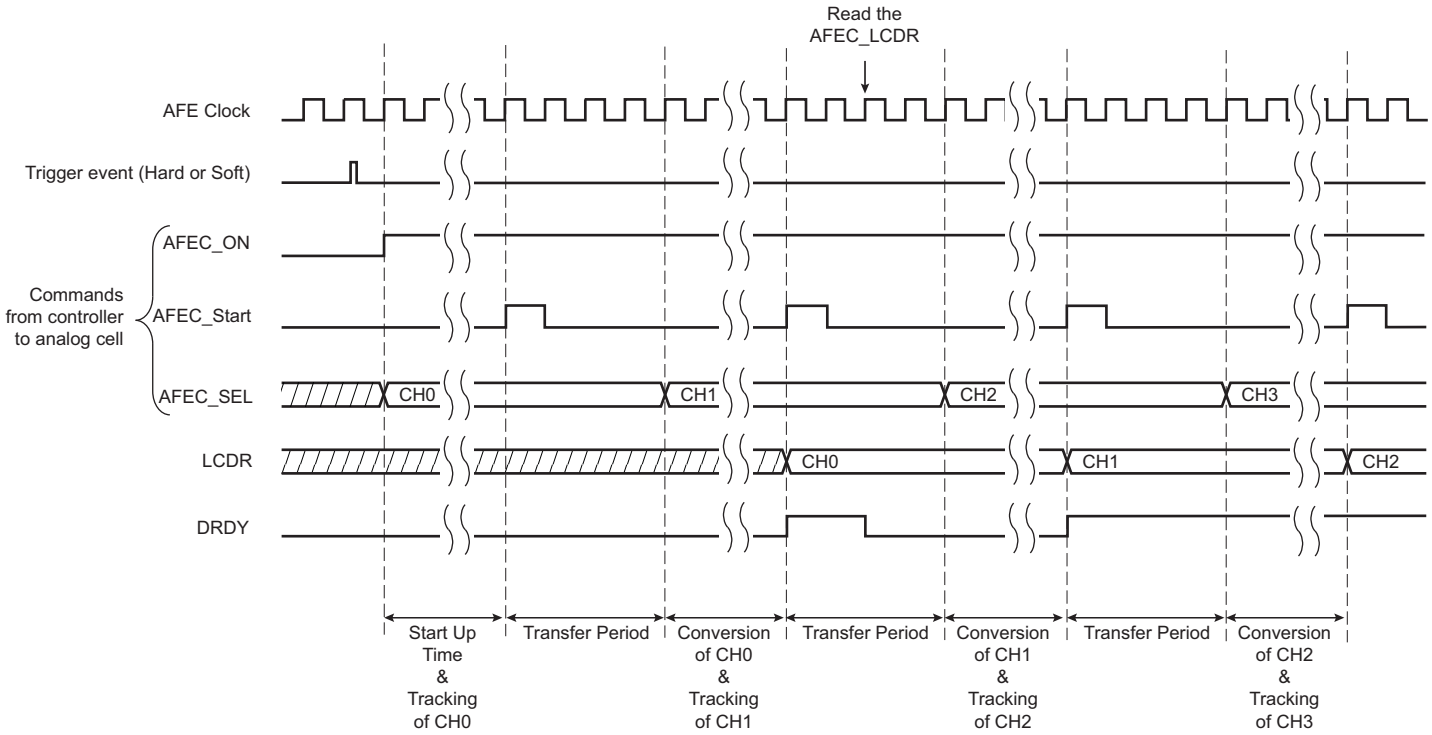


Figure 43-3. Sequence of AFE Conversions when Tracking Time < Conversion Time



43.6.2 Conversion Reference

The conversion is performed on a full range between 0V and the reference voltage carried on pin ADVREF. Analog inputs between these voltages convert to values based on a linear conversion.

43.6.3 Conversion Resolution

The AFEC supports 10-bit or 12-bit native resolutions. The 10-bit selection is performed by setting the RES field in the Extended Mode register (AFEC_EMR). By default, after a reset, the resolution is the highest and the DATA field in the data registers is fully used. By setting the RES field, the AFEC switches to the lowest resolution and the conversion results can be read in the lowest significant bits of the data registers. The highest bits of the DATA field in the corresponding Channel Data register (AFEC_CDR) and of the LDATA field in the Last Converted Data register (AFEC_LCDR) read 0. Writing two or more to the RES field in the Extended Mode register (AFEC_EMR) automatically enables the Enhanced Resolution mode. For details on this mode, see Section 43.6.12.

Moreover, when a PDC channel is connected to the AFEC, a resolution lower than 16 bits sets the transfer request size to 16 bits.

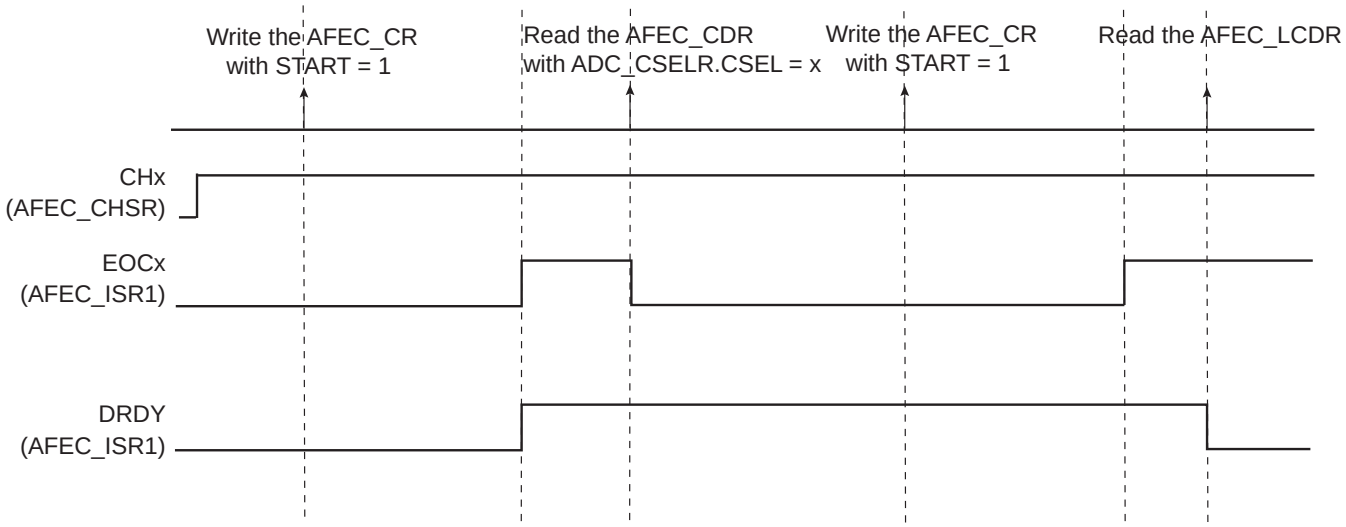
43.6.4 Conversion Results

When a conversion is completed, the resulting 12-bit digital value is stored in an internal register (one register for each channel) that can be read by means of the Channel Data Register (AFEC_CDR) and the Last Converted Data Register (AFEC_LCDR). By setting the bit TAG in the AFEC_EMR, the AFEC_LCDR presents the channel number associated with the last converted data in the CHNB field.

The bits EOCx, where ‘x’ corresponds to the value programmed in the CSEL bit of AFEC_CSELR, and DRDY in the Interrupt Status Register (AFEC_ISR) are set. In the case of a connected PDC channel, DRDY rising triggers a data transfer request. In any case, either EOCx or DRDY can trigger an interrupt.

Reading the AFEC_CDR clears the EOCx bit. Reading AFEC_LCDR clears the DRDY bit and the EOCx bit corresponding to the last converted channel.

Figure 43-4. EOCx and DRDY Flag Behavior

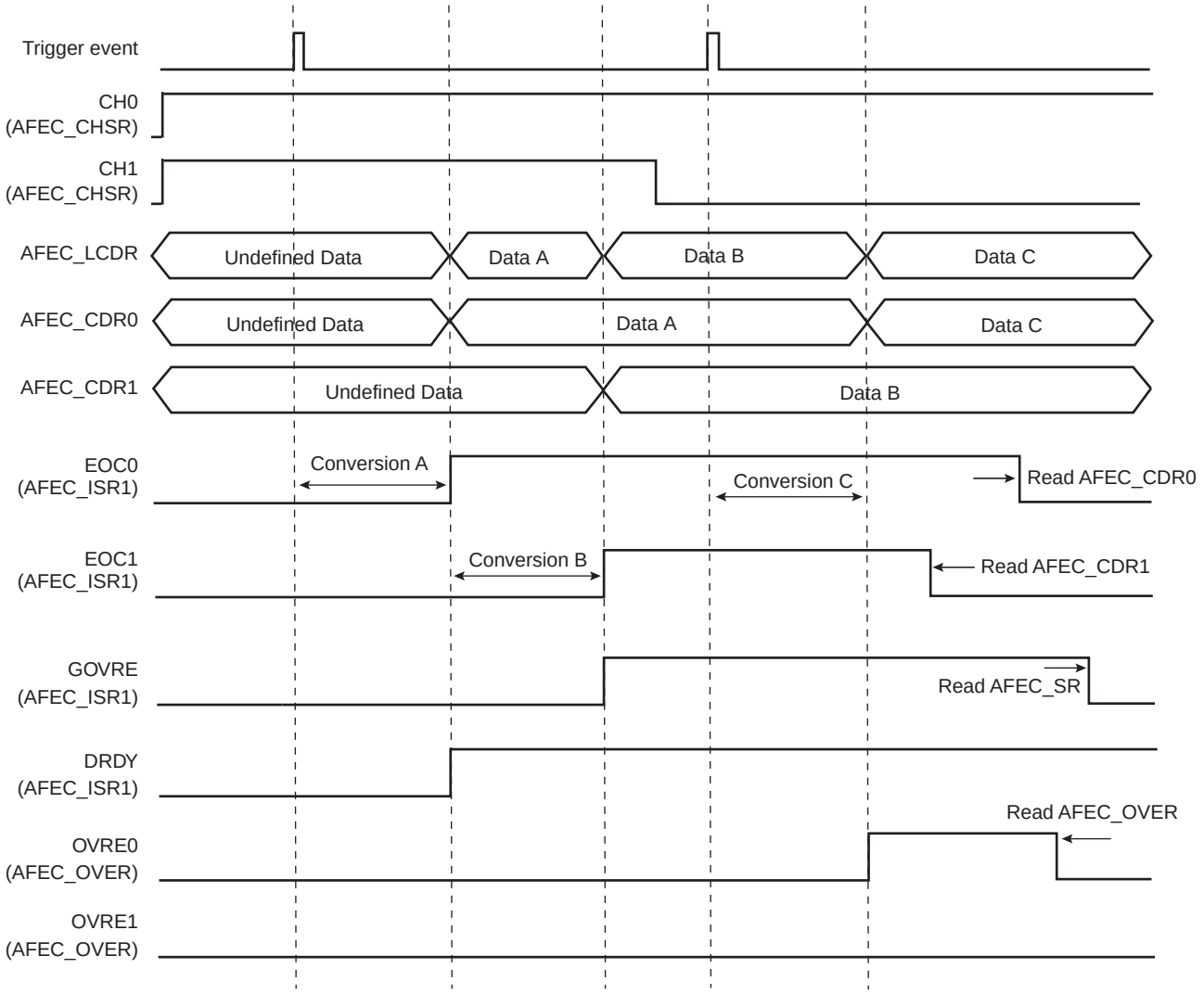


If AFEC_CDR is not read before further incoming data is converted, the corresponding OVREx flag is set in the Overrun Status Register (AFEC_OVER).

New data converted when DRDY is high sets the GOVRE bit in AFEC_ISR.

The OVREx flag is automatically cleared when AFEC_OVER is read, and the GOVRE flag is automatically cleared when AFEC_ISR is read.

Figure 43-5. EOCx, 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 EOCx and GOVRE flags in AFEC_ISR and OVREx flags in AFEC_OVER are unpredictable.

43.6.5 Conversion Triggers

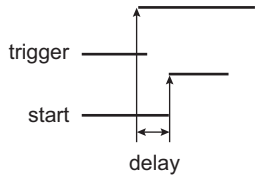
Conversions of the active analog channels are started with a software or hardware trigger. The software trigger is provided by writing the Control Register (AFEC_CR) with the START bit at 1.

The hardware trigger can be one of the TIOA outputs of the Timer Counter channels, PWM Event line, or the external trigger input of the AFEC (ADTRG). The hardware trigger is selected with the TRGSEL field in the AFEC_MR. The selected hardware trigger is enabled with the TRGEN bit in the AFEC_MR.

The minimum time between two consecutive trigger events must be strictly greater than the duration time of the longest conversion sequence according to configuration of registers AFEC_MR, AFEC_CHSR, AFEC_SEQ1R, AFEC_SEQ2R.

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 two peripheral clock periods to one AFE clock period.

Figure 43-6. Conversion Start with the Hardware Trigger



If one of the TIOA outputs is selected, the corresponding Timer Counter channel must be programmed in Waveform mode.

Only one start command is necessary to initiate a conversion sequence on all the channels. The AFEC hardware logic automatically performs the conversions on the active channels, then waits for a new request. The Channel Enable (AFEC_CHER) and Channel Disable (AFEC_CHDR) registers permit the analog channels to be enabled or disabled independently.

If the AFEC is used with a PDC, only the transfers of converted data from enabled channels are performed and the resulting data buffers should be interpreted accordingly.

43.6.6 Sleep Mode and Conversion Sequencer

The AFEC Sleep mode maximizes power saving by automatically deactivating the AFE when it is not being used for conversions. Sleep mode is selected by setting the SLEEP bit in AFEC_MR.

Sleep mode is managed by a conversion sequencer, which automatically processes the conversions of all channels at lowest power consumption.

This mode can be used when the minimum period of time between two successive trigger events is greater than the startup period of the AFEC. Refer to the AFE Characteristics in the section “Electrical Characteristics”.

When a start conversion request occurs, the AFE is automatically activated. As the analog cell requires a start-up time, the logic waits during this lapse and starts the conversion on the enabled channels. When all conversions are complete, the AFE is deactivated until the next trigger. Triggers occurring during the sequence are not taken into account.

A fast wake-up mode is available in the AFEC_MR as a compromise between power-saving strategy and responsiveness. Setting the FWUP bit enables the Fast Wake-up mode. In Fast Wake-up mode, the AFE is not fully deactivated while no conversion is requested, thereby providing lower power savings but faster wake-up.

The conversion sequencer allows automatic processing with minimum processor intervention and optimized power consumption. Conversion sequences are performed periodically using a Timer/Counter output or the PWM event line.

The PDC can automatically process the periodic acquisition of several samples without processor intervention.

The sequence can be customized by programming the Channel Sequence registers AFEC_SEQ1R and AFEC_SEQ2R and setting the USEQ bit of the AFEC_MR. The user selects a specific order of channels and can program up to 16 conversions by sequence. The user may create a personal sequence by writing channel numbers in AFEC_SEQ1R and AFEC_SEQ2R. Channel numbers can be written in any order and repeated several times. Only enabled USCHx fields are converted. Thus, to program a 15-conversion sequence, the user disables AFEC_CHSR.CH15, thus disabling the field USCH15 of AFEC_SEQ2R.

Note: The reference voltage pins always remain connected in Normal mode as in Sleep mode.

43.6.7 Comparison Window

The AFEC features automatic comparison functions. It compares converted values to a low threshold, a high threshold or both, depending on the value of the CMPMODE bit in AFEC_EMR. The comparison can be done on all channels or only on the channel specified in the CMPSEL field of AFEC_EMR. To compare all channels, the CMPALL bit in AFEC_EMR must 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 the AFEC_EMR.

The flag can be read on the COMPE bit of the AFEC_ISR and can trigger an interrupt.

The high threshold and the low threshold can be read/written in the Compare Window Register (AFEC_CWR).

43.6.8 Differential Inputs

The AFE can be used either as a single-ended AFE (AFEC_DIFFR.DIFF = 0) or as a fully differential AFE (AFEC_DIFFR.DIFF = 1). By default, after a reset, the AFE is in Single-ended mode.

If ANACH is set in AFEC_MR, the AFEC can apply a different mode on each channel. Otherwise the parameters of CH0 are applied to all channels.

The same inputs are used in Single-ended or Differential mode.

Depending on the AFE mode, the analog multiplexer selects one or two inputs to map to a channel. Table 43-4 provides input mapping for both modes.

Table 43-4. Input Pins and Channel Number

Input Pins	Channel Number	
	Single-ended Mode	Differential Mode
AFE_AD0	CH0	CH0
AFE_AD1	CH1	
...	...	...
AFE_AD14	CH14	CH14
AFE_AD15	CH15	

43.6.9 Input Gain and Offset

The AFE has a built-in programmable gain amplifier (PGA) and programmable offset per channel through a DAC.

The programmable gain amplifier can be set to gains of 1/2, 1, 2 and 4 and can be used for single-ended applications or for fully differential applications.

If ANACH is set in AFEC_MR, the AFEC can apply different gain and offset on each channel. Otherwise the parameters of CH0 are applied to all channels.

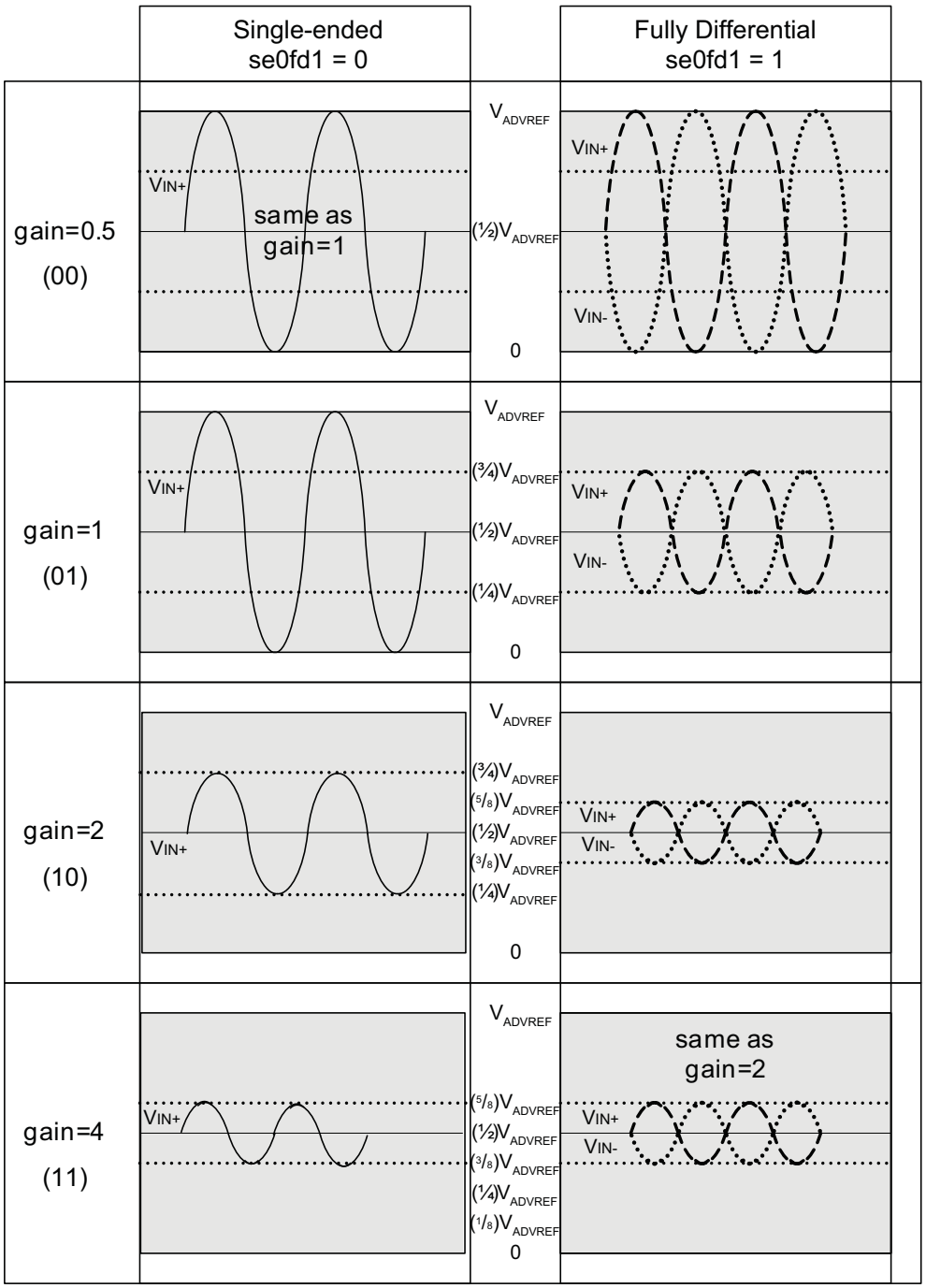
The gain is configured in the GAIN field of the Channel Gain Register (AFEC_CGR) as shown in Table 43-5.

Table 43-5. Gain of the Sample-and-Hold Unit

GAIN	GAIN (DIFFx = 0)	GAIN (DIFFx = 1)
0	1	0.5
1	1	1
2	2	2
3	4	2

The analog offset of the AFE is configured in the AOFF field in the Channel Offset Compensation register (AFEC_COCR). The offset is only available in Single-ended mode. The field AOFF must be configured to 2048 (mid scale of the DAC) when there is no offset error to compensate.To compensate for an offset error of n LSB (positive or negative), the field AOFF must be configured to 2048 + n.

Figure 43-7. Analog Full Scale Ranges in Single-Ended/Differential Applications Versus Gain



43.6.10 AFE Timings

Each AFE has its own minimal startup time configured in the field STARTUP in AFEC_MR.

When the gain, offset or differential input parameters of the analog cell change between two channels, the analog cell may need a specific settling time before starting the tracking phase. In this case, the controller waits during the settling time defined in the AFEC_MR. If the bit ANACH in AFEC_MR is cleared, this time is unused.

Warning: No input buffer amplifier to isolate the source is included in the AFE. This must be taken into consideration.

43.6.11 Temperature Sensor

The temperature sensor is internally connected to channel index 15.

The AFEC manages temperature measurement in several ways. The different methods of measurement depend on the configuration bits TRGEN in the AFEC_MR and CH15 in AFEC_CHSR.

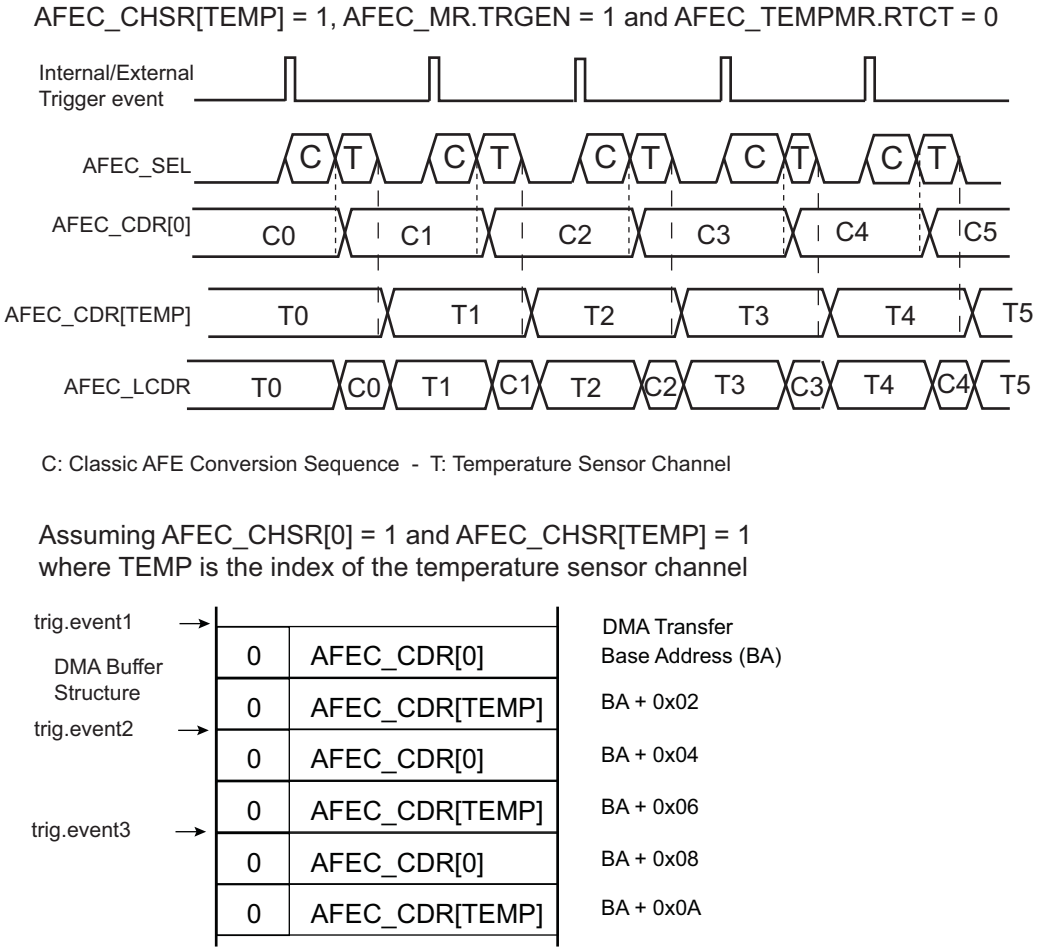
Temperature measurement can be triggered at the same rate as other channels by enabling the conversion channel 15.

If the bit CH15 in AFEC_CHSR is enabled, the temperature sensor analog cell is switched on. If a user sequence is used, the last converted channel of the sequence is always the temperature sensor channel.

A manual start can be performed only if TRGEN bit in AFEC_MR is disabled. When the START bit in AFEC_CR is set, the temperature sensor channel conversion is scheduled together with the other enabled channels (if any). The result of the conversion is placed in an internal register that can be read in the AFEC_CDR (AFEC_CSELR must be programmed accordingly prior to reading AFEC_CDR) and the associated flag EOC15 is set in the AFEC_ISR.

The channel of the temperature sensor is periodically converted together with the other enabled channels and the result is placed into AFEC_LCDR and an internal register (can be read in AFEC_CDR). Thus the temperature conversion result is part of the Peripheral DMA Controller buffer. The temperature channel can be enabled/disabled at anytime, but this may not be optimal for downstream processing.

Figure 43-8. Non-Optimized Temperature Conversion

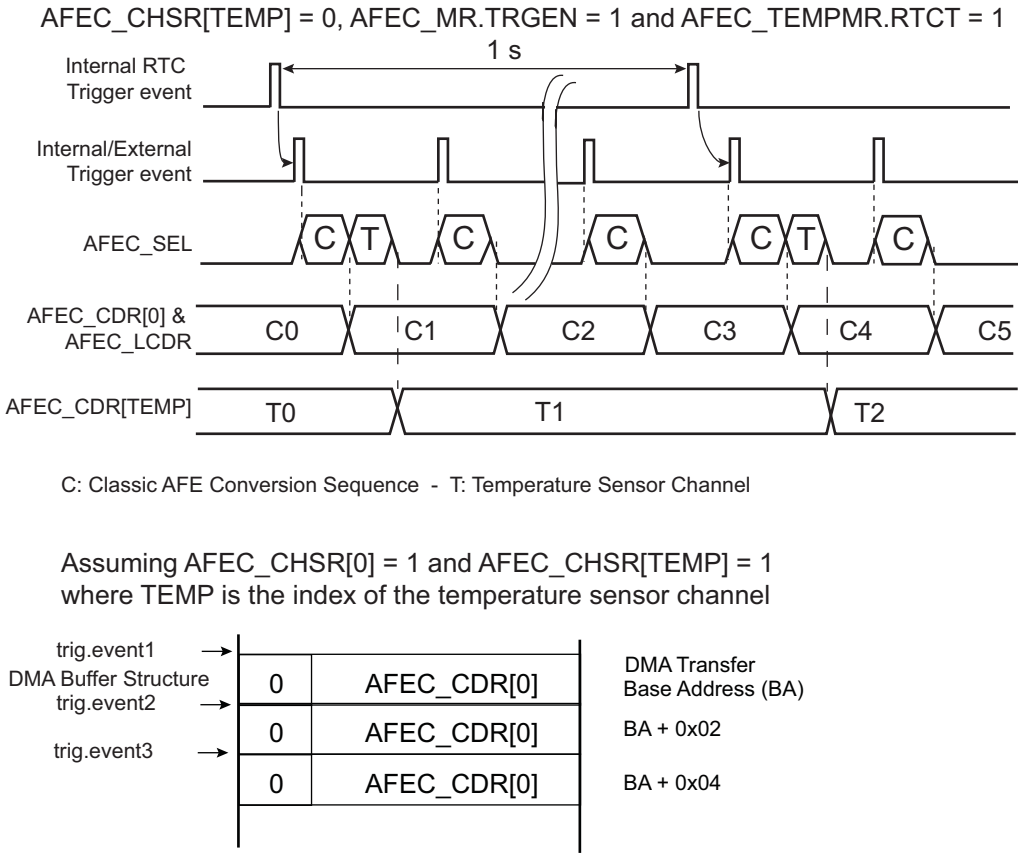


The temperature factor has a slow variation rate and may be different from other conversion channels. As a result, the AFEC allows a different way of triggering temperature measurement when the bit RTCT is set in the AFEC_TEMPMR but the CH15 is cleared in the AFEC_CHSR.

In this configuration, the measurement is triggered every second by means of an internal trigger generated by the RTC. This trigger is always enabled and independent of the triggers used for other channels. It is selected in the TRGSEL field in AFEC_MR. In this mode of operation, the temperature sensor is only powered for a period of time covering startup time and conversion time.

Every second, a conversion is scheduled for channel 15 but the result of the conversion is only uploaded to an internal register read by means of AFEC_CDR, and not to AFEC_LCDR. Therefore, the temperature channel is not part of the Peripheral DMA Controller buffer; only the enabled channel are kept in the buffer. The end of conversion of the temperature channel is reported by means of the EOC15 flag in AFEC_ISR.

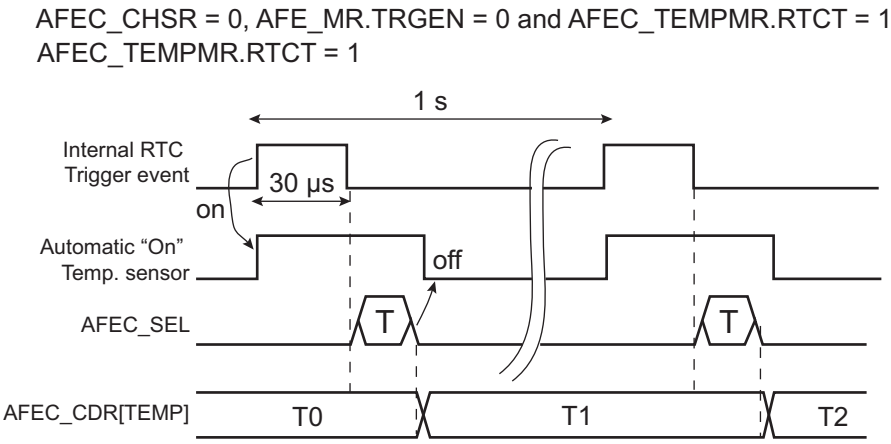
Figure 43-9. Optimized Temperature Conversion Combined with Classical Conversions



If RTCT is set and TRGEN is cleared, then all channels are disabled (AFEC_CHSR = 0) and only channel 15 is converted at a rate of one conversion per second.

This mode of operation, when combined with Sleep mode operation, provides a low-power mode for temperature measurement assuming there is no other AFE conversion to schedule at a higher sampling rate or no other channel to convert.

Figure 43-10. Temperature Conversion Only



Moreover, it is possible to raise a flag only if there is predefined change in the temperature measurement. The user can define a range of temperature or a threshold in AFEC_TEMPCWR and the mode of comparison in AFEC_TEMPMR. These values define the way the TEMPCHG flag will be raised in AFEC_ISR.

The TEMPCHG flag can be used to trigger an interrupt if there is an update/modification to be made in the system resulting from a temperature change.

In any case, if temperature sensor measurement is configured, the temperature can be read at anytime in AFEC_CDR (AFEC_CSELR must be programmed accordingly prior to reading AFEC_CDR) .

43.6.12 Enhanced Resolution Mode and Digital Averaging Function

The Enhanced Resolution mode is enabled when the field RES is set to 13-bit resolution or higher in AFEC_EMR. In this mode, the AFEC trades conversion performance for accuracy by averaging multiple samples, thus providing a digital low-pass filter function. The resolution mode selected determines the oversampling, which represents the performance reduction factor.

To increase the accuracy by averaging multiple samples, some noise must be present in the input signal. The noise level should be between one and two LSB peak-to-peak to get good averaging performance.

Table 43-6 summarizes the oversampling ratio depending on the resolution mode selected.

Table 43-6. Resolution and Oversampling Ratio

Resolution Mode	Oversampling Ratio
13-bit	4
14-bit	16
15-bit	64
16-bit	256

Free Run mode is not supported if Enhanced Resolution mode is used.

The selected oversampling ratio applies to all enabled channels except the temperature sensor channel if triggered by an RTC event. See Section 43.6.11 “Temperature Sensor”.

The average result is valid into an internal register (read by means of the AFEC_CDR) only if EOCx (x corresponding to the index of the channel) flag is set in AFEC_ISR and OVREx flag is cleared in the AFEC_OVER. The average result is valid for all channels in the AFEC_LCDR only if DRDY is set and GOVRE is cleared in the AFEC_ISR.

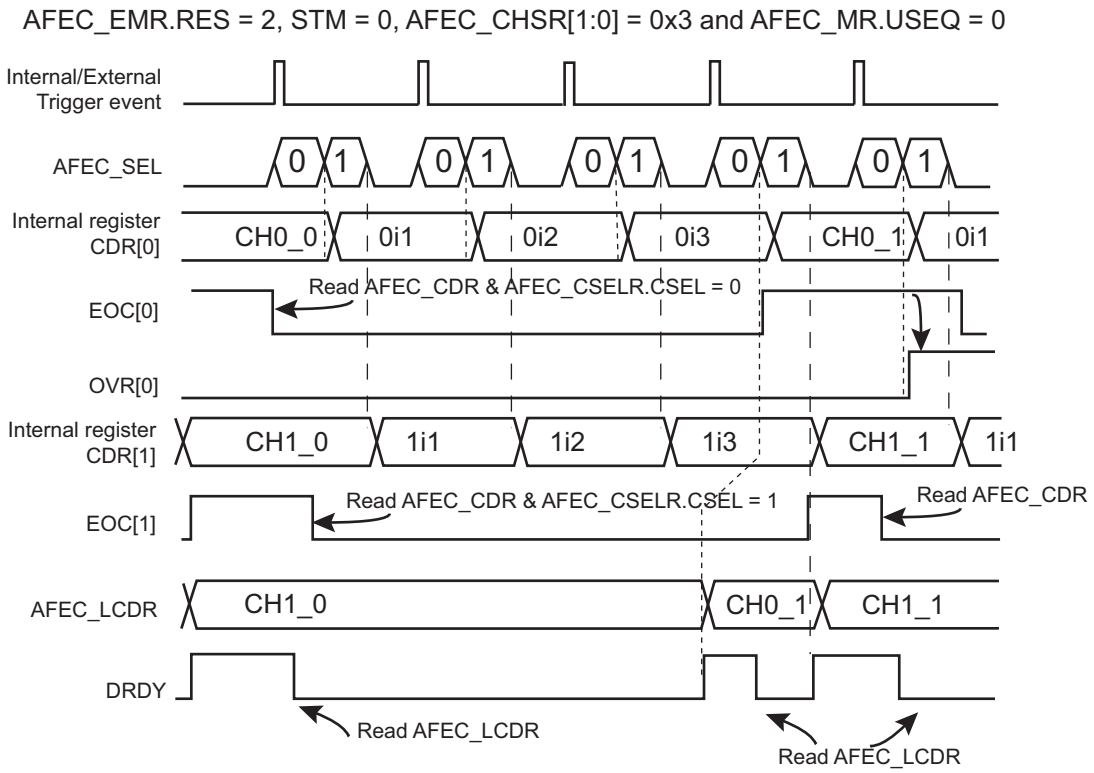
Note that the AFEC_CDR is not buffered. Therefore, when an averaging sequence is on- going, the value in this register changes after each averaging sample. However, overrun flags in the AFEC_OVER rise as soon as the first sample of an averaging sequence is received. Thus the previous averaged value is not read, even if the new averaged value is not ready.

As a result, when an overrun flag rises in the AFEC_OVER, this indicates only that the previous unread data is lost. It does not indicate that this data has been overwritten by the new averaged value, as the averaging sequence concerning this channel can still be on-going.

The samples can be defined in different ways for the averaging function depending on the configuration of the STM bit in AFEC_EMR and the USEQ bit in AFEC_MR.

When USEQ is cleared, there are two possible ways to generate the averaging through the trigger event. If the STM bit is cleared in AFEC_EMR, every trigger event generates one sample for each enabled channel, as described in Figure 43-11. Therefore, four trigger events are requested to get the result of averaging if RES = 2.

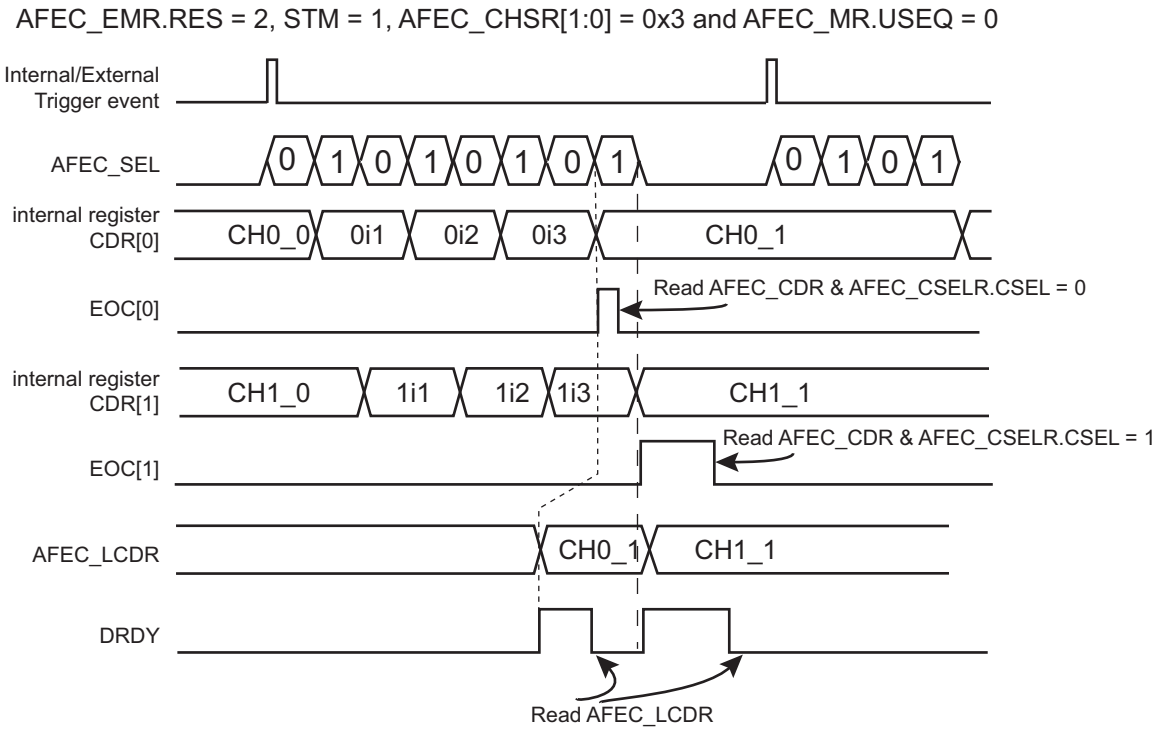
Figure 43-11. Digital Averaging Function Waveforms over Multiple Trigger Events



Note: 0i1,0i2,0i3, 1i1, 1i2, 1i3 are intermediate results and CH0/1_0/1 are final result of average function.

If the STM bit is set in AFEC_EMR and the USEQ bit is cleared in AFEC_MR, the sequence to be converted, defined in the AFEC_CHSR, is automatically repeated n times, where n corresponds to the oversampling ratio defined in the RES field in AFEC_EMR. As a result, only one trigger is required to get the result of the averaging function as shown in Figure 43-12.

Figure 43-12. Digital Averaging Function Waveforms on a Single Trigger Event



Note: 0i1, 0i2, 0i3, 1i1, 1i2, 1i3 are intermediate results and CH0/1_0/1 are final result of average function.

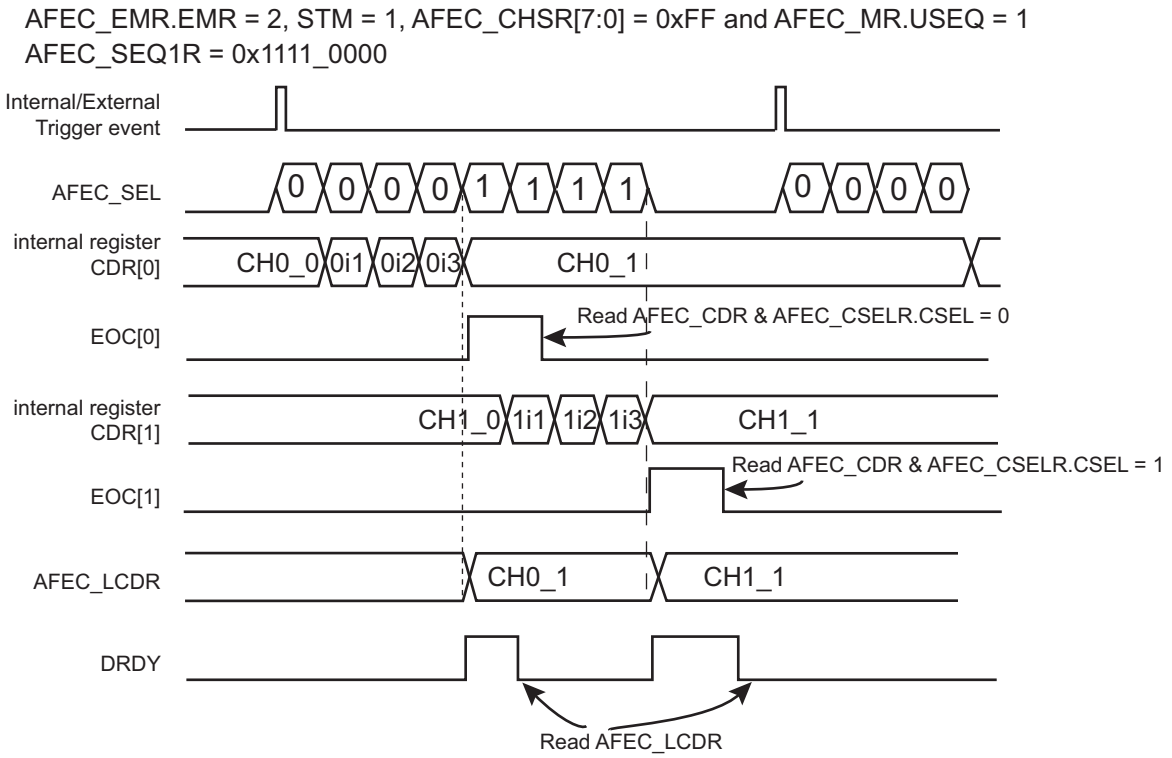
When USEQ is set, the user can define the channel sequence to be converted by configuring AFEC_SEQxR and AFEC_CHER so that channels are not interleaved during the averaging period. Under these conditions, a sample is defined for each end of conversion as described in Figure 43-13.

Therefore, if the same channel is configured to be converted four times consecutively and RES = 2 in the AFEC_EMR, the averaging result is placed in the corresponding channel internal data register (read by means of the AFEC_CDR) and the AFEC_LCDR for each trigger event.

In this case, the AFE real sample rate remains the maximum AFE sample rate divided by 4.

When USEQ is set and the RES field enables the Enhanced Resolution mode, it is important to note that the user sequence must be a sequence being an integer multiple of 4 (i.e., the number of the enabled channel in the Channel Status register (AFEC_CHSR) must be an integer multiple of 4 and the AFEC_SEQxR must be a series of 4 times the same channel index).

Figure 43-13. Digital Averaging Function Waveforms on a Single Trigger Event, Non-interleaved



Note: 0i1, 0i2, 0i3, 1i1, 1i2, 1i3 are intermediate results and CH0/1_0/1 are final result of average function.

43.6.13 Automatic Calibration

The AFE features an automatic calibration (AUTOCALIB) mode for gain errors (calibration).

The automatic calibration sequence can be started at any time by setting the AUTOCAL bit of the AFEC_CR. The end of calibration sequence is given by the EOCAL bit in AFEC_ISR, and an interrupt is generated if the EOCAL interrupt has been enabled (AFEC_IER).

If Free Run mode is to be used, then automatic calibration must be run before enabling the Free Run mode. In any case, automatic calibration should not be started while Free Run mode is active.

The calibration sequence performs an automatic calibration on all enabled channels. The gain settings of all enabled channels must be set before starting the automatic calibration sequence. For each calibrated channel, the corresponding OFFx bit in AFEC_CDOR must be set before launching the autocalibration sequence.

If the gain settings (AFEC_CGR) for a given channel are changed, the automatic calibration sequence must be started again.

The calibration data on one or more enabled channels is stored in the internal AFE memory.

Then, when a new conversion is started on one or more enabled channels, the converted value in AFEC_LCDR or internal data registers read by means of the AFEC_CDR is a calibrated value.

Automatic calibration is for settings, not for channels. Therefore, if a specific combination of gain and offset has already been calibrated, and a new channel with the same settings is enabled after the initial calibration, there is no need to restart a calibration. If different enabled channels have different gain and offset settings, the corresponding channels must be enabled before starting the calibration.

If a software reset is performed (SWRST =1 in AFEC_CR) or after power up (or wake-up from Backup mode), the calibration data in the AFE memory is lost.

Changing the AFEC running mode in the AFEC_MR does not affect the calibration data.

Changing the AFE reference voltage (ADVREF pin) requires a new calibration sequence.

For calibration time, offset and gain error after calibration, refer to the AFE Characteristics in the section “Electrical Characteristics”.

43.6.14 Buffer Structure

The PDC read channel is triggered each time a new data is stored in AFEC_LCDR. The same structure of data is repeatedly stored in AFEC_LCDR each time a trigger event occurs. Depending on the user mode of operation (AFEC_MR, AFEC_CHSR, AFEC_SEQ1R, AFEC_SEQ2R) the structure differs. When TAG is cleared, each data transferred to PDC buffer is carried on a half-word (16-bit) and consists of the last converted data right-aligned. When TAG is set, this data is carried on a word buffer (32-bit) and CHNB carries the channel number, thus simplifying post-processing in the PDC buffer and ensuring the integrity of the PDC buffer.

43.6.15 Fault Output

The AFEC internal fault output is directly connected to PWM fault input. Fault output may be asserted depending on the configuration of AFEC_EMR and AFEC_CWR and converted values. When the compare occurs, the AFEC fault output generates a pulse of one peripheral clock cycle to the PWM fault input. This fault line can be enabled or disabled within the PWM. If it is activated and asserted by the AFEC, the PWM outputs are immediately placed in a safe state (pure combinational path). Note that the AFEC fault output connected to the PWM is not the COMPE bit. Thus the Fault Mode (FMOD) within the PWM configuration must be FMOD = 1.

43.6.16 Register Write Protection

To prevent any single software error from corrupting AFEC behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the AFEC Write Protection Mode Register (AFEC_WPMR).

If a write access to a write-protected register is detected, the WPVS flag in the AFEC Write Protection Status Register (AFEC_WPSR) is set and the field WPVSRC indicates the register in which the write access has been attempted.

The WPVS flag is automatically cleared by reading the AFEC_WPSR.

The protected registers are:

- AFEC Mode Register
- AFEC Extended Mode Register
- AFEC Channel Sequence 1 Register
- AFEC Channel Sequence 2 Register
- AFEC Channel Enable Register
- AFEC Channel Disable Register
- AFEC Compare Window Register
- AFEC Channel Gain Register
- AFEC Channel Calibration DC Offset Register
- AFEC Channel Differential Register
- AFEC Channel Selection Register
- AFEC Channel Offset Compensation Register
- AFEC Temperature Sensor Mode Register
- AFEC Temperature Compare Window Register
- AFEC Analog Control Register

43.7 Analog Front-End Controller (AFEC) User Interface

Table 43-7. Register Mapping

Offset[1]	Register	Name	Access	Reset
0x00	AFEC Control Register	AFEC_CR	Write-only	–
0x04	AFEC Mode Register	AFEC_MR	Read/Write	0x00000000
0x08	AFEC Extended Mode Register	AFEC_EMR	Read/Write	0x00000000
0x0C	AFEC Channel Sequence 1 Register	AFEC_SEQ1R	Read/Write	0x00000000
0x10	AFEC Channel Sequence 2 Register	AFEC_SEQ2R	Read/Write	0x00000000
0x14	AFEC Channel Enable Register	AFEC_CHER	Write-only	–
0x18	AFEC Channel Disable Register	AFEC_CHDR	Write-only	–
0x1C	AFEC Channel Status Register	AFEC_CHSR	Read-only	0x00000000
0x20	AFEC Last Converted Data Register	AFEC_LCDR	Read-only	0x00000000
0x24	AFEC Interrupt Enable Register	AFEC_IER	Write-only	–
0x28	AFEC Interrupt Disable Register	AFEC_IDR	Write-only	–
0x2C	AFEC Interrupt Mask Register	AFEC_IMR	Read-only	0x00000000
0x30	AFEC Interrupt Status Register	AFEC_ISR	Read-only	0x00000000
0x34–0x40	Reserved	–	–	–
0x44–0x48	Reserved	–	–	–
0x4C	AFEC Overrun Status Register	AFEC_OVER	Read-only	0x00000000
0x50	AFEC Compare Window Register	AFEC_CWR	Read/Write	0x00000000
0x54	AFEC Channel Gain Register	AFEC_CGR	Read/Write	0x00000000
0x5C	AFEC Channel Calibration DC Offset Register	AFEC_CDOR	Read/Write	0x00000000
0x60	AFEC Channel Differential Register	AFEC_DIFFR	Read/Write	0x00000000
0x64	AFEC Channel Selection Register	AFEC_CSELR	Read/Write	0x00000000
0x68	AFEC Channel Data Register	AFEC_CDR	Read-only	0x00000000
0x6C	AFEC Channel Offset Compensation Register	AFEC_COCR	Read/Write	0x00000000
0x70	AFEC Temperature Sensor Mode Register	AFEC_TEMPMR	Read/Write	0x00000000
0x74	AFEC Temperature Compare Window Register	AFEC_TEMPCWR	Read/Write	0x00000000
0x78–0x90	Reserved	–	–	–
0x94	AFEC Analog Control Register	AFEC_ACR	Read/Write	0x00000100
0x98–0xAC	Reserved	–	–	–
0xB0–0xE0	Reserved	–	–	–
0xE4	AFEC Write Protection Mode Register	AFEC_WPMR	Read/Write	0x00000000
0xE8	AFEC Write Protection Status Register	AFEC_WPSR	Read-only	0x00000000
0xEC–0xF8	Reserved	–	–	–
0xFC	Reserved	–	–	–

Notes: 1. Any offset not listed in Table 43-7 must be considered as “reserved”.

43.7.1 AFEC Control Register

Name: AFEC_CR

Address: 0x400B0000 (0), 0x400B4000 (1)

Access: Write-only

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

- **SWRST: Software Reset**

0: No effect.

1: Resets the AFEC simulating a hardware reset.

- **START: Start Conversion**

0: No effect.

1: Begins Analog Front-End conversion.

- **AUTOCAL: Automatic Calibration of AFE**

0: No effect.

1: Launches an automatic calibration of the AFE on the next sequence.

43.7.2 AFEC Mode Register

Name: AFEC_MR

Address: 0x400B0004 (0), 0x400B4004 (1)

Access: Read/Write

31	30	29	28	27	26	25	24
USEQ	–	TRANSFER		TRACKTIM			
23	22	21	20	19	18	17	16
ANACH	–	SETTLING		STARTUP			
15	14	13	12	11	10	9	8
PRESCAL							
7	6	5	4	3	2	1	0
FREERUN	FWUP	SLEEP	–	TRGSEL			TRGEN

This register can only be written if the WPEN bit is cleared in the AFEC Write Protection Mode Register.

- **TRGEN: Trigger Enable**

Value	Name	Description
0	DIS	Hardware triggers are disabled. Starting a conversion is only possible by software.
1	EN	Hardware trigger selected by TRGSEL field is enabled.

- **TRGSEL: Trigger Selection**

Value	Name	Description
0	AFEC_TRIG0	ADTRG pin
1	AFEC_TRIG1	TIO Output of the Timer Counter Channel 0
2	AFEC_TRIG2	TIO Output of the Timer Counter Channel 1
3	AFEC_TRIG3	TIO Output of the Timer Counter Channel 2
4	AFEC_TRIG4	PWM Event Line 0
5	AFEC_TRIG5	PWM Event Line 1
6	AFEC_TRIG6	Reserved
7	–	Reserved

- **SLEEP: Sleep Mode**

Value	Name	Description
0	NORMAL	Normal mode: The AFE and reference voltage circuitry are kept ON between conversions.
1	SLEEP	Sleep mode: The AFE 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 AFE 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**

PRESCAL = $f_{peripheral\ clock}$/ ($f_{AFE\ Clock} \times 2$) - 1

- **STARTUP: Start-up Time**

Value	Name	Description
0	SUT0	0 periods of AFE clock
1	SUT8	8 periods of AFE clock
2	SUT16	16 periods of AFE clock
3	SUT24	24 periods of AFE clock
4	SUT64	64 periods of AFE clock
5	SUT80	80 periods of AFE clock
6	SUT96	96 periods of AFE clock
7	SUT112	112 periods of AFE clock
8	SUT512	512 periods of AFE clock
9	SUT576	576 periods of AFE clock
10	SUT640	640 periods of AFE clock
11	SUT704	704 periods of AFE clock
12	SUT768	768 periods of AFE clock
13	SUT832	832 periods of AFE clock
14	SUT896	896 periods of AFE clock
15	SUT960	960 periods of AFE clock

- **SETTLING: Analog Settling Time**

Value	Name	Description
0	AST3	3 periods of AFE clock
1	AST5	5 periods of AFE clock
2	AST9	9 periods of AFE clock
3	AST17	17 periods of AFE clock

- **ANACH: Analog Change**

Value	Name	Description
0	NONE	No analog change on channel switching: DIFF0, GAIN0 are used for all channels
1	ALLOWED	Allows different analog settings for each channel. See AFEC_CGR.

- **TRACKTIM: Tracking Time**

Inherent tracking time is always 15 AFE clock cycles.

- **TRANSFER: Transfer Period**

The TRANSFER field should be configured to 2 to guarantee the optimal transfer time.

- **USEQ: User Sequence Enable**

Value	Name	Description
0	NUM_ORDER	Normal mode: The controller converts channels in a simple numeric order.
1	REG_ORDER	User Sequence mode: The sequence respects what is defined in AFEC_SEQ1R and AFEC_SEQ1R.

43.7.3 AFEC Extended Mode Register

Name: AFEC_EMR

Address: 0x400B0008 (0), 0x400B4008 (1)

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	STM	TAG
23	22	21	20	19	18	17	16
–	–	–	–	–	RES		
15	14	13	12	11	10	9	8
–	–	CMPFILTER		–	–	CMPALL	–
7	6	5	4	3	2	1	0
CMPSEL					–	CMPMODE	

This register can only be written if the WPEN bit is cleared in the AFEC Write Protection Mode Register.

- **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 the channel indicated in CMPSEL field is compared.

1: All channels are compared.

- **CMPFILTER: Compare Event Filtering**

Number of consecutive compare events necessary to raise the flag = CMPFILTER+1.

When programmed to ‘0’, the flag rises as soon as an event occurs.

- **RES: Resolution**

Value	Name	Description
0	NO_AVERAGE	12-bit resolution, AFE sample rate is maximum (no averaging).
1	LOW_RES	10-bit resolution, AFE sample rate is maximum (no averaging).
2	OSR4	13-bit resolution, AFE sample rate divided by 4 (averaging).
3	OSR16	14-bit resolution, AFE sample rate divided by 16 (averaging).
4	OSR64	15-bit resolution, AFE sample rate divided by 64 (averaging).
5	OSR256	16-bit resolution, AFE sample rate divided by 256 (averaging).

- **TAG: TAG of the AFEC_LDCR**

0: Clears CHNB in AFEC_LDCR.

1: Appends the channel number to the conversion result in AFEC_LDCR.

- **STM: Single Trigger Mode**

0: Multiple triggers are required to get an averaged result.

1: Only a single trigger is required to get an averaged value.

43.7.4 AFEC Channel Sequence 1 Register

Name: AFEC_SEQ1R

Address: 0x400B000C (0), 0x400B400C (1)

Access: Read/Write

31	30	29	28	27	26	25	24
USCH7				USCH6			
23	22	21	20	19	18	17	16
USCH5				USCH4			
15	14	13	12	11	10	9	8
USCH3				USCH2			
7	6	5	4	3	2	1	0
USCH1				USCH0			

This register can only be written if the WPEN bit is cleared in the AFEC Write Protection Mode Register.

- **USCHx: User Sequence Number x**

The sequence number x (USCHx) can be programmed by the Channel number CHy where y is the value written in this field. The allowed range is 0 up to 15. So it is only possible to use the sequencer from CH0 to CH15.

This register activates only if AFEC_MR.USEQ bit is set.

Any USCHx field is taken into account only if the AFEC_CHSR.CHx bit is set, 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, depending on user needs.

43.7.5 AFEC Channel Sequence 2 Register

Name: AFEC_SEQ2R

Address: 0x400B0010 (0), 0x400B4010 (1)

Access: Read/Write

31	30	29	28	27	26	25	24
USCH15				USCH14			
23	22	21	20	19	18	17	16
USCH13				USCH12			
15	14	13	12	11	10	9	8
USCH11				USCH10			
7	6	5	4	3	2	1	0
USCH9				USCH8			

This register can only be written if the WPEN bit is cleared in the AFEC Write Protection Mode Register.

- **USCHx: User Sequence Number x**

The sequence number x (USCHx) can be programmed by the Channel number CHy where y is the value written in this field. The allowed range is 0 up to 15. So it is only possible to use the sequencer from CH0 to CH15.

This register activates only if AFEC_MR.USEQ field is set.

Any USCHx field is taken into account only if the AFEC_CHSR.CHx bit is written to one, else any value written in USCHx does not add the corresponding channel in the conversion sequence.

Configuring the same value in different fields leads to multiple samples of the same channel during the conversion sequence. This can be done consecutively, or not, according to user needs.

43.7.6 AFEC Channel Enable Register

Name: AFEC_CHER

Address: 0x400B0014 (0), 0x400B4014 (1)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8
7	6	5	4	3	2	1	0
CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0

This register can only be written if the WPEN bit is cleared in the AFEC Write Protection Mode Register.

- **CHx: Channel x Enable**

0: No effect.

1: Enables the corresponding channel.

Note: If USEQ = 1 in the AFEC_MR, CHx corresponds to the xth channel of the sequence described in AFEC_SEQ1R, AFEC_SEQ2R.

43.7.7 AFEC Channel Disable Register

Name: AFEC_CHDR

Address: 0x400B0018 (0), 0x400B4018 (1)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8
7	6	5	4	3	2	1	0
CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0

This register can only be written if the WPEN bit is cleared in the AFEC Write Protection 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 and then reenabled during a conversion, its associated data and its corresponding EOCx and GOVRE flags in AFEC_ISR and OVREx flags in AFEC_OVER are unpredictable.

43.7.8 AFEC Channel Status Register

Name: AFEC_CHSR

Address: 0x400B001C (0), 0x400B401C (1)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8
7	6	5	4	3	2	1	0
CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0

- **CHx: Channel x Status**

0: The corresponding channel is disabled.

1: The corresponding channel is enabled.

43.7.9 AFEC Last Converted Data Register

Name: AFEC_LCDR

Address: 0x400B0020 (0), 0x400B4020 (1)

Access: Read-only

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

- **LDATA: Last Data Converted**

The AFE 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 TAG is set in the AFEC_EMR. If TAG is cleared, CHNB = 0.

43.7.10 AFEC Interrupt Enable Register

Name: AFEC_IER

Address: 0x400B0024 (0), 0x400B4024 (1)

Access: Write-only

31	30	29	28	27	26	25	24
EOCAL	TEMPCHG	–	RXBUFF	ENDRX	COMPE	GOVRE	DRDY
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
EOC15	EOC14	EOC13	EOC12	EOC11	EOC10	EOC9	EOC8
7	6	5	4	3	2	1	0
EOC7	EOC6	EOC5	EOC4	EOC3	EOC2	EOC1	EOC0

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **EOCx: End of Conversion Interrupt Enable x**
- **DRDY: Data Ready Interrupt Enable**
- **GOVRE: General Overrun Error Interrupt Enable**
- **COMPE: Comparison Event Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable**
- **RXBUFF: Receive Buffer Full Interrupt Enable**
- **TEMPCHG: Temperature Change Interrupt Enable**
- **EOCAL: End of Calibration Sequence Interrupt Enable**

43.7.11 AFEC Interrupt Disable Register

Name: AFEC_IDR

Address: 0x400B0028 (0), 0x400B4028 (1)

Access: Write-only

31	30	29	28	27	26	25	24
EOCAL	TEMPCHG	–	RXBUFF	ENDRX	COMPE	GOVRE	DRDY
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
EOC15	EOC14	EOC13	EOC12	EOC11	EOC10	EOC9	EOC8
7	6	5	4	3	2	1	0
EOC7	EOC6	EOC5	EOC4	EOC3	EOC2	EOC1	EOC0

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Disables the corresponding interrupt.

- **EOCx: End of Conversion Interrupt Disable x**
- **DRDY: Data Ready Interrupt Disable**
- **GOVRE: General Overrun Error Interrupt Disable**
- **COMPE: Comparison Event Interrupt Disable**
- **ENDRX: End of Receive Buffer Interrupt Disable**
- **RXBUFF: Receive Buffer Full Interrupt Disable**
- **TEMPCHG: Temperature Change Interrupt Disable**
- **EOCAL: End of Calibration Sequence Interrupt Disable**

43.7.12 AFEC Interrupt Mask Register

Name: AFEC_IMR

Address: 0x400B002C (0), 0x400B402C (1)

Access: Read-only

31	30	29	28	27	26	25	24
EOCAL	TEMPCHG	–	RXBUFF	ENDRX	COMPE	GOVRE	DRDY
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
EOC15	EOC14	EOC13	EOC12	EOC11	EOC10	EOC9	EOC8
7	6	5	4	3	2	1	0
EOC7	EOC6	EOC5	EOC4	EOC3	EOC2	EOC1	EOC0

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is disabled.

1: The corresponding interrupt is enabled.

- **EOCx: End of Conversion Interrupt Mask x**
- **DRDY: Data Ready Interrupt Mask**
- **GOVRE: General Overrun Error Interrupt Mask**
- **COMPE: Comparison Event Interrupt Mask**
- **ENDRX: End of Receive Buffer Interrupt Mask**
- **RXBUFF: Receive Buffer Full Interrupt Mask**
- **TEMPCHG: Temperature Change Interrupt Mask**
- **EOCAL: End of Calibration Sequence Interrupt Mask**

43.7.13 AFEC Interrupt Status Register

Name: AFEC_ISR

Address: 0x400B0030 (0), 0x400B4030 (1)

Access: Read-only

31	30	29	28	27	26	25	24
EOCAL	TEMPCHG	–	RXBUFF	ENDRX	COMPE	GOVRE	DRDY
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
EOC15	EOC14	EOC13	EOC12	EOC11	EOC10	EOC9	EOC8
7	6	5	4	3	2	1	0
EOC7	EOC6	EOC5	EOC4	EOC3	EOC2	EOC1	EOC0

- **EOCx: End of Conversion x (cleared by reading AFEC_CDRx)**

0: The corresponding analog channel is disabled, or the conversion is not finished. This flag is cleared when reading the AFEC_CDR if the CSEL bit is programmed with ‘x’ in the AFEC_CSELR.

1: The corresponding analog channel is enabled and conversion is complete.

- **TEMPCHG: Temperature Change (cleared on read)**

0: There is no comparison match (defined in the AFEC_TEMPCMPR) since the last read of AFEC_ISR.

1: The temperature value reported on AFEC_CDR (AFEC_CSELR.CSEL = 15) has changed since the last read of AFEC_ISR, according to what is defined in the Temperature Mode register (AFEC_TEMPMR) and the Temperature Compare Window register (AFEC_TEMPCWR).

- **DRDY: Data Ready (cleared by reading AFEC_LCDR)**

0: No data has been converted since the last read of AFEC_LCDR.

1: At least one data has been converted and is available in AFEC_LCDR.

- **GOVRE: General Overrun Error (cleared by reading AFEC_ISR)**

0: No general overrun error occurred since the last read of AFEC_ISR.

1: At least one general overrun error has occurred since the last read of AFEC_ISR.

- **COMPE: Comparison Error (cleared by reading AFEC_ISR)**

0: No comparison error since the last read of AFEC_ISR.

1: At least one comparison error has occurred since the last read of AFEC_ISR.

- **ENDRX: End of RX Buffer (cleared by writing AFEC_RCR or AFEC_RNCR)**

0: The Receive Counter Register has not reached 0 since the last write in AFEC_RCR(1) or AFEC_RNCR(1).

1: The Receive Counter Register has reached 0 since the last write in AFEC_RCR or AFEC_RNCR.

- **RXBUFF: RX Buffer Full (cleared by writing AFEC_RCR or AFEC_RNCR)**

0: AFEC_RCR or AFEC_RNCR has a value other than 0.

1: Both AFEC_RCR and AFEC_RNCR have a value of 0.

Note: 1. AFEC_RCR and AFEC_RNCR are PDC registers.

- **EOCAL: End of Calibration Sequence (cleared on read)**

0: Calibration sequence is on-going, or no calibration sequence has been requested.

1: Calibration sequence is complete.

43.7.14 AFEC Overrun Status Register

Name: AFEC_OVER

Address: 0x400B004C (0), 0x400B404C (1)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
OVRE15	OVRE14	OVRE13	OVRE12	OVRE11	OVRE10	OVRE9	OVRE8
7	6	5	4	3	2	1	0
OVRE7	OVRE6	OVRE5	OVRE4	OVRE3	OVRE2	OVRE1	OVRE0

- **OVREx: Overrun Error x**

0: No overrun error on the corresponding channel since the last read of AFEC_OVER.

1: There has been an overrun error on the corresponding channel since the last read of AFEC_OVER.

Note: An overrun error does not always mean that the unread data has been replaced by a new valid data. Refer to Section 43.6.12 “Enhanced Resolution Mode and Digital Averaging Function” for details.

43.7.15 AFEC Compare Window Register

Name: AFEC_CWR

Address: 0x400B0050 (0), 0x400B4050 (1)

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the AFEC Write Protection Mode Register.

- **LOWTHRES: Low Threshold**

Low threshold associated to compare settings of AFEC_EMR. For comparisons lower than 16 bits and signed, the sign should be extended up to the bit 15.

- **HIGHTHRES: High Threshold**

High threshold associated to compare settings of AFEC_EMR. For comparisons lower than 16 bits and signed, the sign should be extended up to the bit 15.

43.7.16 AFEC Channel Gain Register

Name: AFEC_CGR

Address: 0x400B0054 (0), 0x400B4054 (1)

Access: Read/Write

31	30	29	28	27	26	25	24
GAIN15		GAIN14		GAIN13		GAIN12	
23	22	21	20	19	18	17	16
GAIN11		GAIN10		GAIN9		GAIN8	
15	14	13	12	11	10	9	8
GAIN7		GAIN6		GAIN5		GAIN4	
7	6	5	4	3	2	1	0
GAIN3		GAIN2		GAIN1		GAIN0	

This register can only be written if the WPEN bit is cleared in the AFEC Write Protection Mode Register.

- **GAINx: Gain for Channel x**

Gain applied on input of Analog Front-End.

GAINx	Gain Applied	
	DIFFx = 0(1)	DIFFx = 1(1)
0	1	0.5
1	1	1
2	2	2
3	4	2

Note: 1. See Section 43.7.18 “AFEC Channel Differential Register” for the description of DIFFx.

43.7.17 AFEC Channel Calibration DC Offset Register

Name: AFEC_CDOR

Address: 0x400B005C (0), 0x400B405C (1)

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
OFF15	OFF14	OFF13	OFF12	OFF11	OFF10	OFF9	OFF8
7	6	5	4	3	2	1	0
OFF7	OFF6	OFF5	OFF4	OFF3	OFF2	OFF1	OFF0

This register can only be written if the WPEN bit is cleared in the AFEC Write Protection Mode Register.

- **OFFx: Offset for Channel x, used in Automatic Calibration Procedure**

0: No offset.

1: Centers the analog signal on $V_{ADVREF}/2$ before the gain scaling. The applied offset is: $(G-1)V_{ADVREF}/2$

where:

G = applied gain (see AFEC_CGR)

Note: When a channel requires calibration, the corresponding OFF bit must be configured to ‘1’ prior to launch of the automatic calibration.

43.7.18 AFEC Channel Differential Register

Name: AFEC_DIFFR

Address: 0x400B0060 (0), 0x400B4060 (1)

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
DIFF15	DIFF14	DIFF13	DIFF12	DIFF11	DIFF10	DIFF9	DIFF8
7	6	5	4	3	2	1	0
DIFF7	DIFF6	DIFF5	DIFF4	DIFF3	DIFF2	DIFF1	DIFF0

This register can only be written if the WPEN bit is cleared in the AFEC Write Protection Mode Register.

- **DIFFx: Differential inputs for channel x**

0: Single-ended mode.

1: Fully-differential mode.

43.7.19 AFEC Channel Selection Register

Name: AFEC_CSELR

Address: 0x400B0064 (0), 0x400B4064 (1)

Access: Read/Write

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

- **CSEL: Channel Selection**

0–15: Selects the channel to be displayed in AFEC_CDR and AFEC_COCR. To be configured with the appropriate channel number.

43.7.20 AFEC Channel Data Register

Name: AFEC_CDR

Address: 0x400B0068 (0), 0x400B4068 (1)

Access: Read-only

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

- **DATA: Converted Data**

Returns the AFE conversion data corresponding to channel CSEL (configured in the AFEC Channel Selection Register).

At the end of a conversion, the converted data is loaded into one of the 16 internal registers (one for each channel) and remains in this internal register until a new conversion is completed on the same channel index. The AFEC_CDR together with AFEC_CSELR allows to multiplex all the internal channel data registers.

The data carried on AFEC_CDR is valid only if AFEC_CHSR.CH*x* bit is set (where *x* = AFEC_CSELR.CSEL field value).

43.7.21 AFEC Channel Offset Compensation Register

Name: AFEC_COCR

Address: 0x400B006C (0), 0x400B406C (1)

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the AFEC Write Protection Mode Register.

- **AOFF: Analog Offset**

Defines the analog offset to be used for channel CSEL (configured in the AFEC Channel Selection Register). This value is used as an input value for the DAC included in the AFE.

Note: The field AOFF must be configured to 2048 (mid scale of the DAC) when there is no offset error to compensate.To compensate for an offset error of n LSB (positive or negative), the field AOFF must be configured to 2048 + n.

43.7.22 AFEC Temperature Sensor Mode Register

Name: AFEC_TEMPMR

Address: 0x400B0070 (0), 0x400B4070 (1)

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the AFEC Write Protection Mode Register.

- **RTCT: Temperature Sensor RTC Trigger Mode**

0: The temperature sensor measure is not triggered by RTC event.

1: The temperature sensor measure is triggered by RTC event (if TRGEN = 1).

- **TEMPCMPMOD: Temperature Comparison Mode**

Value	Name	Description
0	LOW	Generates an event when the converted data is lower than the low threshold of the window.
1	HIGH	Generates an event when the converted data is higher than the high threshold of the window.
2	IN	Generates an event when the converted data is in the comparison window.
3	OUT	Generates an event when the converted data is out of the comparison window.

43.7.23 AFEC Temperature Compare Window Register

Name: AFEC_TEMPCWR

Address: 0x400B0074 (0), 0x400B4074 (1)

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the AFEC Write Protection Mode Register.

- **TLOWTHRES: Temperature Low Threshold**

Low threshold associated to compare settings of the AFEC_TEMPMR. For comparisons less than 16 bits and signed, the sign should be extended up to the bit 15.

- **THIGHTHRES: Temperature High Threshold**

High threshold associated to compare settings of the AFEC_TEMPMR. For comparisons less than 16 bits and signed, the sign should be extended up to the bit 15.

43.7.24 AFEC Analog Control Register

Name: AFEC_ACR

Address: 0x400B0094 (0), 0x400B4094 (1)

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the AFEC Write Protection Mode Register.

- **IBCTL: AFE Bias Current Control**

Adapts performance versus power consumption. (Refer the AFE Characteristics in the section “Electrical Characteristics”.)

43.7.25 AFEC Write Protection Mode Register

Name: AFEC_WPMR

Address: 0x400B00E4 (0), 0x400B40E4 (1)

Access: Read/Write

31	30	29	28	27	26	25	24
WPKEY							

23	22	21	20	19	18	17	16
WPKEY							

15	14	13	12	11	10	9	8
WPKEY							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WPEN

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x414443 (“ADC” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x414443 (“ADC” in ASCII).

See Section 43.6.16 “Register Write Protection” for the list of registers which can be protected.

- **WPKEY: Write Protect KEY**

Value	Name	Description
0x414443	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

43.7.26 AFEC Write Protection Status Register

Name: AFEC_WPSR

Address: 0x400B00E8 (0), 0x400B40E8 (1)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
WPVSRC							

15	14	13	12	11	10	9	8
WPVSRC							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WPVS

- **WPVS: Write Protect Violation Status**

0: No Write Protect Violation has occurred since the last read of the AFEC_WPSR.

1: A Write Protect Violation has occurred since the last read of the AFEC_WPSR. 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 = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

44. Digital-to-Analog Converter Controller (DACC)

44.1 Description

The Digital-to-Analog Converter Controller (DACC) provides up to 2 analog outputs, making it possible for the digital-to-analog conversion to drive up to 2 independent analog lines.

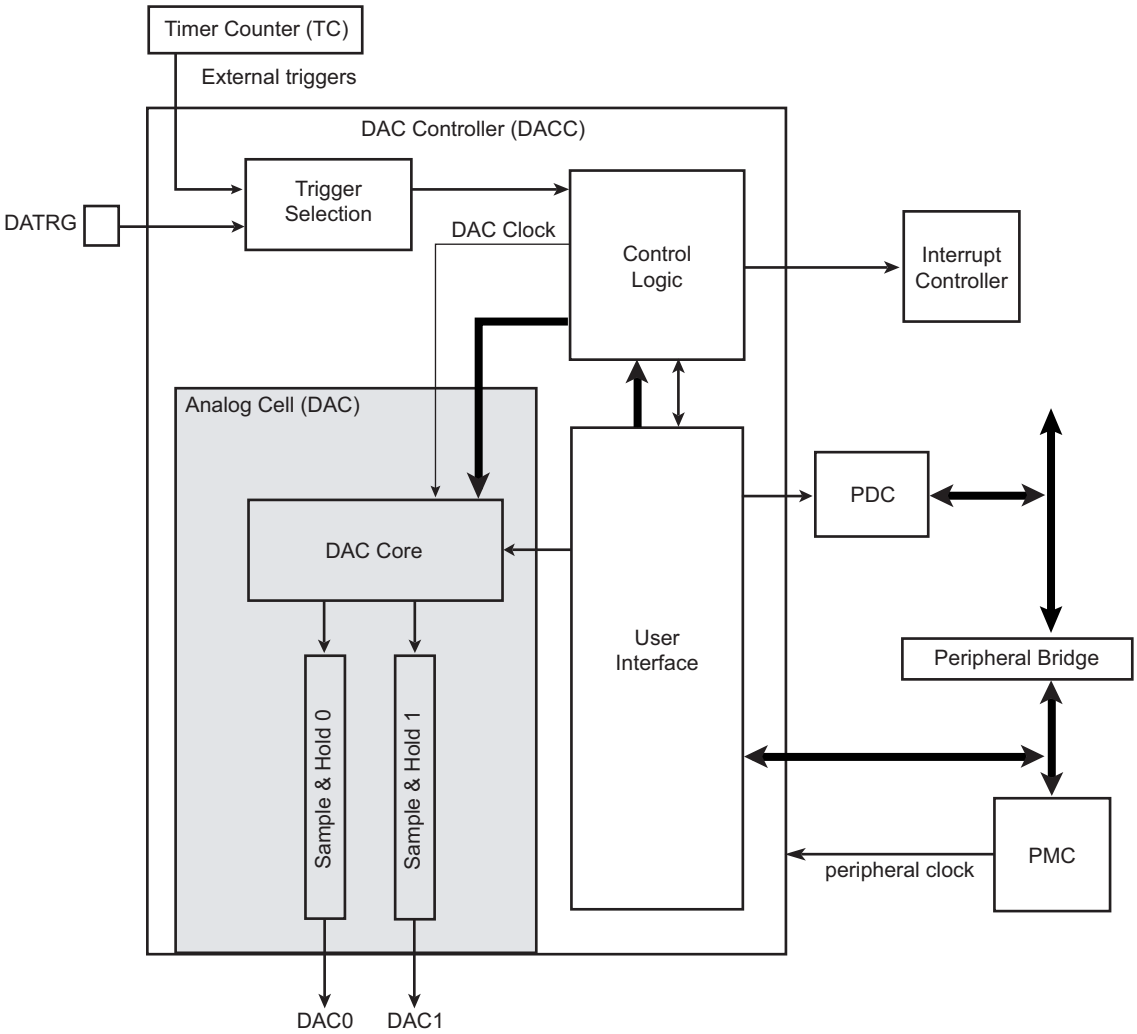
The DACC supports 12-bit resolution. Data to be converted are sent in a common register for all channels. External triggers or free-running mode are configurable.

44.2 Embedded Characteristics

- Up to Two Independent Analog Outputs
- 12-bit Resolution
- Individual Enable and Disable of Each Analog Channel
- Hardware Trigger
  - External Trigger Pins
- PDC Support
- Internal FIFO
- Register Write Protection

44.3 Block Diagram

Figure 44-1. DACC 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 user must first enable the DAC Controller Clock in the Power Management Controller (PMC) before using the DACC.

The DACC becomes active as soon as a conversion is requested and at least one channel is enabled. The DACC is automatically deactivated when no channels are enabled.

For power-saving options, see Section 44.6.6 “DACC Timings”.

44.5.2 Interrupt Sources

The DACC interrupt line is connected to 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	32

44.5.3 Conversion Performances

For performance and electrical characteristics of the DACC, see the product DC Characteristics section of the datasheet.

44.6 Functional Description

44.6.1 Digital-to-Analog Conversion

The DACC uses the peripheral clock divided by either two or four to perform conversions. This clock is named DAC clock. If the peripheral clock frequency is above 100 MHz, the CLKDIV bit must be set in the DACC Mode Register (DACC_MR). Once a conversion starts, the DACC takes 25 clock periods to provide the analog result on the selected analog output.

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 (DACC_ISR) is set.

Reading the DACC_ISR 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 (DACC_CDR). 25 DAC 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 four half-word FIFO is used to handle the data to be converted.

If the TXRDY flag in the DACC_ISR is active, the DACC is ready to accept conversion requests by writing data into the DATA field in the DACC_CDR. Data which cannot be converted immediately is stored in the DACC FIFO.

When the FIFO is full or when the DACC is not ready to accept conversion requests, the TXRDY flag is inactive.

The WORD field of the DACC Mode Register (DACC_MR) allows the user to switch between half-word and word transfers in order to write into the FIFO.

In half-word transfer mode, only the 16 LSBs of DACC_CDR data are processed. Bits DATA[15:0] are stored in the FIFO. Bits DATA[11:0] are used as data. Bits DATA[15:12] are used for channel selection if the TAG field is set in DACC_MR.

In word transfer mode, each time DACC_CDR is written, two data items are stored in the FIFO. The first data item sampled for conversion is DATA[15:0] and the second is DATA[31:16]. Bits DATA[15:12] and DATA[31:28] are used for channel selection if the TAG field is set in DACC_MR.

Warning: Writing in the DACC_CDR while the TXRDY flag is inactive will corrupt FIFO data.

44.6.5 Channel Selection

There are two ways to select the channel to perform data conversion.

- By default, the USER_SEL field of the DACC_MR is used. Data requests are converted to the channel selected with the USER_SEL field.
- Alternatively, the tag mode can be used by setting the TAG field of the DACC_MR to 1. In this mode, the two 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 two bits, DACC_CDR[13:12] are used for channel selection of the first data and the two bits, DACC_CDR[29:28] for channel selection of the second data.

44.6.6 DACC Timings

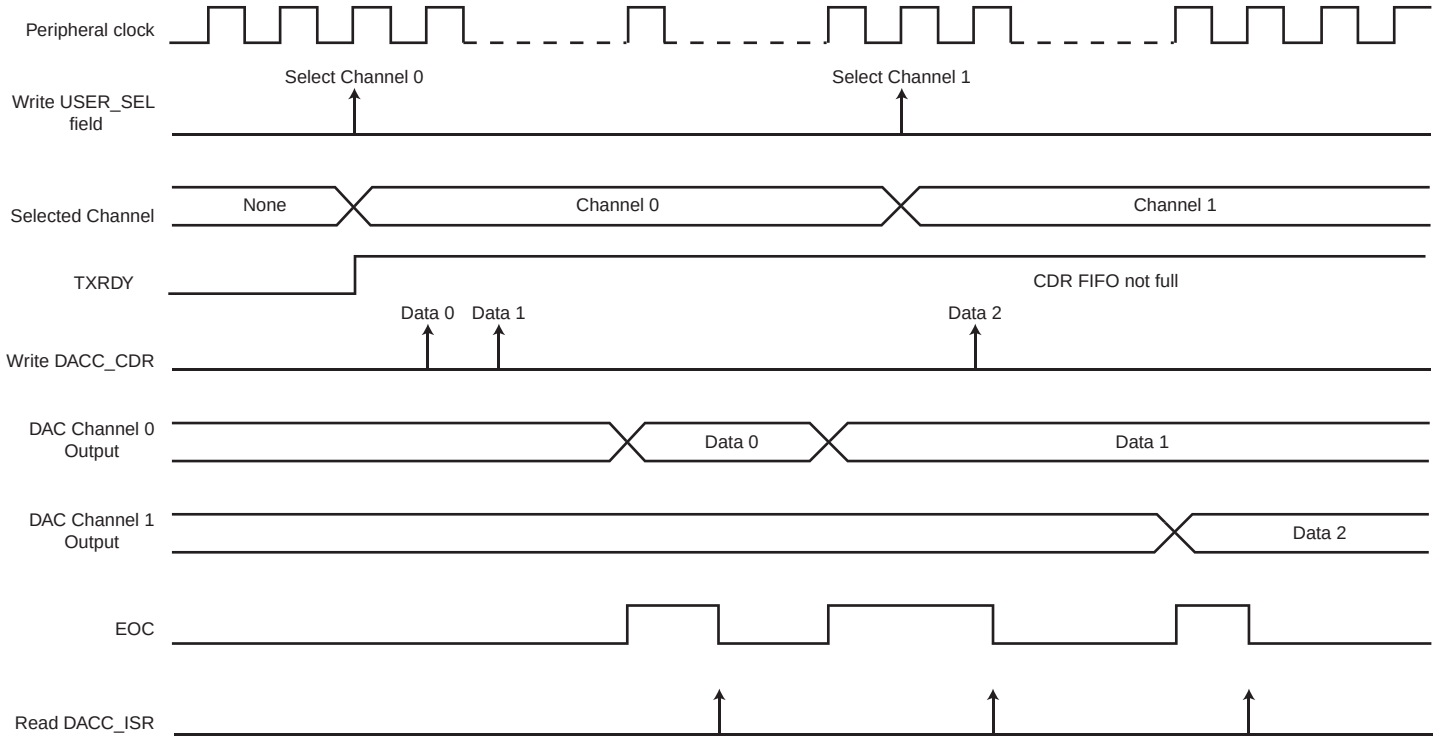
The DACC start-up time must be defined by the user in the STARTUP field of the DACC_MR.

A maximum speed mode is available by setting the MAXS bit in the DACC_MR. In this mode, the DACC no longer waits to sample the end-of-cycle signal coming from the DACC block to start the next conversion. An internal counter is used instead, thus gaining two peripheral clock periods between each consecutive conversion.

Warning: If the maximum speed mode is used, the EOC interrupt of the DACC_IER should not be used as it is two peripheral clock periods late.

The accuracy of the analog voltage resulting from the data conversion process cannot be guaranteed due to leakage. To ensure accuracy, the channel must be refreshed on a regular basis. A value is correctly refreshed if the correct sampling period is selected (see DACC electrical characteristics) and the software or PDC is able to sustain writing to DACC_CDR at the rate imposed by the trigger period.

Figure 44-2. Conversion Sequence



44.6.7 Register Write Protection

To prevent any single software error from corrupting DACC behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the DACC Write Protection Mode Register (DACC_WPMR).

If a write access to a write-protected register is detected, the WPVS flag in the DACC Write Protection Status Register (DACC_WPSR) is set and the field WPVSRC indicates the register in which the write access has been attempted.

The WPVS bit is automatically cleared after reading the DACC_WPSR.

The following registers can be write-protected:

- DACC Mode Register
- DACC Channel Enable Register
- DACC Channel Disable Register
- DACC Analog Current Register

44.7 Digital-to-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
0x08–0x0C	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	–
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
0x34–0x90	Reserved	–	–	–
0x94	Analog Current Register	DACC_ACR	Read/Write	0x00000000
0x98–0xE0	Reserved	–	–	–
0xE4	Write Protection Mode Register	DACC_WPMR	Read/Write	0x00000000
0xE8	Write Protection Status Register	DACC_WPSR	Read-only	0x00000000
0xEC–0xFC	Reserved	–	–	–

44.7.1 DACC Control Register

Name: DACC_CR

Address: 0x400B8000

Access: Write-only

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

- **SWRST: Software Reset**

0: No effect

1: Resets the DACC, simulating a hardware reset

44.7.2 DACC Mode Register

Name: DACC_MR

Address: 0x400B8004

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	STARTUP					
23	22	21	20	19	18	17	16
–	CLKDIV	MAXS	TAG	–	–	USER_SEL	
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	ONE
7	6	5	4	3	2	1	0
–	–	–	WORD	TRGSEL			TRGEN

This register can only be written if the WPEN bit is cleared in the DACC Write Protection 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**

Value	Name	Description
0	TRGSEL0	External trigger
1	TRGSEL1	TIO Output of the Timer Counter Channel 0
2	TRGSEL2	TIO Output of the Timer Counter Channel 1
3	TRGSEL3	TIO Output of the Timer Counter Channel 2
4	TRGSEL4	PWM Event Line 0
5	TRGSEL5	PWM Event Line 1
6	TRGSEL6	Reserved

- **WORD: Word Transfer**

Value	Name	Description
0	HALF	Half-word transfer
1	WORD	Word transfer

- **ONE: Must Be Set to 1**

Bit 8 must always be set to 1 when programming the DACC_MR

- **USER_SEL: User Channel Selection**

Value	Name	Description
0	CHANNEL0	Channel 0
1	CHANNEL1	Channel 1

- **TAG: Tag Selection Mode**

Value	Name	Description
0	DIS	Tag selection mode disabled. Using USER_SEL to select the channel for the conversion.
1	EN	Tag selection mode enabled

- **MAXS: Maximum Speed Mode**

Value	Name	Description
0	NORMAL	Normal mode
1	MAXIMUM	Maximum speed mode enabled

- **CLKDIV: Clock Divider**

Value	Name	Description
0	DIV_2	DAC clock is peripheral clock divided by 2
1	DIV_4	DAC clock is peripheral clock divided by 4 (to be used when peripheral clock frequency is above 100 MHz)

- **STARTUP: Startup Time Selection**

Value	Name	Description
0	0	0 periods of peripheral clock
1	8	8 periods of peripheral clock
2	16	16 periods of peripheral clock
3	24	24 periods of peripheral clock
4	64	64 periods of peripheral clock
5	80	80 periods of peripheral clock
6	96	96 periods of peripheral clock
7	112	112 periods of peripheral clock
8	512	512 periods of peripheral clock
9	576	576 periods of peripheral clock
10	640	640 periods of peripheral clock
11	704	704 periods of peripheral clock
12	768	768 periods of peripheral clock
13	832	832 periods of peripheral clock
14	896	896 periods of peripheral clock
15	960	960 periods of peripheral clock
16	1024	1024 periods of peripheral clock
17	1088	1088 periods of peripheral clock
18	1152	1152 periods of peripheral clock
19	1216	1216 periods of peripheral clock
20	1280	1280 periods of peripheral clock
21	1344	1344 periods of peripheral clock
22	1408	1408 periods of peripheral clock

Value	Name	Description
23	1472	1472 periods of peripheral clock
24	1536	1536 periods of peripheral clock
25	1600	1600 periods of peripheral clock
26	1664	1664 periods of peripheral clock
27	1728	1728 periods of peripheral clock
28	1792	1792 periods of peripheral clock
29	1856	1856 periods of peripheral clock
30	1920	1920 periods of peripheral clock
31	1984	1984 periods of peripheral clock
32	2048	2048 periods of peripheral clock
33	2112	2112 periods of peripheral clock
34	2176	2176 periods of peripheral clock
35	2240	2240 periods of peripheral clock
36	2304	2304 periods of peripheral clock
37	2368	2368 periods of peripheral clock
38	2432	2432 periods of peripheral clock
39	2496	2496 periods of peripheral clock
40	2560	2560 periods of peripheral clock
41	2624	2624 periods of peripheral clock
42	2688	2688 periods of peripheral clock
43	2752	2752 periods of peripheral clock
44	2816	2816 periods of peripheral clock
45	2880	2880 periods of peripheral clock
46	2944	2944 periods of peripheral clock
47	3008	3008 periods of peripheral clock
48	3072	3072 periods of peripheral clock
49	3136	3136 periods of peripheral clock
50	3200	3200 periods of peripheral clock
51	3264	3264 periods of peripheral clock
52	3328	3328 periods of peripheral clock
53	3392	3392 periods of peripheral clock
54	3456	3456 periods of peripheral clock
55	3520	3520 periods of peripheral clock
56	3584	3584 periods of peripheral clock
57	3648	3648 periods of peripheral clock
58	3712	3712 periods of peripheral clock
59	3776	3776 periods of peripheral clock
60	3840	3840 periods of peripheral clock

Value	Name	Description
61	3904	3904 periods of peripheral clock
62	3968	3968 periods of peripheral clock
63	4032	4032 periods of peripheral clock

Note: Refer to the DAC electrical characteristics section in the datasheet for start-up time value.

44.7.3 DACC Channel Enable Register

Name: DACC_CHER

Address: 0x400B8010

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	–	–	CH1	CH0

This register can only be written if the WPEN bit is cleared in the DACC Write Protection 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: 0x400B8014

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	–	–	CH1	CH0

This register can only be written if the WPEN bit is cleared in the DACC Write Protection 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 disabled then re-enabled during a conversion, the associated analog value and the corresponding EOC flags in the DACC_ISR are unpredictable.

44.7.5 DACC Channel Status Register

Name: DACC_CHSR

Address: 0x400B8018

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	–	–	CH1	CH0

- **CHx: Channel x Status**

0: Corresponding channel is disabled

1: Corresponding channel is enabled

44.7.6 DACC Conversion Data Register

Name: DACC_CDR

Address: 0x400B8020

Access: Write-only

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

- **DATA: Data to Convert**

When the WORD bit in DACC_MR is cleared, only DATA[15:0] is used; else DATA[31:0] is used to write two data to be converted.

44.7.7 DACC Interrupt Enable Register

Name: DACC_IER

Address: 0x400B8024

Access: Write-only

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

The following configuration values are valid for all listed bit names of this register:

0: No effect

1: Enables the corresponding interrupt

- **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: 0x400B8028

Access: Write-only

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

The following configuration values are valid for all listed bit names of this register:

0: No effect

1: Disables the corresponding interrupt

- **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: 0x400B802C

Access: Read-only

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

The following configuration values are valid for all listed bit names of this register:

0: Corresponding interrupt is not enabled

1: Corresponding interrupt is enabled

- **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: 0x400B8030

Access: Read-only

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

- **TXRDY: Transmit Ready Interrupt Flag**

0: DACC is not ready to accept new conversion requests.

1: DACC is ready to accept new conversion requests.

- **EOC: End of Conversion Interrupt Flag**

0: No conversion has been performed since the last DACC_ISR read.

1: At least one conversion has been performed since the last DACC_ISR read.

- **ENDTX: End of DMA Interrupt Flag**

0: The Transmit Counter register has not reached 0 since the last write in DACC_TCR or DACC_TNCR.

1: The Transmit Counter register has reached 0 since the last write in DACC _TCR or DACC_TNCR.

- **TXBUFE: Transmit Buffer Empty**

0: The Transmit Counter register has not reached 0 since the last write in DACC_TCR or DACC_TNCR.

1: The Transmit Counter register has reached 0 since the last write in DACC _TCR or DACC_TNCR.

44.7.11 DACC Analog Current Register

Name: DACC_ACR

Address: 0x400B8094

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	IBCTLDACCORE	
7	6	5	4	3	2	1	0
–	–	–	–	IBCTLCH1		IBCTLCH0	

This register can only be written if the WPEN bit is cleared in the DACC Write Protection Mode Register.

- **IBCTLCHx: Analog Output Current Control**

Used to modify the slew rate of the analog output (See the product Electrical Characteristics section for further details.)

- **IBCTLDACCORE: Bias Current Control for DAC Core**

Used to modify performance versus power consumption (See the product Electrical Characteristics section for further details.)

44.7.12 DACC Write Protection Mode Register

Name: DACC_WPMR

Address: 0x400B80E4

Access: Read/Write

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

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x444143 (“DAC” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x444143 (“DAC” in ASCII).

See Section 44.6.7 ”Register Write Protection” for the list of registers that can be write-protected.

- **WPKEY: Write Protection Key**

Value	Name	Description
0x444143	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

44.7.13 DACC Write Protection Status Register

Name: DACC_WPSR

Address: 0x400B80E8

Access: Read-only

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

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of the DACC_WPSR.

1: A write protection violation has occurred since the last read of the DACC_WPSR. 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**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

45. Analog Comparator Controller (ACC)

45.1 Description

The Analog Comparator Controller (ACC) configures the analog comparator and generates an interrupt depending on user settings. The analog comparator embeds two 8-to-1 multiplexers that generate two internal inputs. These inputs are compared, resulting in a compare output. The hysteresis level, edge detection and polarity are configurable.

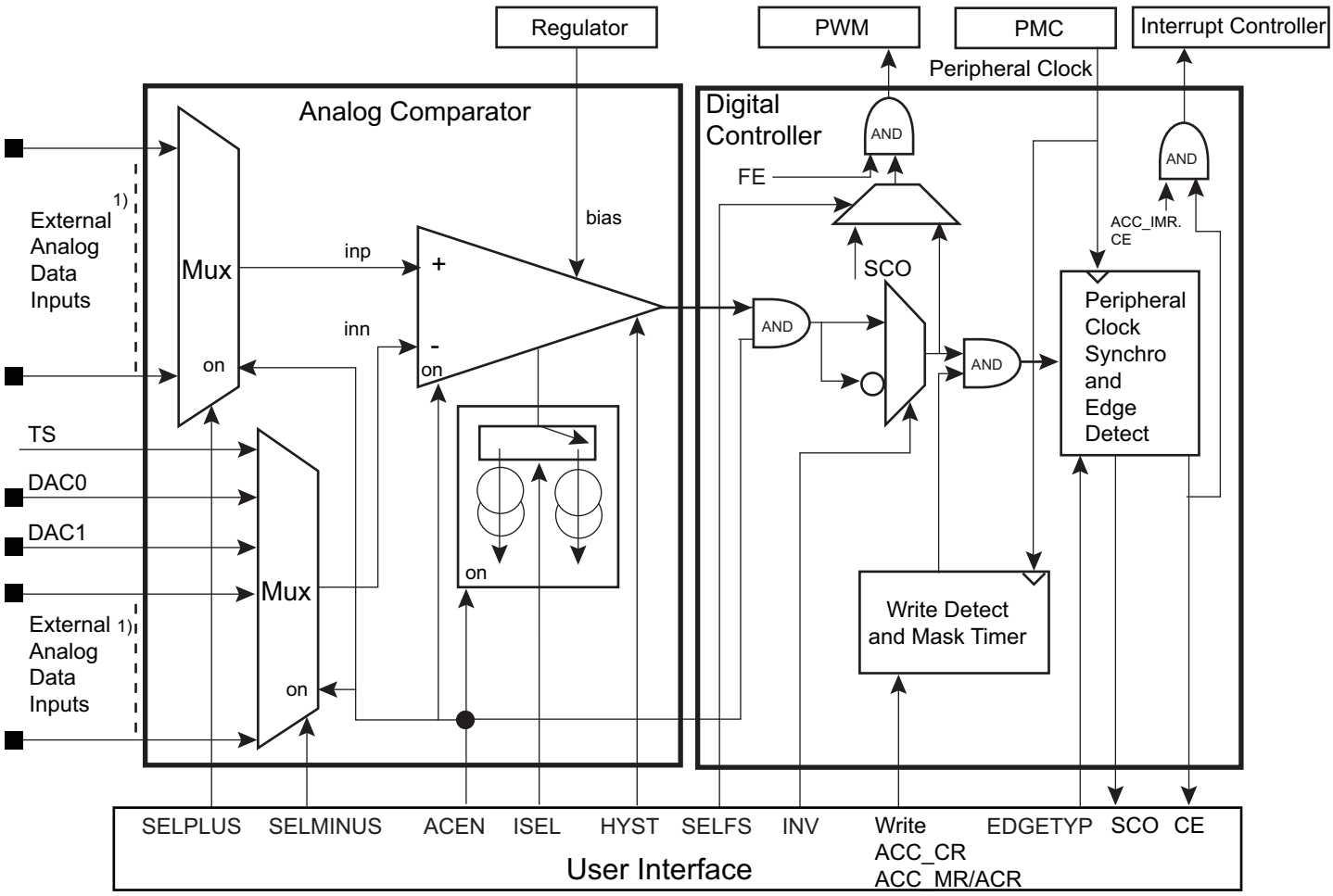
The ACC also generates a compare event which can be used by the Pulse Width Modulator (PWM).

45.2 Embedded Characteristics

- Eight User Analog Inputs Selectable for Comparison
- Four Voltage References Selectable for Comparison: Temperature Sensor (TS), External Voltage Reference, DAC0 and DAC1
- Interrupt Generation
- Compare Event Fault Generation for PWM

45.3 Block Diagram

Figure 45-1. Analog Comparator Controller Block Diagram



45.4 Signal Description

Table 45-1. ACC Signal Description

Pin Name	Description	Type
AFE0_AD[5:0]	External analog data inputs	Input
AFE1_AD[1:0]		
TS	On-chip temperature sensor	Input
ADVREF	AFE and DAC voltage reference	Input
DAC0, DAC1	On-chip DAC inputs	Input

45.5 Product Dependencies

45.5.1 I/O Lines

The analog input pins (AFE0_AD[5:0], AFE1_AD[1:0]) are multiplexed with digital functions (PIO) on the IO line. By writing the SELMINUS and SELPLUS fields in the ACC Mode Register (ACC_MR), the associated IO lines are set to Analog mode.

45.5.2 Power Management

The ACC is clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the ACC clock.

Note that the voltage regulator must be activated to use the analog comparator.

45.5.3 Interrupt Sources

The ACC has an interrupt line connected to the Interrupt Controller (IC). In order to handle interrupts, the Interrupt Controller must be programmed before configuring the ACC.

Table 45-2. Peripheral IDs

Instance	ID
ACC	33

45.5.4 Fault Output

The ACC has the FAULT output connected to the FAULT input of PWM. Please refer to chapter Section 45.6.4 “Fault Mode” and the implementation of the PWM in the product.

45.6 Functional Description

45.6.1 Description

The Analog Comparator Controller (ACC) controls the analog comparator settings and performs post-processing of the analog comparator output.

When the analog comparator settings are modified, the output of the analog cell may be invalid. The ACC masks the output for the invalid period.

A comparison flag is triggered by an event on the output of the analog comparator and an interrupt is generated. The event on the analog comparator output can be selected among falling edge, rising edge or any edge.

The ACC registers are listed in Table 45-3.

45.6.2 Analog Settings

The user can select the input hysteresis and configure two different options, characterized as follows:

- High-speed: shortest propagation delay/highest current consumption
- Low-power: longest propagation delay/lowest current consumption

45.6.3 Output Masking Period

As soon as the analog comparator settings change, the output is invalid for a duration depending on ISEL current.

A masking period is automatically triggered as soon as a write access is performed on the ACC_MR or ACC Analog Control Register (ACC_ACR) (whatever the register data content).

When ISEL = 0, the mask period is $8 \times t_{\text{peripheral clock}}$.

When ISEL = 1, the mask period is $128 \times t_{\text{peripheral clock}}$.

The masking period is reported by reading a negative value (bit 31 set) on the ACC Interrupt Status Register (ACC_ISR).

45.6.4 Fault Mode

In Fault mode, a comparison match event is communicated by the ACC fault output which is directly and internally connected to a PWM fault input.

The source of the fault output can be configured as either a combinational value derived from the analog comparator output or as the peripheral clock resynchronized value (Refer to Figure 45-1 “Analog Comparator Controller Block Diagram”).

45.6.5 Register Write Protection

To prevent any single software error from corrupting ACC behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the ACC Write Protection Mode Register (ACC_WPMR).

If a write access to a write-protected register is detected, the WPVS flag in the ACC Write Protection Status Register (ACC_WPSR) is set and the field WPVSRC indicates the register in which the write access has been attempted.

The WPVS bit is automatically cleared after reading the ACC_WPSR register.

The following registers can be write-protected:

- ACC Mode Register
- ACC Analog Control Register

45.7 Analog Comparator Controller (ACC) User Interface

Table 45-3. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	ACC_CR	Write-only	–
0x04	Mode Register	ACC_MR	Read/Write	0
0x08–0x20	Reserved	–	–	–
0x24	Interrupt Enable Register	ACC_IER	Write-only	–
0x28	Interrupt Disable Register	ACC_IDR	Write-only	–
0x2C	Interrupt Mask Register	ACC_IMR	Read-only	0
0x30	Interrupt Status Register	ACC_ISR	Read-only	0
0x34–0x90	Reserved	–	–	–
0x94	Analog Control Register	ACC_ACR	Read/Write	0
0x98–0xE0	Reserved	–	–	–
0xE4	Write Protection Mode Register	ACC_WPMR	Read/Write	0
0xE8	Write Protection Status Register	ACC_WPSR	Read-only	0
0xEC–0xFC	Reserved	–	–	–

45.7.1 ACC Control Register

Name: ACC_CR

Address: 0x400BC000

Access: Write-only

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

- **SWRST: Software Reset**

0: No effect.

1: Resets the module.

45.7.2 ACC Mode Register

Name: ACC_MR

Address: 0x400BC004

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	FE	SELFS	INV	–	EDGETYP		ACEN
7	6	5	4	3	2	1	0
–	SELPLUS			–	SELMINUS		

This register can only be written if the WPEN bit is cleared in the ACC Write Protection Mode Register.

- **SELMINUS: Selection for Minus Comparator Input**

0..7: Selects the input to apply on analog comparator SELMINUS comparison input.

Value	Name	Description
0	TS	Select TS
1	ADVREF	Select ADVREF
2	DAC0	Select DAC0
3	DAC1	Select DAC1
4	AFE0_AD0	Select AFE0_AD0
5	AFE0_AD1	Select AFE0_AD1
6	AFE0_AD2	Select AFE0_AD2
7	AFE0_AD3	Select AFE0_AD3

- **SELPLUS: Selection For Plus Comparator Input**

0..7: Selects the input to apply on analog comparator SELPLUS comparison input.

Value	Name	Description
0	AFE0_AD0	Select AFE0_AD0
1	AFE0_AD1	Select AFE0_AD1
2	AFE0_AD2	Select AFE0_AD2
3	AFE0_AD3	Select AFE0_AD3
4	AFE0_AD4	Select AFE0_AD4
5	AFE0_AD5	Select AFE0_AD5
6	AFE1_AD0	Select AFE1_AD0
7	AFE1_AD1	Select AFE1_AD1

- **ACEN: Analog Comparator Enable**

0 (DIS): Analog comparator disabled.

1 (EN): Analog comparator enabled.

- **EDGETYP: Edge Type**

Value	Name	Description
0	RISING	Only rising edge of comparator output
1	FALLING	Falling edge of comparator output
2	ANY	Any edge of comparator output

- **INV: Invert Comparator Output**

0 (DIS): Analog comparator output is directly processed.

1 (EN): Analog comparator output is inverted prior to being processed.

- **SELFS: Selection Of Fault Source**

0 (CE): The CE flag is used to drive the FAULT output.

1 (OUTPUT): The output of the analog comparator flag is used to drive the FAULT output.

- **FE: Fault Enable**

0 (DIS): The FAULT output is tied to 0.

1 (EN): The FAULT output is driven by the signal defined by SELFS.

45.7.3 ACC Interrupt Enable Register

Name: ACC_IER

Address: 0x400BC024

Access: Write-only

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

- **CE: Comparison Edge**

0: No effect.

1: Enables the interrupt when the selected edge (defined by EDGETYP) occurs.

45.7.4 ACC Interrupt Disable Register

Name: ACC_IDR

Address: 0x400BC028

Access: Write-only

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

- **CE: Comparison Edge**

0: No effect.

1: Disables the interrupt when the selected edge (defined by EDGETYP) occurs.

45.7.5 ACC Interrupt Mask Register

Name: ACC_IMR

Address: 0x400BC02C

Access: Read-only

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

- **CE: Comparison Edge**

0: The interrupt is disabled.

1: The interrupt is enabled.

45.7.6 ACC Interrupt Status Register

Name: ACC_ISR

Address: 0x400BC030

Access: Read-only

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

- **CE: Comparison Edge (cleared on read)**

0: No edge occurred (defined by EDGETYP) on analog comparator output since the last read of ACC_ISR.

1: A selected edge (defined by EDGETYP) on analog comparator output occurred since the last read of ACC_ISR.

- **SCO: Synchronized Comparator Output**

Returns an image of the analog comparator output after being pre-processed (refer to Figure 45-1).

If INV = 0

SCO = 0 if inn > inp

SCO = 1 if inp > inn

If INV = 1

SCO = 1 if inn > inp

SCO = 0 if inp > inn

- **MASK: Flag Mask**

0: The CE flag and SCO value are valid.

1: The CE flag and SCO value are invalid.

45.7.7 ACC Analog Control Register

Name: ACC_ACR

Address: 0x400BC094

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in ACC Write Protection Mode Register.

- **ISEL: Current Selection**

Refer to ACC electrical characteristics section.

0 (LOPW): Low-power option.

1 (HISP): High-speed option.

- **HYST: Hysteresis Selection**

0 to 3: Refer to ACC electrical characteristics section.

45.7.8 ACC Write Protection Mode Register

Name: ACC_WPMR

Address: 0x400BC0E4

Access: Read/Write

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

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x414343 (“ACC” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x414343 (“ACC” in ASCII).

See Section 45.6.5 “Register Write Protection” for the list of registers that can be write-protected.

- **WPKEY: Write Protection Key**

Value	Name	Description
0x414343	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

45.7.9 ACC Write Protection Status Register

Name: ACC_WPSR

Address: 0x400BC0E8

Access: Read-only

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

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of ACC_WPSR.

1: A write protection violation (WPEN = 1) has occurred since the last read of ACC_WPSR.

46. SAM4E Electrical Characteristics

46.1 Absolute Maximum Ratings

Table 46-1. Absolute Maximum Ratings*

Operating Temperature (Industrial).....-40°C to +105°C
Storage Temperature.....-60°C to +150°C
Voltage on Input Pins with Respect to Ground.....-0.3V to +4.0V
Maximum Operating Voltage (V_{DDCORE}).....1.32V
Maximum Operating Voltage (V_{DDIO}).....4.0V
Total DC Output Current on all I/O lines
144-ball LFBGA.....150 mA
144-lead LQFP.....150 mA
100-ball TFBGA.....150 mA
100-lead LQFP.....150 mA

*NOTICE: Stresses beyond those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. **Exposure to absolute maximum rating conditions for extended periods may affect device reliability.**

46.2 DC Characteristics

The following characteristics are applicable to the operating temperature range T_A = -40°C to 105°C, unless otherwise specified.

Table 46-2. DC Characteristics

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
V_{DDCORE}	DC Supply Core	—		1.08	1.20	1.32	V
V_{DDIO}	DC Supply I/Os	(2)		1.62	3.3	3.6	V
V_{DDPLL}	PLLA and Main Oscillator Supply	—		1.08	—	1.32	V
V_{IL}	Low-level Input Voltage	PA0–PA31, PB0–PB14, PC0–PC31, PD0–PD31, PE0–PE5, NRST		-0.3	—	MIN [0.8V, 0.3 × V_{DDIO}]	V
V_{IH}	High-level Input Voltage	PA0–PA31, PB0–PB14, PC0–PC31, PD0–PD31, PE0–PE5, NRST		MIN [2.0V, 0.7 × V_{DDIO}]	—	V_{DDIO} +0.3V	V
V_{OH}	High-level Output Voltage	PA0–PA31, PB0–PB9, PB12–PB14, PC0–PC31, PD0–PD31, PE0–PE5 I_{OL} = I_{OL} Max		V_{DDIO} - 0.4V	—	—	V
		VDDIO[3.0–3.60 V] PB10–PB11		V_{DDIO} - 0.15V	—	—	
V_{OL}	Low-level Output Voltage	PA0–PA31, PB0–PB9, PB12–PB14, PC0–PC31, PD0–PD31, PE0–PE5 I_{OH} = I_{OH} Max		—	—	0.4	V
		VDDIO[3.0–3.60 V] PB10–PB11		—	—	0.15	
V_{hys}	Hysteresis Voltage	PA0–PA31, PB0–PB9, PB12–PB14, PC0–PC31 (Hysteresis mode enabled)		150	—	—	mV
I_{OH}	Source Current	VDDIO[1.62–3.60 V]; V_{OH} = V_{DDIO} - 0.4	PA14 (SPCK), PA29 (MCCK) pins	—	—	-4	mA
			PA[5–8], PA[12–13], PA[26–28], PA[30–31], PB[8–9], PB[14], PD[0–1], PD[3–17] pins	—	—	-4	
			PA[0–3]	—	—	-2	
			NRST	—	—	-2	
			Other pins(1)	—	—	-2	
		VDDIO[3.0–3.60 V]	PB[10–11]	—	—	-30	
I_{OL}	Sink Current	VDDIO[1.62–3.60 V]; V_{OL} = 0.4V	PA14 (SPCK), PA29 (MCCK) pins	—	—	4	mA
			PA[5–8], PA[12–13], PA[26–28], PA[30–31], PB[8–9], PB[14], PD[0–1], PD[3–17] pins	—	—	4	
			PA [0–3]	—	—	2	
			NRST	—	—	2	
			Other pins(1)	—	—	2	
		VDDIO[3.0–3.60 V]	PB[10–11]	—	—	30	

Table 46-2. DC Characteristics (Continued)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
I_{IL}	Input Low	Pull-up OFF	-1	—	1	µA
		Pull-up ON	10	—	50	
I_{IH}	Input High	Pull-down OFF	-1	—	1	
		Pull-down ON	10	—	50	
R_{PULLUP}	Pull-up Resistor	PA0–PA31, PB0–PB14, PC0–PC31, PD0–PD31, PE0–PE5, NRST	70	100	130	kΩ
$R_{PULLDOWN}$	Pull-down Resistor	PA0–PA31, PB0–PB14, PC0–PC3, PD0–PD31, PE0–PE51, NRST	70	100	130	kΩ
R_{ODT}	On-die Series Termination Resistor	PA4–PA31, PB0–PB9, PB12–PB14, PC0–PC31, PD0–PD31, PE0–PE5	—	36	—	Ω
		PA0–PA3	—	18	—	

Notes: 1. PA[4], PA[9–11], PA[15–25], PB[0–7], PB[12–13], PC[0–31], PD[2], PD[18–31], PE[0–5]

2. Refer to Section 5.2.2 “VDDIO Versus VDDIN”

Table 46-3. 1.2V Voltage Regulator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{DDIN}	DC Input Voltage Range	(4)	1.6	3.3	3.6	V
V_{DDOUT}	DC Output Voltage	Normal Mode	—	1.2	—	V
		Standby Mode	—	0	—	
$V_{O(accuracy)}$	Output Voltage Accuracy	I_{LOAD} = 0.8 mA to 80 mA (after trimming)	-4	—	4	%
I_{LOAD}	Maximum DC Output Current	V_{DDIN} > 1.8V	—	—	120	mA
		V_{DDIN} ≤ 1.8V	—	—	70	
$I_{LOAD-START}$	Maximum Peak Current during startup	(3)	—	—	400	mA
$V_{DROPOUT}$	Dropout Voltage	V_{DDIN} = 1.6V I_{LOAD} = 70 mA	—	400		mV
V_{LINE}	Line Regulation	V_{DDIN} from 2.7 to 3.6 V I_{LOAD} max	—	10	30	mV
$V_{LINE-TR}$	Transient Line Regulation	V_{DDIN} from 2.7 to 3.6 V I_{LOAD} Max $t_r = t_f$ = 5 µs CD_{OUT} = 4.7 µF	—	50	150	mV
V_{LOAD}	Load Regulation	V_{DDIN} ≥ 1.8 V I_{LOAD} = 10% to 90% max	—	25	60	mV
$V_{LOAD-TR}$	Transient Load Regulation	V_{DDIN} 1.8 V I_{LOAD} = 10% to 90% max $t_r = t_f$ = 5 µs CD_{OUT} = 4.7 µF	—	45	210	mV
I_Q	Quiescent Current	Normal Mode, @ I_{LOAD} = 0 mA	—	5.5	—	µA
		Normal Mode, @ I_{LOAD} = 120 mA	—	350	—	
		Standby Mode	—	0.06	—	
CD_{IN}	Input Decoupling Capacitor	(1)	—	4.7	—	µF
CD_{OUT}	Output Decoupling Capacitor	(2)	1.85	2.2	5.9	µF
		ESR	0.1	—	10	Ω
t_{on}	Turn on Time	CD_{OUT} = 2.2 µF V_{DDOUT} reaches 1.2V (± 3%)	—	300	—	µs
t_{off}	Turn off Time	CD_{OUT} = 2.2 µF V_{DDIN} 1.8V	—	—	9.5	ms

Notes:
1. A 4.7 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 2.2 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 0.1Ω to 10Ω. Solid tantalum and multilayer ceramic capacitors are all suitable as output capacitor. A 100 nF bypass capacitor between VDDOUT and the closest GND pin of the device helps decrease output noise and improves the load transient response.
3. Defined as the current needed to charge external bypass/decoupling capacitor network.
4. See Section 5.2.2 “VDDIO Versus VDDIN”

Table 46-4. Core Power Supply Brownout Detector Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{T-}	Supply Falling Threshold[1]	—	0.98	1.0	1.04	V
V_{hys}	Hysteresis Voltage	—	—	—	110	mV
V_{T+}	Supply Rising Threshold	—	0.8	1.0	1.08	V
t_{RST}	Reset Period	V_{DDIO} rising from 0 to 1.2V ± 10%	90	—	320	µs
I_{DDON}	Current Consumption on VDDCORE	Brownout Detector enabled	—	—	24	µA
I_{DDOFF}		Brownout Detector disabled	—	—	2	µA
I_{DD33ON}	Current Consumption on VDDIO	Brownout Detector enabled	—	—	24	µA
$I_{DD33OFF}$		Brownout Detector disabled	—	—	2	µA
t_{d-}	V_{T-} Detection Propagation Time	VDDCORE = V_{T+} to (V_{T-} - 100mV)	—	200	300	ns
t_{START}	Startup Time	From disabled state to enabled state	—	—	300	µs

Note: 1. The product is guaranteed to be functional at V_{T-}.

Figure 46-1. Core Brownout Output Waveform

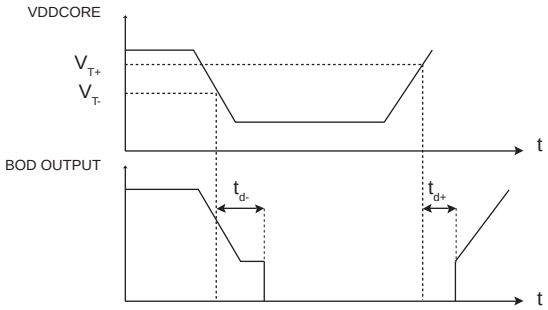


Table 46-5. VDDIO Supply Monitor

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_T	Supply Monitor Threshold	16 selectable steps	1.6	—	3.4	V
$V_{T(accuracy)}$	Threshold Level Accuracy	-40/+105°C	-2.5	—	+2.5	%
V_{hys}	Hysteresis Voltage		—	20	30	mV
I_{DDON}	Current Consumption	Enabled	—	23	40	µA
I_{DDOFF}		Disabled	—	0.02	2	µA
t_{START}	Startup Time	From disabled state to enabled state	—	—	300	µs

Table 46-6. Threshold Selection

Digital Code	Threshold min (V)	Threshold typ (V)	Threshold max (V)
0000	1.56	1.6	1.64
0001	1.68	1.72	1.76
0010	1.79	1.84	1.89
0011	1.91	1.96	2.01
0100	2.03	2.08	2.13
0101	2.15	2.2	2.23
0110	2.26	2.32	2.38
0111	2.38	2.44	2.50
1000	2.50	2.56	2.62
1001	2.61	2.68	2.75
1010	2.73	2.8	2.87
1011	2.85	2.92	2.99
1100	2.96	3.04	3.12
1101	3.08	3.16	3.24
1110	3.20	3.28	3.36
1111	3.32	3.4	3.49

Figure 46-2. VDDIO Supply Monitor

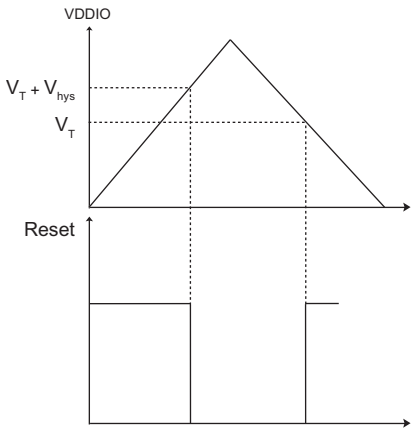


Table 46-7. Zero-Power-On Reset Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{T+}	Threshold Voltage Rising	At Startup	1.45	1.53	1.59	V
V_{T-}	ThreshoLd Voltage Falling	—	1.35	1.45	1.55	V
t_{RST}	Reset Period	—	100	340	580	µs

Figure 46-3. Zero-Power-On Reset Characteristics

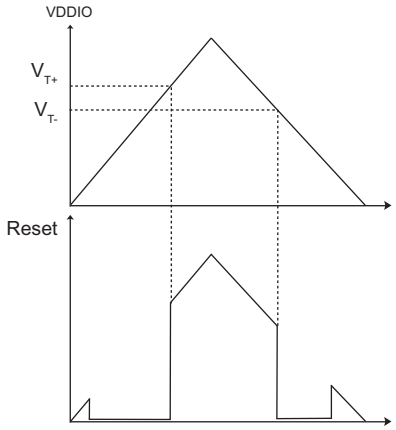


Table 46-8. DC Flash Characteristics

Symbol	Parameter	Conditions	Typ	Max	Unit
I_{CC}	Active current	Random 128-bit Read: Maximum Read Frequency onto VDDCORE = 1.2V @ 25°C	16	25	mA
		Random 64-bit Read: Maximum Read Frequency onto VDDCORE = 1.2V @ 25°C	10	18	mA
		Program onto VDDCORE = 1.2V @ 25°C	3	5	mA

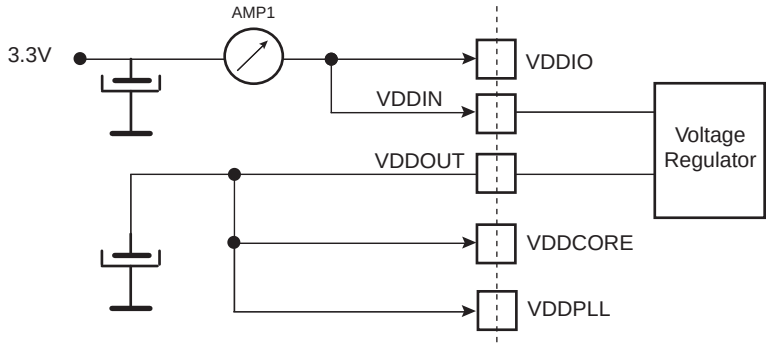
46.3 Power Consumption

- Power consumption of the device depending on the different Low-Power mode Capabilities (Backup, Wait, Sleep) and Active mode.
- Power consumption on power supply in different modes: Backup, Wait, Sleep and Active.
- Power consumption by peripheral: calculated as the difference in current measurement after having enabled then disabled the corresponding clock.
- All power consumption values are based on characterization. Note that these values are not covered by test limits in production.

46.3.1 Backup Mode Current Consumption

The backup mode configuration and measurements are defined as follows.

Figure 46-4. Measurement Setup



46.3.1.1 Configuration A: Embedded Slow Clock RC Oscillator Enabled

- Supply Monitor on VDDIO is disabled
- RTC is running
- RTT is enabled on 1Hz mode
- BOD is disabled
- One WKUPx enabled
- Current measurement on AMP1 (see Figure 46-4)

46.3.1.2 Configuration B: 32.768 kHz Crystal Oscillator Enabled

- Supply Monitor on VDDIO is disabled
- RTC is running
- RTT enabled on 1Hz mode
- BOD disabled
- One WKUPx enabled
- Current measurement on AMP1 (see Figure 46-4)

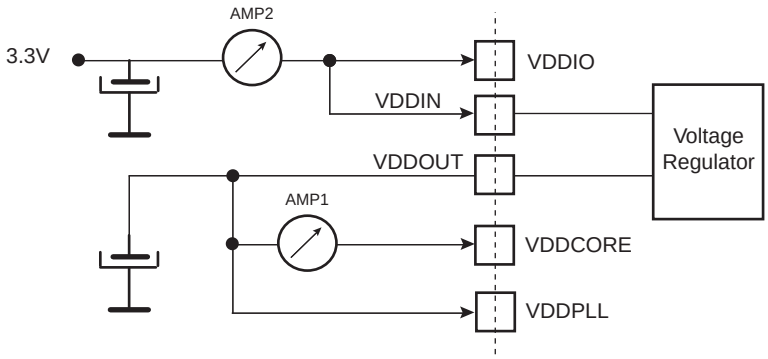
Table 46-9. Typical Power Consumption for Backup Mode Configuration A and B

BACKUP Total Consumption	Typical value				
	@25°C		@85°C	@105°C	
Conditions	(AMP1) Configuration A	(AMP1) Configuration B	(AMP1) Configuration A	(AMP1) Configuration A	Unit
V_{DDIO} = 3.6V	2.0	1.9	7.2	15.4	µA
V_{DDIO} = 3.3V	1.7	1.6	6.9	12.0	
V_{DDIO} = 3.0V	1.5	1.5	6.2	11.2	
V_{DDIO} = 2.5V	1.3	1.2	5.5	9.8	
V_{DDIO} = 1.8V	1	0.9	4.6	8.3	

46.3.2 Sleep and Wait Mode Current Consumption

The Wait mode and Sleep mode configuration and measurements are defined below.

Figure 46-5. Measurement Setup for Sleep Mode



46.3.2.1 Sleep Mode

- Core Clock OFF
- $V_{DDIO} = V_{DDIN} = 3.3V$
- Master Clock (MCK) running at various frequencies with PLLA or the fast RC oscillator
- Fast start-up through pins WKUP0–15
- Current measurement as shown in Figure 46-5
- All peripheral clocks deactivated
- $T_A = 25°C$

Table 46-10 gives current consumption in typical conditions.

Figure 46-6. Current Consumption in Sleep Mode (AMP1) versus Master Clock Ranges (refer to Table 46-10)

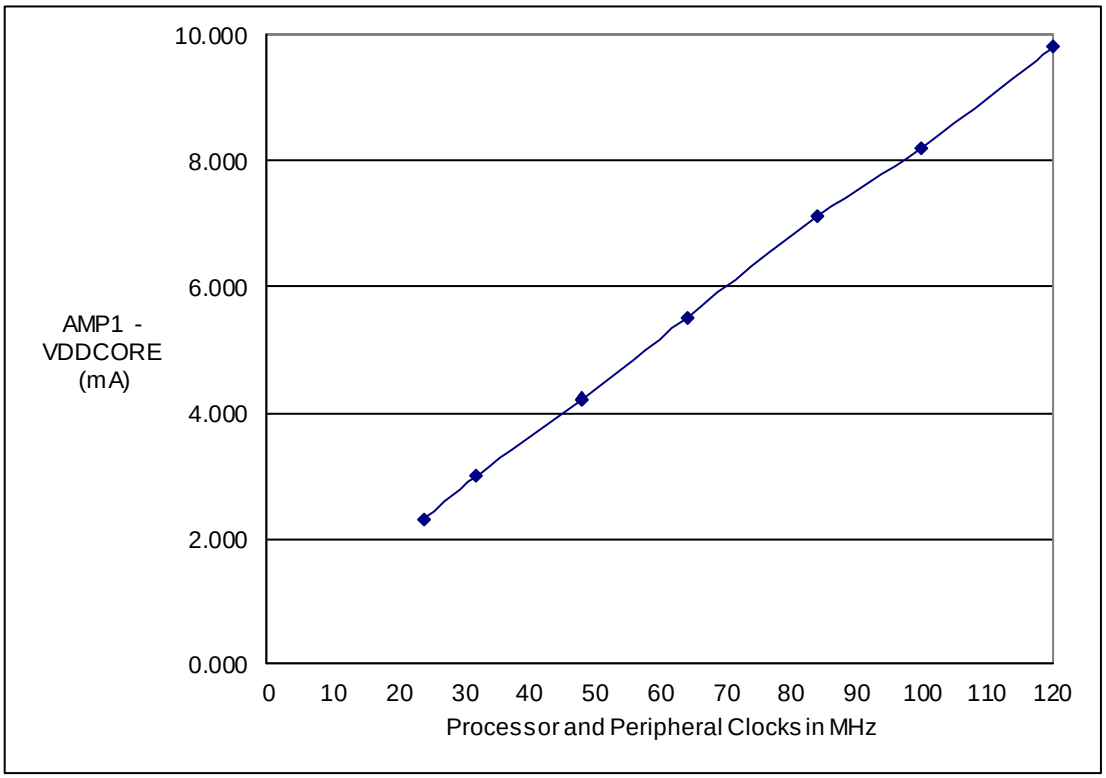


Table 46-10. Sleep Mode Current Consumption versus Master Clock (MCK) Variation with PLLA

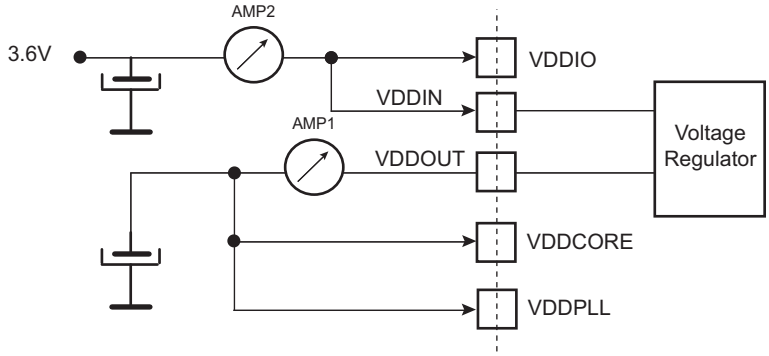
Core Clock/MCK (MHz)	VDDCORE Consumption (AMP1)	Total Consumption (AMP2)	Unit
120	9.8	11.4	mA
100	8.2	9.5	
84	7.1	9.4	
64	5.5	7.2	
48	4.2	5.5	
32	3.0	4.7	
24	2.3	3.5	

Table 46-11. Sleep Mode Current Consumption versus Master Clock (MCK) Variation with Fast RC

Core Clock/MCK (MHz)	VDDCORE Consumption (AMP1)	Total Consumption (AMP2)	Unit
12	1.11	1.14	mA
8	0.77	0.8	
4	0.45	0.48	
2	0.3	0.33	
1	0.22	0.25	
0.5	0.18	0.21	
0.25	0.16	0.19	

46.3.2.2 Wait Mode

Figure 46-7. Measurement Setup for Wait Mode



- $V_{DDIO} = V_{DDIN} = 3.6V$
- Core Clock and Master Clock stopped
- Current measurement as shown in the above figure
- All peripheral clocks deactivated
- BOD disabled
- RTT enabled

Table 46-12 gives current consumption in typical conditions.

Table 46-12. Typical Current Consumption in Wait Mode (1)

Conditions	Typical Value				Unit
	@25°C		@85°C	@105°C	
	VDDOUT Consumption (AMP1)	Total Consumption (AMP2)	Total Consumption (AMP2)	Total Consumption (AMP2)	
See Figure 46-7 on page 1364 There is no activity on the I/Os of the device; Flash in Standby mode.	43	56	550	1200	µA
See Figure 46-7 on page 1364 There is no activity on the I/Os of the device; Flash in Deep Power Down Mode.	39	47	496	980	

Note: 1. Value from characterization, not tested in production.

46.3.3 Active Mode Power Consumption

The Active Mode configuration and measurements are defined as follows:

- $V_{DDIO} = V_{DDIN} = 3.3V$
- $V_{DDCORE} = 1.2V$ (internal voltage regulator used)
- $T_A = 25°C$
- Application running from Flash Memory with 128-bit access mode
- All peripheral clocks are deactivated.
- Master Clock (MCK) running at various frequencies with PLLA or the fast RC oscillator
- Current measurement on AMP1 (VDDCORE) and total current on AMP2

Figure 46-8. Active Mode Measurement Setup

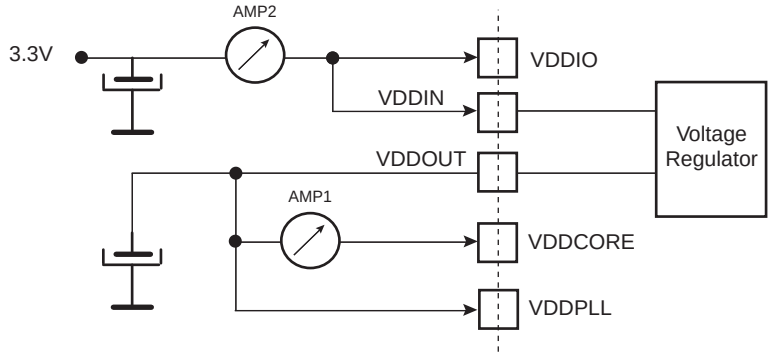


Table 46-13 on page 1365 and Figure 46-14 on page 1366 give the Active Mode Current Consumption in typical conditions.

- VDDCORE at 1.2V
- T_A = 25°C

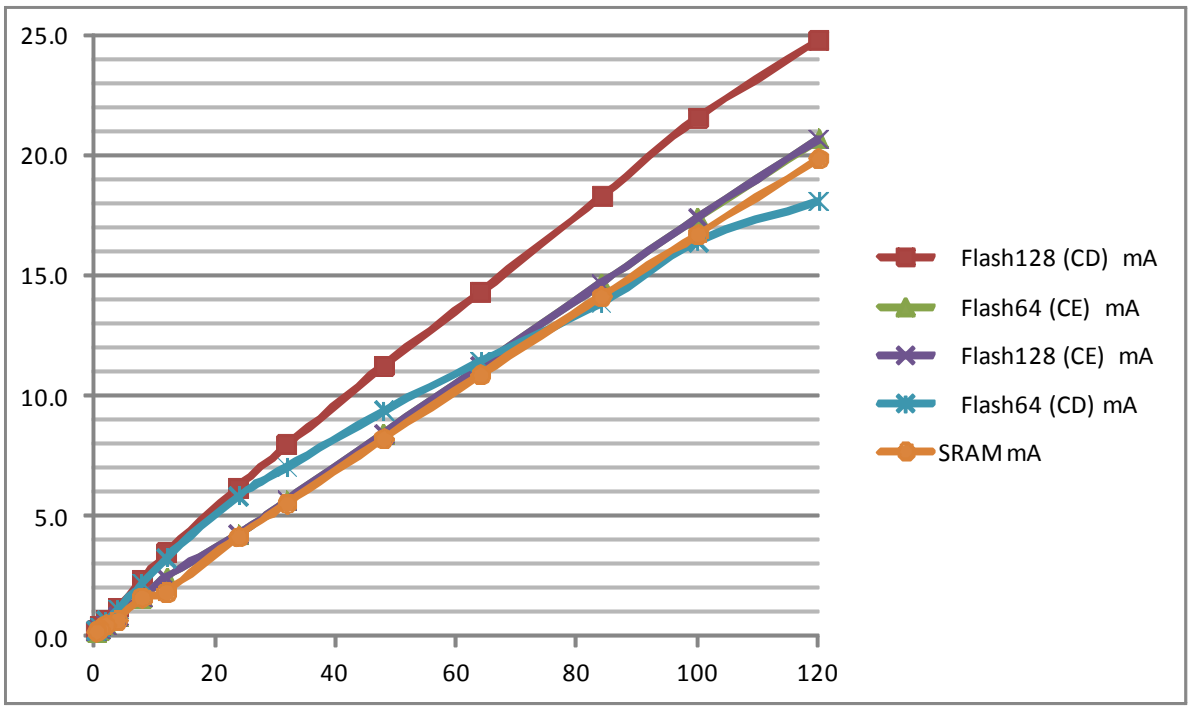
46.3.3.1 SAM4E Active Power Consumption

Table 46-13. Active Power Consumption with VDDCORE @ 1.2V running from Embedded Memory (IDDCORE - AMP1)

Core Clock (MHz)	Core Mark					Unit
	Cache Enable (CE)		Cache Disable (CD)		SRAM	
	128-bit Flash access(1)	64-bit Flash access(1)	128-bit Flash access(1)	64-bit Flash access(1)		
120	21.1	21.0	25.5	19.0	17.9	mA
100	18.1	18.1	22.5	17.2	15.0	
84	15.5	15.5	20.0	16.1	12.86	
64	11.9	11.9	16.4	13.6	9.9	
48	9.0	9.0	12.7	11.7	7.5	
32	6.2	6.2	9.1	8.9	5.2	
24	4.6	4.6	7	6.8	3.9	
12	2.5	2.4	4.1	3.8	2.2	
8	1.9	1.8	2.9	2.8	1.6	
4	1.2	1.1	1.7	1.7	1.0	
2	0.81	0.79	1	1	0.75	
1	0.46	0.46	0.6	0.6	0.44	
0.5	0.38	0.38	0.47	0.44	0.36	

Note: 1. Flash Wait State (FWS) in EEFC_FMR is adjusted depending on core frequency.

Figure 46-9. Active Power Consumption with VDDCORE @ 1.2V



- VDDCORE at 1.2V
- T_A = 25°C

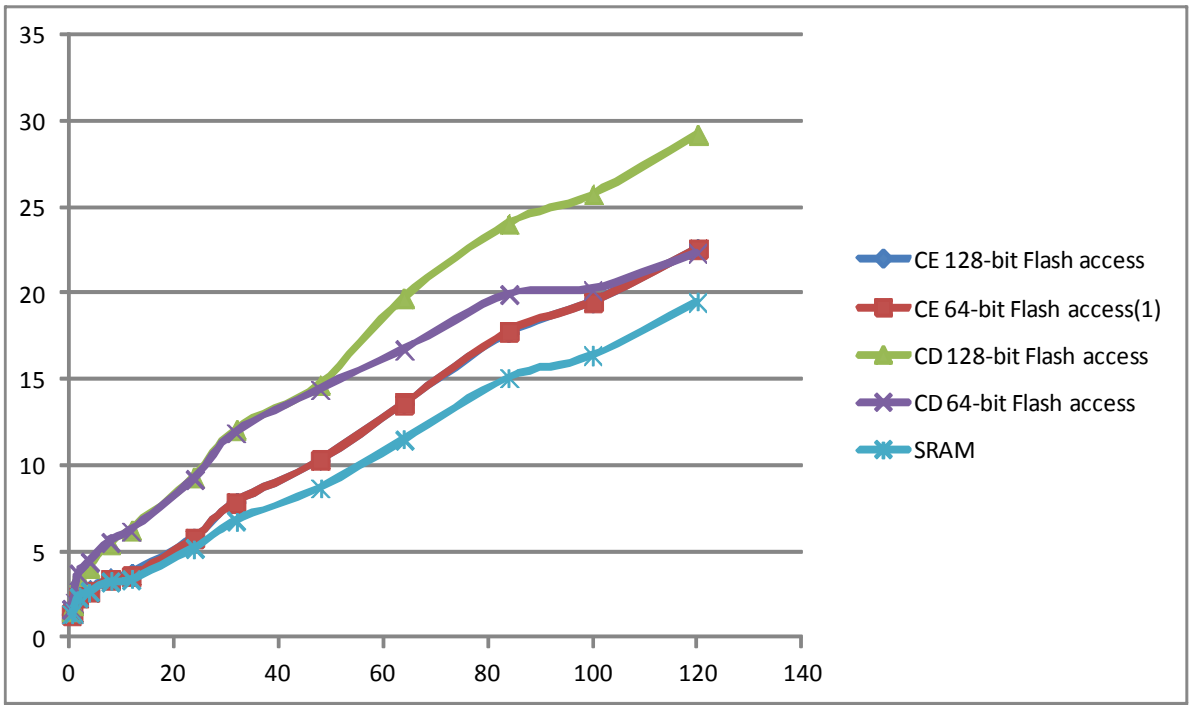
46.3.3.2 SAM4E Active Total Power Consumption

Table 46-14. Active Total Power Consumption with VDDCORE @ 1.2V running from Embedded Memory (IDDIO + IDDIN - AMP2)

Core Clock (MHz)	CoreMark					Unit
	Cache Enable (CE)		Cache Disable (CD)		SRAM	
	128-bit Flash Access(1)	64-bit Flash Access(1)	128-bit Flash Access(1)	64-bit Flash Access(1)		
120	22.6	22.6	29.2	22.3	19.5	mA
100	19.5	19.5	25.7	20.2	16.4	
84	17.7	17.8	24.0	19.9	15.1	
64	13.6	13.6	19.7	16.7	11.5	
48	10.3	10.3	14.7	14.4	8.7	
32	7.9	7.9	12.1	11.9	6.8	
24	5.8	5.8	9.3	9.2	5.2	
12	3.8	3.6	6.3	6.2	3.4	
8	3.5	3.4	5.5	5.6	3.3	
4	2.8	2.7	4.1	4.4	2.7	
2	2.5	2.4	3.6	3.7	2.4	
1	1.5	1.5	1.9	2.1	1.5	
0.5	1.4	1.4	1.6	1.7	1.4	

Note: 1. Flash Wait State (FWS) in EEFC_FMR adjusted depending on Core Frequency

Figure 46-10. Active Total Power Consumption with VDDCORE @ 1.2V



46.3.4 Peripheral Power Consumption in Active Mode

Table 46-15. Power Consumption on VDDCORE (VDDIO = 3.3V, VDDCORE = 1.08V, T_A = 25°C)

Peripheral	Consumption (Typ)	Unit
PIO Controller A (PIOA)	5.23	µA/MHz
PIO Controller B (PIOB)	1.44	
PIO Controller C (PIOC)	4.02	
PIO Controller D (PIOD)	3.17	
PIO Controller E (PIOE)	0.86	
UART	4.50	
USART	6.5	
PWM	11.00	
TWI	4.70	
SPI	4.42	
Timer Counter (TCx)	3.7	
AFEC	5.15	
DACC	3.0	
ACC	0.28	
HSMCI	6.43	
CAN	6.5	
SMC	2.77	
UDP	5.11	
GMAC	44.2	
AES	2.39	
DMAC	7.21	

46.4 Oscillator Characteristics

46.4.1 32 kHz RC Oscillator Characteristics

Table 46-16. 32 kHz RC Oscillator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{OSC}	RC Oscillator Frequency	—	20	32	44	kHz
—	Frequency Supply Dependency	—	-3	—	3	%/V
—	Frequency Temperature Dependency	Over temperature range (-40 to 105 °C) versus T_A 25°C	-7	—	7	%
Duty	Duty Cycle	—	45	50	55	%
t_{START}	Startup Time	—	—	—	100	µs
I_{DDON}	Current Consumption	After startup time T_A range = -40 to 105 °C Typical consumption at 2.2V supply and T_A 25°C	—	540	860	nA

46.4.2 4/8/12 MHz RC Oscillators Characteristics

Table 46-17. 4/8/12 MHz RC Oscillators Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{OSC}	RC Oscillator Frequency Range	(1)	4	—	12	MHz
ACC_4	4 MHz Total Accuracy	-40°C < T_A < +105°C 4 MHz output selected (1)(2)	—	—	±30	%
ACC_8	8 MHz Total Accuracy	-40°C < T_A < +105°C 8 MHz output selected (1)(2)	—	—	±30	%
		-40°C < T_A < +105°C 8 MHz output selected (1)(3)	—	—	±5	
ACC_{12}	12 MHz Total Accuracy	-40°C < T_A < +105°C 12 MHz output selected (1)(2)	—	—	±30	%
		-40°C < T_A < +105°C 12 MHz output selected (1)(3)	—	—	±5	
—	Frequency deviation versus trimming code	8 MHz 12 MHz	— —	47 64	— —	kHz/trimming code
Duty	Duty Cycle	—	45	50	55	%
t_{START}	Startup Time	—	—	—	10	µs
I_{DDON}	Active Current Consumption(2)	4 MHz 8 MHz 12 MHz	— — —	50 65 82	75 95 118	µA

Notes: 1. Frequency range can be configured in the Supply Controller Registers
2. Not trimmed from factory
3. After Trimming from factory

The 4/8/12 MHz Fast RC oscillator is calibrated in production. This calibration can be read through the Get CALIB Bit command (see the EEFC section) and the frequency can be trimmed by software through the PMC.

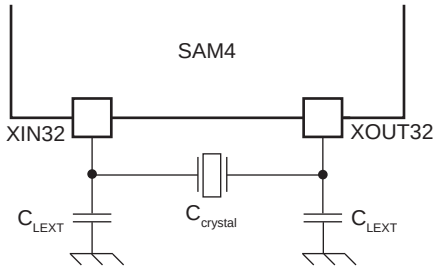
46.4.3 32.768 kHz Crystal Oscillator Characteristics

Table 46-18. 32.768 kHz Crystal Oscillator Characteristics

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
f_{OSC}	Operating Frequency	Normal mode with crystal		—	—	32.768	kHz
$V_{rip(VDDIO)}$	Supply Ripple Voltage (on VDDIO)	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	—	550	1150	nA
			$C_{crystal}$ = 6 pF	—	380	980	
		R_S < 100 kΩ (1)	$C_{crystal}$ = 12.5 pF	—	820	1600	
			$C_{crystal}$ = 6 pF	—	530	1350	
P_{ON}	Drive Level	—		—	—	0.1	µW
R_f	Internal Resistor	Between XIN32 and XOUT32		—	10	—	MΩ
$C_{crystal}$	Allowed Crystal Capacitance Load	From crystal specification		6	—	12.5	pF
C_{para}	Internal Parasitic Capacitance	—		0.6	0.7	0.8	pF

Note: 1. R_S is the series resistor.

Figure 46-11. 32.768 kHz Crystal Oscillator Schematics



$C_{LEXT} = 2 \times (C_{crystal} - C_{para} - C_{PCB})$

where:

C_{PCB} is the capacitance of the printed circuit board (PCB) track layout from the crystal to the SAM4 pin.

46.4.4 32.768 kHz Crystal Characteristics

Table 46-19. Crystal Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
ESR	Equivalent Series Resistor (R_S)	Crystal @ 32.768 kHz	—	50	100	kΩ
C_m	Motional Capacitance	Crystal @ 32.768 kHz	0.6	—	3	fF
C_{SHUNT}	Shunt Capacitance	Crystal @ 32.768 kHz	0.6	—	2	pF

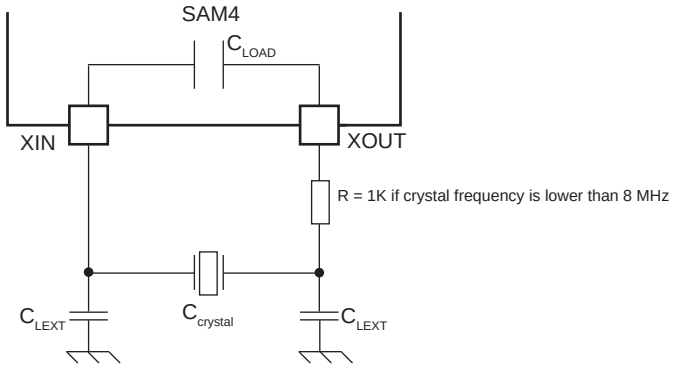
46.4.5 3 to 20 MHz Crystal Oscillator Characteristics

Table 46-20. 3 to 20 MHz Crystal Oscillator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{OSC}	Operating Frequency	Normal mode with crystal	3	16	20	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	—	—	14.5	ms
		8 MHz, C_{SHUNT} = 7 pF	—	—	4	
		16 MHz, C_{SHUNT} = 7 pF with C_m = 8 fF	—	—	1.4	
		16 MHz, C_{SHUNT} = 7 pF with C_m = 1.6 fF	—	—	2.5	
		20 MHz, C_{SHUNT} = 7 pF	—	—	1	
I_{DDON}	Current consumption (on VDDIO)	3 MHz(2)	—	230	350	µA
		8 MHz(3)	—	300	400	
		16 MHz(4)	—	390	470	
		20 MHz(5)	—	450	560	
P_{ON}	Drive level	3 MHz	—	—	15	µW
		8 MHz	—	—	30	
		16 MHz, 20 MHz	—	—	50	
R_f	Internal Resistance	Between XIN and XOUT	—	0.5	—	MΩ
$C_{crystal}$	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	10.5	pF

Notes:
1. R_S is the series resistor
2. R_S = 100–200 Ω; C_{SHUNT} = 2.0–2.5 pF; C_m = 2–1.5 fF (typ, worst case) using 1 kΩ serial resistor on XOUT.
3. R_S = 50–100 Ω; C_{SHUNT} = 2.0–2.5 pF; C_m = 4–3 fF (typ, worst case).
4. R_S = 25–50 Ω; C_{SHUNT} = 2.5–3.0 pF; C_m = 7–5 fF (typ, worst case).
5. R_S = 20–50 Ω; C_{SHUNT} = 3.2–4.0 pF; C_m = 10–8 fF (typ, worst case).

Figure 46-12. 3 to 20 MHz Crystal Oscillator Schematics



$C_{LEXT} = 2 \times (C_{crystal} - C_{LOAD} - C_{PCB})$

where:

C_{PCB} is the capacitance of the printed circuit board (PCB) track layout from the crystal to the SAM4 pin.

46.4.6 3 to 20 MHz Crystal Characteristics

Table 46-21. Crystal Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
ESR	Equivalent Series Resistor (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

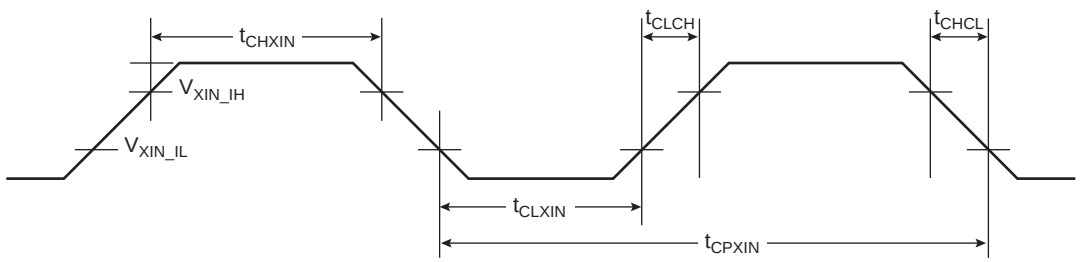
46.4.7 3 to 20 MHz XIN Clock Input Characteristics in Bypass Mode

Table 46-22. XIN Clock Electrical Characteristics (In Bypass Mode)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$1/(t_{CPXIN})$	XIN Clock Frequency	(1)	—	—	50	MHz
t_{CPXIN}	XIN Clock Period	(1)	20	—	—	ns
t_{CHXIN}	XIN Clock High Half-period	(1)	8	—	—	ns
t_{CLXIN}	XIN Clock Low Half-period	(1)	8	—	—	ns
t_{CLCH}	Rise Time	(1)	2.2	—	—	ns
t_{CHCL}	Fall Time	(1)	2.2	—	—	ns
V_{XIN_IL}	V_{XIN} Low-level Input Voltage	(1)	-0.3	—	MIN $[0.8V, 0.3 \times V_{DDIO}]$	V
V_{XIN_IH}	V_{XIN} High-level Input Voltage	(1)	MIN $[2.0V, 0.7 \times V_{DDIO}]$	—	V_{DDIO} + 0.3V	V
$C_{para(standby)}$	Internal Parasitic Capacitance During Standby	(1)	—	5.5	6.3	pF
$R_{para(standby)}$	Internal Parasitic Resistance During Standby	(1)	—	300	—	Ω

Note: 1. These characteristics apply only when the 3–20 MHz crystal oscillator is in Bypass mode.

Figure 46-13. XIN Clock Timing



46.4.8 Crystal Oscillator Design Considerations Information

46.4.8.1 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 SAM4E 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. The 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.

46.4.8.2 Printed Circuit Board (PCB)

SAM4E oscillators are low-power oscillators requiring particular attention when designing PCB systems.

46.5 PLLA Characteristics

Table 46-23. Supply Voltage Phase Lock Loop Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{DDPLLR}	Supply Voltage Range		1.08	1.2	1.32	V
$V_{rip(VDDPLL)}$	Allowable Voltage Ripple	RMS value 10 kHz to 10 MHz	—	—	20	mV
		RMS value > 10 MHz	—	—	10	

Table 46-24. PLLA Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{IN}	Input Frequency	—	3	—	32	MHz
f_{OUT}	Output Frequency	—	80	—	240	MHz
I_{PLL}	Current Consumption	Active mode @ 80 MHz @ 1.2V	—	0.94	1.2	mA
		Active mode @ 96 MHz @ 1.2V	—	1.2	1.5	
		Active mode @ 160 MHz @ 1.2V	—	2.1	2.5	
		Active Mode @ 240 MHz @ 1.2V	—	3.34	4	
t_s	Settling Time	—	—	60	150	µs

46.6 USB Transceiver Characteristics

46.6.1 Typical Connection

For typical connection please refer to Section 41. “USB Device Port (UDP)”.

46.6.2 USB Electrical Characteristics

Table 46-25. USB Electrical Characteristics

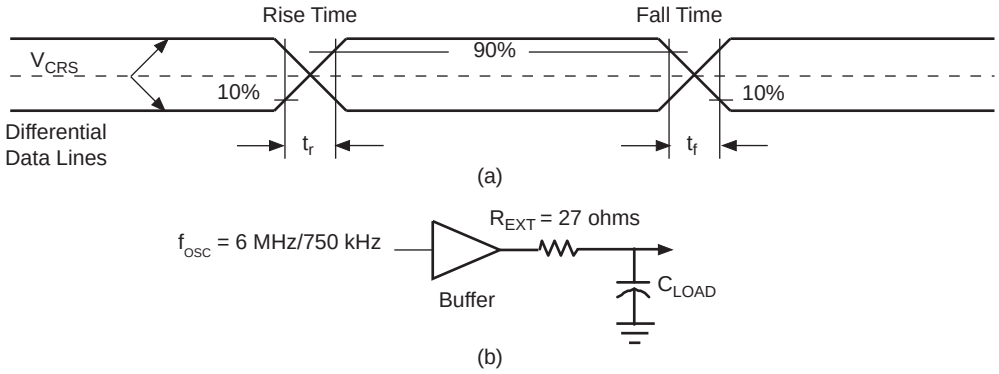
Symbol	Parameter	Conditions	Min	Typ	Max	Unit
Input Levels						
V_{IL}	Low Level	—	—	—	0.8	V
V_{IH}	High Level	—	2.0	—	—	V
V_{DI}	Differential Input Sensitivity	|(D+) - (D-)|	0.2	—	—	V
V_{DICR}	Differential Input Common Mode Range	—	0.8	—	2.5	V
C_i	Transceiver capacitance	Capacitance to ground on each line	—	—	9.18	pF
I_{lkg}	Hi-Z State Data Line Leakage	$0V < V_I < 3.3V$	-10	—	+10	µA
R_{EXT}	Recommended External USB Series Resistor	In series with each USB pin with ±5%	—	27	—	Ω
Output Levels						
V_{OL}	Low Level Output	Measured with R_L of 1.425 kΩ tied to 3.6V	0.0	—	0.3	V
V_{OH}	High Level Output	Measured with R_L of 14.25 kΩ tied to GND	2.8	—	3.6	V
V_{CRS}	Output Signal Crossover Voltage	Measure conditions described in Figure 46-14 “USB Data Signal Rise and Fall Times”	1.3	—	2.0	V
Consumption						
I_{VDDIO}	Current Consumption (on VDDIO)	Transceiver enabled in input mode DDP = 1 and DDM = 0	—	105	200	µA
$I_{VDDCORE}$	Current Consumption (on VDDCORE)		—	80	150	µA
Pull-up Resistor						
R_{PUI}	Bus Pull-up Resistance on Upstream Port (idle bus)	—	0.900	—	1.575	kΩ
R_{PUA}	Bus Pull-up Resistance on Upstream Port (upstream port receiving)	—	1.425	—	3.090	kΩ

46.6.3 Switching Characteristics

Table 46-26. In Full Speed

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
t_r	Transition Rise Time	C_{LOAD} = 50 pF	4	—	20	ns
t_f	Transition Fall Time	C_{LOAD} = 50 pF	4	—	20	ns
t_{rfm}	Rise/Fall time Matching	—	90	—	111.11	%

Figure 46-14. USB Data Signal Rise and Fall Times

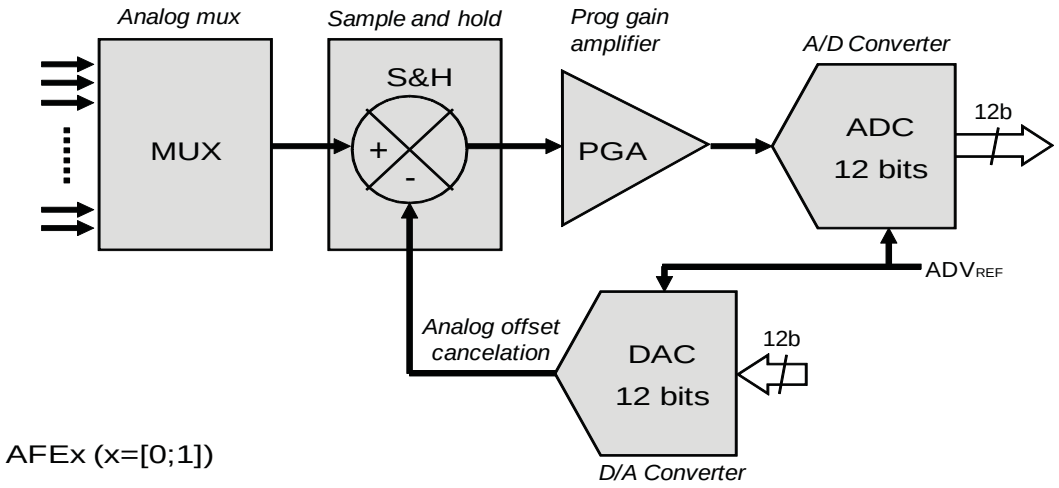


46.7 12-bit AFE (Analog Front End) Characteristics

Electrical data are in accordance with the following standard conditions unless otherwise specified:

- Operating temperature range from -40 to 105 °C
- Min and max data are defined as three times the standard deviation of the manufacturing process

Figure 46-15. 12-bit AFE (Analog Front End) Diagram



46.7.1 ADC Power Supply

Table 46-27. Analog Power Supply Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{DDIN}	Supply Voltage Range	Full operational	2.4	—	3.6	V
		(1)	2	—	2.4	
I_{VDDIN}	Analog Current Consumption	ADC Sleep Mode(2)	—	4	8	µA
		ADC Fast Wake-up Mode(3)		1.8	3	mA
		ADC Normal Mode		3.8	6	mA
$I_{VDDcore}$	Digital Current Consumption	ADC Sleep Mode (all off)(2)	—	—	0.1	µA
		ADC Normal Mode		0.2	0.4	mA

Notes: 1. See Section "Low Voltage Supply".
2. In Sleep mode the ADC core, sample and hold, and internal reference operational amplifier are off.
3. In Fast Wake-up mode, only the ADC core is off.

46.7.1.1 ADC Bias Current

All current consumption is performed when the field IBCTL in the AFEC Control Register (AFEC_ACR) is set to 01.

IBCTL controls the ADC biasing current, with the nominal setting IBCTL = 01.

IBCTL = 01 is the default configuration suitable for a sampling frequency of up to 1 MHz. If the sampling frequency is below 500 kHz, IBCTL = 00 can also be used to reduce the current consumption.

Table 46-28. ADC Bias Current Adjustment

IBCTL = 00	IBCTL = 01	IBCTL = 10	IBCTL = 11
Typ-22%	Typ	Reserved	Reserved

46.7.2 External Reference Voltage

V_{ADVREF} is an external reference voltage applied on the pin ADVREF. The quality of the reference voltage V_{ADVREF} is critical to the performance of the ADC. A DC variation of the reference voltage V_{ADVREF} is converted to a gain error by the ADC. The noise generated by V_{ADVREF} is converted by the ADC to count noise.

Table 46-29. ADVREF Electrical Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{ADVREF}	ADVREF Voltage Range	Full operational	2.4	—	3.6	V
		(1)	2	—	2.4	
V_n	Input Voltage Noise(2)	Gain = 0.5, DIFF(3) mode	—	—	1100	µVrms
		Gain = 1, SE(4) and DIFF(3)	—	—	550	
		Gain = 2, SE(4) and DIFF(3)	—	—	274	
		Gain = 4, SE(4) mode	—	—	137	
R_{ADVREF}	ADVREF Input DC Impedance	ADC+DAC reference resistor bridge(5)	2.4	3	10	kΩ
I_{ADVREF}	ADVREF Current (ADV_{REF} + DAC Current)	ADVREF = 3V	—	1	1.5	mA

Notes:
1. See Section "Low Voltage Supply".
2. Over a bandwidth from 20 Hz to 20 MHz.
3. DIFF is Differential mode.
4. SE is Single-ended mode.
5. When the ADC is in Sleep mode, the ADVREF impedance has a minimum of 10 MΩ.

46.7.3 ADC Timings

Table 46-30. ADC Timing Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{ADC}	Clock Frequency		1	20	22	MHz
t_{CP_ADC}	Clock Period		45	50	1000	ns
f_S	Sampling Frequency		0.05	1	1.1	MHz
t_{START}	ADC Startup time	Sleep mode to Normal mode	—	16	32	µs
		Fast Wake-up mode to Normal mode	—	4	8	
t_{CONV}	Conversion Time	Number of ADC clock pulses to perform a conversion	—	20	—	t_{CP_ADC}
t_{CAL}	Calibration time		200	—	—	ns

46.7.4 ADC Transfer Function

The first operation of the ADC is a sampling function relative to a common mode voltage. The common mode voltage (V_{CM}) is equal to $V_{ADVREF}/2$ when the bits OFFx = 1, in Differential and in Single-ended mode. When the bits OFFx = 0, sampling is done versus $V_{ADVREF}/4$ for gain = 2, and $V_{ADVREF}/8$ for gain = 4, in Single-ended mode only.

The code in AFEC_CDR is a 12-bit positive integer. The internal DAC is set for the code 2047.

46.7.4.1 Differential Mode

A differential input voltage $V_I = V_{I+} - V_{I-}$ can be applied between two selected differential pins, e.g., AD0 and AD1. The ideal code Ci is calculated by using the following formula and rounding the result to the nearest positive integer.

$$Ci = \frac{4096}{V_{ADVREF}} \times V_I \times Gain + 2047$$

Table 46-31 is a computation example for the above formula, where V_{ADVREF} = 3V.

Table 46-31. Input Voltage Values in Differential Mode

Ci	Gain = 0.5	Gain = 1	Gain = 2
0	-3	-1.5	-0.75
2047	0	0	0
4095	3	1.5	0.75

46.7.4.2 Single-ended Mode

A single input voltage V_I can be applied to selected pins, e.g., AD0 or AD1. The ideal code Ci is calculated by using the following formula and rounding the result to the nearest positive integer.

The single-ended ideal code conversion formula for OFFx = 1 is:

$$Ci = \frac{4096}{V_{ADVREF}} \times \left(V_I - \frac{V_{ADVREF}}{2}\right) \times Gain + 2047$$

Table 46-32 is a computation example for the above formula, where V_{ADVREF} = 3V.

Table 46-32. Input Voltage Values in Single-ended Mode, OFFx = 1

Ci	Gain = 1	Gain = 2	Gain = 4
0	0	0.75	1.125
2047	1.5	1.5	1.5
4095	3	2.25	1.875

The single-ended ideal code conversion formula for OFFx = 0 is:

$$Ci = V_I \times Gain \times \frac{4096}{V_{ADVREF}} - 1$$

Table 46-33 is a computation example for the above formula, where V_{ADVREF} = 3V.

Table 46-33. Input Voltage Values in Single-ended Mode, OFFx = 0

Ci	Gain = 1	Gain = 2	Gain = 4
0	0	0	0
2047	1.5	0.75	0.375
4095	3	1.5	0.75

46.7.4.3 Example of LSB Computation

The LSB is relative to the analog scale V_{ADVREF}.

The term LSB expresses the quantization step in volts, also used for one ADC code variation.

- Single-ended (SE) (ex: V_{ADVREF} = 3.0V)
  - Gain = 1, LSB = (3.0V / 4096) = 732 µV
  - Gain = 2, LSB = (1.5V / 4096) = 366 µV
  - Gain = 4, LSB = (750 mV / 4096) = 183 µV
- Differential (DIFF) (ex: V_{ADVREF} = 3.0V)
  - Gain = 0.5, LSB = (6.0V / 4096) = 1465 µV
  - Gain = 1, LSB = (3.0V / 4096) = 732 µV
  - Gain = 2, LSB = (1.5V / 4096) = 366 µV

46.7.5 ADC Electrical Characteristics

The gain error depends on the gain value and the OFFx bit. The data are given with and without autocorrection at T_A 27°C. The data include the ADC performances as the PGA and ADC core cannot be separated. The temperature and voltage dependency are given as separate parameters.

Table 46-34. Voltage and Temperature Dependencies

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
α_G	Gain Temperature dependency	-40 to 105 °C	—	—	5	ppm/°C
α_{GV}	Gain Supply dependency	V_{DDIN}	—	—	0.025	%/V
α_O	Offset Temperature dependency	-40 to 105 °C	—	—	5	ppm/°C
α_{OV}	Offset Supply dependency	V_{DDIN}	—	—	0.025	%/V

46.7.5.1 Gain and Offset Errors

For:

- a given gain error: E_G (%)
- a given ideal code (Ci)
- a given offset error: E_O (LSB)

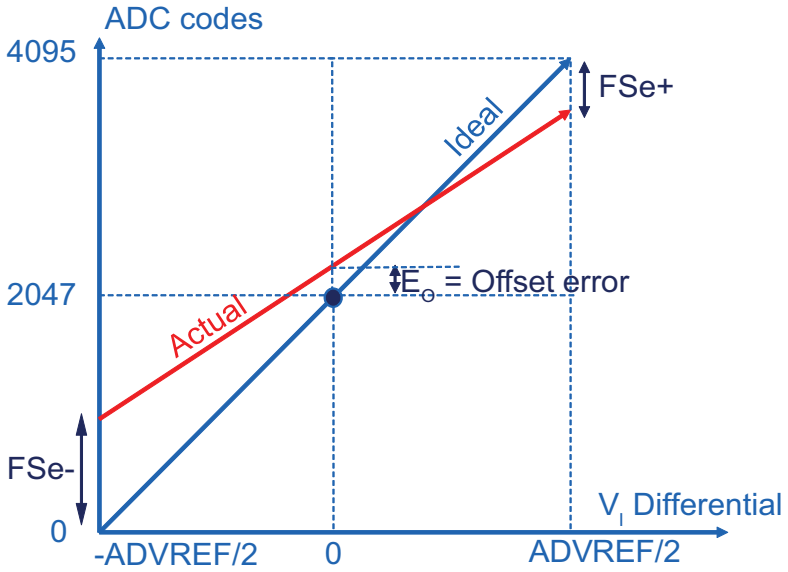
the actual code (Ca) is calculated using the following formula:

$$Ca = \left(1 + \frac{E_G}{100}\right) \times (Ci - 2047) + 2047 + E_O$$

Differential Mode

In differential mode, the offset is defined when the differential input voltage is zero.

Figure 46-16. Gain and Offset Errors in Differential Mode



where:

- FSe = (FSe+) - (FSe-) is for full-scale error, unit is LSB code
- Offset error E_O is the offset error measured for V_I = 0V
- Gain error E_G = 100 × FSe / 4096, unit in %

The error values in Table 46-35 and Table 46-36 include the sample and hold error as well as the PGA gain error.

Table 46-35. Differential Gain Error E_G

Gain Mode	0.5		1		2	
Auto Correction	**No**	**Yes**	**No**	**Yes**	**No**	**Yes**
Average Gain Error (%)	-0.107	0.005	0.444	0.112	0.713	0.005
Standard Deviation (%)	0.410	0.210	0.405	0.229	0.400	0.317
Gain Min Value (%)	-1.338	-0.625	-0.771	-0.576	-0.488	-0.947
Gain Max Value (%)	1.123	0.635	1.660	0.801	1.914	0.957

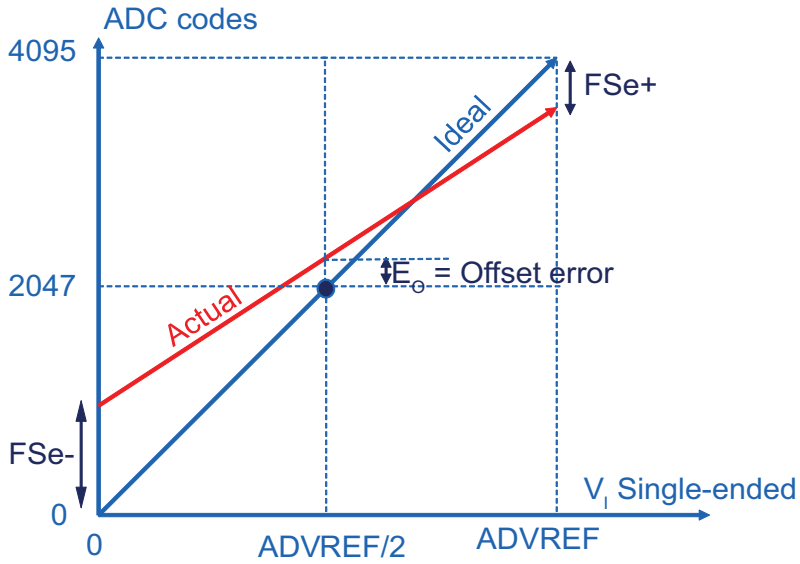
Table 46-36. Differential Output Offset Error E_O

Gain	0.5	1	2
Average Offset Error (LSB)	-1.2	-1.2	-0.6
Standard Deviation (LSB)	0.3	0.4	0.4
Offset Min value (LSB)	-2.1	-2.4	-1.8
Offset Max value (LSB)	-0.3	0	0.6

Single-ended Mode

Figure 46-17 illustrates the ADC output code relative to an input voltage V_I between 0V (Ground) and V_{ADVREF}. The ADC is configured in Single-ended mode by connecting internally the negative differential input to $V_{ADVREF}/2$. As the ADC continues to work internally in Differential mode, the offset is measured at $V_{ADVREF}/2$.

Figure 46-17. Gain and Offset Errors in Single-ended Mode



where:

- FSe = (FSe+) - (FSe-) is for full-scale error, unit is LSB code
- Offset error E_O is the offset error measured for V_I = 0V
- Gain error E_G = 100 × FSe / 4096, unit in %

The error values in Table 46-37 and Table 46-38 include the sample and hold error as well as the PGA gain error.

Table 46-37. Single-ended Gain Error

Offset Mode	OFFx = 0		OFFx = 0		OFFx = 1		OFFx = 0		OFFx = 1	
Gain Mode	1		2		2		4		4	
AutoCorrection	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
Average Gain Error (%)	0.449	0.078	0.771	-0.010	0.781	0.117	1.069	-0.029	1.064	0.151
Standard Deviation (%)	0.420	0.200	0.430	0.313	0.425	0.327	0.420	0.415	0.415	0.371
Min Value (%)	-0.811	-0.522	-0.518	-0.947	-0.493	-0.864	-0.190	-1.274	-0.181	-0.962
Max Value (%)	1.709	0.679	2.061	0.928	2.056	1.099	2.329	1.216	2.310	1.265

Table 46-38. Single-ended Output Offset Error

Offset Mode	OFFx = 0	OFFx = 0	OFFx = 1	OFFx = 0	OFFx = 1
Gain	1	2	2	4	4
Average Offset Error (LSB)	-5.7	-7.7	-10.3	-7.3	-18.7
Standard Deviation (LSB)	1.8	3.9	3.4	6	7
Min Value (LSB)	-11.1	-19.4	-20.5	-25.3	-39.7
Max Value (LSB)	-0.3	4	-0.1	10.7	2.3

46.7.5.2 ADC Electrical Performances

Single-ended Static Performances

Table 46-39. Single-ended Static Electrical Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
INL	ADC Integral Non-linearity	—	-2	±1	2	LSB
DNL	ADC Differential Non-linearity	—	-1	±0.5	1	LSB

Single-ended Dynamic Performances

Table 46-40. Single-ended Dynamic Electrical Characteristics (1)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
SNR	Signal to Noise Ratio	(1)	56	64	72	dB
THD	Total Harmonic Distortion	(1)	66	74	—	dB
SINAD	Signal to Noise and Distortion	(1)	55	62	—	dB
ENOB	Effective Number of Bits	(1)	9	10.5	—	bits

Note: 1. ADC Clock (f_{ADC}) = 20 MHz, f_S = 1 MHz, f_{IN} = 127 kHz, Frequency band = [1 kHz, 500 kHz] - Nyquist conditions fulfilled.

Differential Static Performances

Table 46-41. Differential Static Electrical Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
INL	Integral Non-linearity		-2	±1	2	LSB
DNL	Differential Non-linearity		-1	±0.5	1	LSB

Differential Dynamic Performances

Table 46-42. Differential Dynamic Electrical Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
SNR	Signal to Noise Ratio	(1)	60	64	74	dB
THD	Total Harmonic Distortion	(1)	76	80	—	dB
SINAD	Signal to Noise and Distortion	(1)	60	64	73	dB
ENOB	Effective Number of Bits	(1)	9.5	10.5	12	bits

Note: 1. ADC Clock (f_{ADC}) = 20 MHz,
f_S = 1 MHz,
f_{IN} = 127 kHz,
Frequency band = [1 kHz, 500 kHz]
Nyquist conditions fulfilled.

10-bit ADC Mode

In 10-bit mode, the ADC produces 12-bit output but the output data in AFEC_CDR is shifted two bits to the right, removing the two LSBs of the 12-bit ADC.

The gain and offset have the same values as for 12-bit mode, with digital full-scale output code range reduced to 1024 (vs 4096).

The INL and DNL have the same values as for 12-bit mode.

The dynamic performances are the 12-bit mode values, reduced by 12 dB.

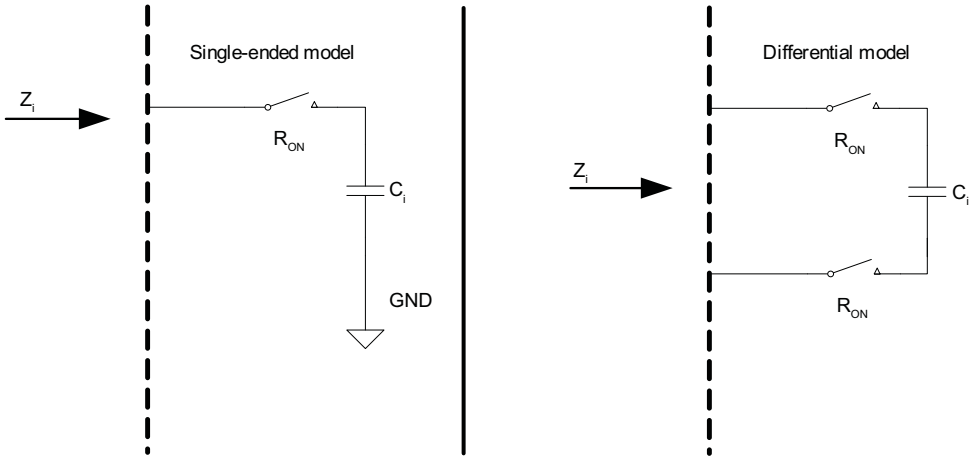
Low Voltage Supply

The ADC operates in 10-bit mode or 12-bit mode. Working at low voltage (V_{DDIN} or/and V_{ADVREF}) between 2 and 2.4V is subject to the following restrictions:

- The field IBCTL must be 00 to reduce the biasing of the ADC under low voltage. See Section 46.7.1.1 “ADC Bias Current”.
- In 10-bit mode, the ADC clock should not exceed 5 MHz (max signal bandwidth is 250 kHz).
- In 12-bit mode, the ADC clock should not exceed 2 MHz (max signal bandwidth is 100 kHz).

46.7.5.3 ADC Channel Input Impedance

Figure 46-18. Input Channel Model



where:

- Z_i is input impedance in single-ended or differential mode
- C_i = 1 to 8 pF ±20% depending on the gain value and mode (SE or DIFF); temperature dependency is negligible
- R_{ON} is typical 2 kΩ and 8 kΩ max (worst case process and high temperature)
- R_{ON} is negligible regarding the value of Z_i

The following formula is used to calculate input impedance:

$$Z_i = \frac{1}{f_S \times C_i}$$

where:

- f_S is the sampling frequency of the ADC channel
- Typ values are used to compute ADC input impedance Z_i

Table 46-43. Input Capacitance (C_{IN}) Values

Gain Selection	Single-ended	Differential
0.5	–	2 pF
1	2 pF	4 pF
2	2 pF	8 pF
4	4 pF	–

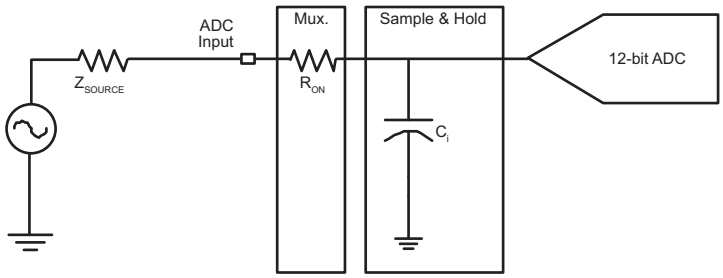
Table 46-44. Z_i Input Impedance

f_S (MHz)	1	0.5	0.25	0.125	0.0625	0.03125	0.015625	0.007813
C_i = 2 pF								
Z_i (MΩ)	0.5	1	2	4	8	16	32	64
C_i = 4 pF								
Z_i (MΩ)	0.25	0.5	1	2	4	8	16	32
C_i = 8 pF								
Z_i (MΩ)	0.125	0.25	0.5	1	2	4	8	16

Track and Hold Time versus Source Output Impedance

Figure 46-19 shows a simplified acquisition path.

Figure 46-19. Simplified Acquisition Path



During the tracking phase, the ADC needs to track the input signal during the tracking time shown below:

$$t_{TRACK} = 0.054 \times Z_{SOURCE} + 205$$

with t_{TRACK} expressed in ns and Z_{SOURCE} expressed in Ω.

The ADC already includes a tracking time of 15 t_{CP_ADC}

Two cases must be considered:

- If the calculated tracking time (t_{TRACK}) is lower than 15 t_{CP_ADC}, then AFEC_MR.TRACKTIM can be set to 0.
- If the calculated tracking time (t_{TRACK}) is higher than 15 t_{CP_ADC}, then AFEC_MR.TRACKTIM must be set to the correct value.

46.7.5.4 AFE DAC Offset Compensation

Table 46-45. AFE DAC Offset Compensation

Parameter	Conditions	Min	Typ	Max	Unit
Resolution - N	—	—	12	—	bits
INL	Range [32 to 4063]	-4	—	+4	LSB
DNL	—	-2	—	+2	LSB
LSB relative to V_{REFIN} Scale	LSB = V_{REFIN} /2e12	—	732	—	µV

46.7.6 ADC Resolution with Averaging

46.7.6.1 Conditions @ 25°C with Gain = 1

- f_{ADC} = 20 MHz; f_{ADC} = 2 MHz for INL and DNL static measurement only
- f_S = 1 MHz, ADC Sampling Frequency in Free Run Mode
- V_{ADVREF} = 3V
- Signal Amplitude: V_{ADVREF}/2, Signal Frequency < 100 Hz
- OSR: Number of Averaged Samples
- V_{DDIN} = 2.4V

Table 46-46. ADC Resolution following Digital Averaging (Gain = 1)

Parameter Averaging Resolution RES (AFEC_EMR)	Over Sampling Ratio	Mode (bits)	INL (LSB)	DNL (LSB)	SNR (dB)	THD (dB)	ENOB (bits)	FS (ksps)
Single-ended Mode								
RES = 0	1	12	±1	±0.5	63.5	-83	10.2	1000
RES = 2	4	13	±1	±1	68.3	-84.5	11	250
RES = 3	16	14	+4 / -2	+3.2 / -1	73	-85.7	11.8	62.5
RES = 4	64	15	+6 / -3.5	—	76.8	-85.8	12.4	15.6
RES = 5	256	16	+15 / -8	—	82.7	-86.2	13.2	3.9
Differential Mode								
RES = 0	1	12	±1	±0.5	64	-83	10.3	1000
RES = 2	4	13	±1	±1	69.2	-83.7	11.2	250
RES = 3	16	14	+3 / -1.5	+3 / -1	74.8	-84.5	12.1	62.5
RES = 4	64	15	+6 / -3.5	—	80.2	-84.5	12.8	15.6
RES = 5	256	16	+10 / -7	—	83.9	-84.9	13.2	3.9

46.7.6.2 Conditions @ 25°C with Gain = 4

- f_{ADC} = 20 MHz
- f_S = 1 MHz, ADC Sampling Frequency in Free Run Mode
- V_{ADVREF}= 3V
- Signal Amplitude: V_{ADVREF}/2, Signal Frequency < 100 Hz
- OSR: Number of Averaged Samples

Table 46-47. ADC Resolution following Digital Averaging (Gain = 4)

Parameter Averaging Resolution RES (AFEC_EMR)	Over Sampling Ratio	Mode (bits)	INL (LSB)	DNL (LSB)	SNR (dB)	THD (dB)	ENOB (bits)	FS (ksps)
Single-ended Mode								
RES = 0	1	12	±1	±0.5	59	-81	9.5	1000
RES = 2	4	13	+1.7 / -1.3	+1.6 / -1	63.1	-82.9	10.2	250
RES = 3	16	14	+1.7 / -2.5	+2 / -1	67	-83.6	10.8	62.5
RES = 4	64	15	±8	—	70.3	-84.5	11.4	15.6
RES = 5	256	16	±12	—	74.8	-85.1	12.1	3.9
Differential Mode								
RES = 0	1	12	±1	±0.5	62	-84.5	10	1000
RES = 2	4	13	±1	±1	67.7	-85.7	10.9	25
RES = 3	16	14	+4.1 / -1.6	+3.4 / -1	73.6	-86.8	11.9	6.25
RES = 4	64	15	±3.5	—	78.7	-86.8	12.7	1.56
RES = 5	256	16	±7.5	—	82.1	-86.8	13.1	0.39

46.8 12-bit DAC Characteristics

Table 46-48. Analog Power Supply Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{DDIN}	Analog Supply	—	2.4	3.0	3.6	V
I_{VDDIN}	Current Consumption	Sleep Mode (Clock OFF)	—	—	3	µA
		Fast Wake-up (Standby Mode, Clock on)	—	2	3	mA
		Normal Mode with 1 Output ON (IBCTLDACCORE = 01, IBCTLCHx = 10)	—	4.3	5.6	mA
		Normal Mode with 2 Outputs ON (IBCTLDACCORE = 01, IBCTLCHx = 1 0)	—	5	6.5	mA

Table 46-49. 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
$t_{REFRESH}$	Refresh Time Between Conversions	—	20	—	—	µs
f_S	Sampling Frequency	—	0.05	—	2	MHz
t_{START}	Startup time	From Sleep Mode to Normal Mode: – Voltage Reference OFF – DAC Core OFF	20	30	40	µs
		From Fast Wake-Up to Normal Mode: – Voltage Reference ON – DAC Core OFF	2.5	3.75	5	
t_{CONV}	Conversion Time	—	—	—	25	t_{CP_DAC}

External voltage reference for DAC is ADVREF. See the ADC voltage reference characteristics Table 46-29 on page 1377.

Table 46-50. Static Performance Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
	Resolution	—	—	12	—	bit
INL	Integral Non-linearity	$2.4V < V_{DDIN} < 2.7V$	-6	—	+6	LSB
		$2.7V < V_{DDIN} < 3.6V$	-2.5	±1	+2.5	
DNL	Differential Non-linearity	$2.4V < V_{DDIN} < 2.7V$	-2.5	±1	+2.5	LSB
		$2.7V < V_{DDIN} < 3.6V$				
E_O	Offset Error	—	-32	±8	32	LSB
E_G	Gain Error	—	-32	±2	32	LSB

Note: DAC Clock (f_{DAC}) = 5 MHz, f_S = 200 kHz, IBCTL = 01.

Table 46-51. Dynamic Performance Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
SNR	Signal to Noise Ratio	2.4V < V_{DDIN} < 2.7V	47	58	70	dB
		2.7V < V_{DDIN} < 3.6V	56	61	74	
THD	Total Harmonic Distortion	2.4V < V_{DDIN} < 2.7V	—	-72	-60	dB
		2.7V < V_{DDIN} < 3.6V	—	-76	-68	
SINAD	Signal to Noise and Distortion	2.4V < V_{DDIN} < 2.7V	47	58	—	dB
		2.7V < V_{DDIN} < 3.6V	56	61	—	
ENOB	Effective Number of Bits	2.4V < V_{DDIN} < 2.7V	7.5	9	12	bits
		2.7V < V_{DDIN} < 3.6V	9	10	12	

Note: DAC Clock (f_{DAC}) = 50 MHz, f_S = 2 MHz, f_{IN} = 241 kHz, IBCTL = 01, FFT using 1024 points or more, Frequency band = [10 kHz, 1 MHz] - Nyquist conditions fulfilled.

Table 46-52. Analog Outputs

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{OR}	Voltage Range	—	(1/6) x V_{ADVREF}	—	(5/6) x V_{ADVREF}	V
SR	Slew Rate	Channel output current versus slew rate (IBCTL for DAC0 or DAC1, noted IBCTLCHx)				V/µs
		R_{LOAD} = 10 kΩ/0 pF < C_{LOAD}< 50 pF				
		IBCTLCHx = 00	—	2.7	—	
		IBCTLCHx = 01	—	5.3	—	
		IBCTLCHx = 10	—	8	—	
		IBCTLCHx = 11	—	10.7	—	
	Output Channel Current Consumption	No resistive load				mA
		IBCTLCHx = 00	—	0.23	—	
		IBCTLCHx = 01	—	0.45	—	
		IBCTLCHx = 10	—	0.67	—	
		IBCTLCHx = 11	—	0.89	—	
t_{sa}	Settling Time	R_{LOAD} = 10 kΩ/0 pF < C_{LOAD}< 50 pF	—	—	0.5	µs
R_{LOAD}	Output Load Resistor		10	—	—	kΩ
C_{LOAD}	Output Load Capacitor		—	30	50	pF

46.9 Analog Comparator Characteristics

Table 46-53. Analog Comparator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_R	Voltage Range	Analog Comparator is supplied by VDDIN	1.62	3.3	3.6	V
V_{IR}	Input Voltage Range	—	GND + 0.2	—	V_{DDIN} - 0.2	V
V_{IO}	Input Offset Voltage	—	—	—	20	mV
I_{VDDIN}	Current Consumption (VDDIN)	Low-power option (ISEL = 0) High-speed option (ISEL = 1)	— —	— —	25 170	µA
V_{hys}	Hysteresis	HYST = 0x01 or 0x10 HYST = 0x11	— —	15 30	50 90	mV
t_{sa}	Settling Time	Overdrive > 100 mV; Low-power option Overdrive > 100 mV; High-speed option	— —	— —	1 0.1	µs

46.10 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 = 4.7 mV/°C.

V_{O_TS} equals 1.44V at T_A 27°C, with a ±60 mV accuracy. The V_{O_TS} slope versus temperature dV_{O_TS}/dT = 4.7 mV/°C only shows a ±7% 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 46-54. Temperature Sensor Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{O_TS}	Output Voltage	T_A = 27° C(1)	—	1.44	—	V
$V_{O\_TS(accuracy)}$	Output Voltage Accuracy	T_A = 27° C(1)	-60		+60	mV
dV_{O_TS}/dT	Temperature Sensitivity (Slope Voltage vs Temperature	(1)	—	4.7	—	mV/°C
—	Slope Accuracy	Over temperature range -40 to 105 °C (1)	-7	—	+7	%
—	Temperature Accuracy(2)	After offset calibration over temperature range -40 to 105 °C	-6	—	+6	°C
		After offset calibration over temperature range 0 to 80 °C	-5	—	+5	°C
t_{START}	Startup Time	(1)	—	5	10	µs
$I_{VDDCORE}$	Current Consumption	(1)	50	70	80	µA

Notes: 1. The value of TS only (the value does not take into account the ADC offset/gain/errors).
2. The temperature accuracy takes into account the ADC offset error, gain error in single ended mode with Gain = 1.

46.11 AC Characteristics

46.11.1 Master Clock Characteristics

Table 46-55. Master Clock Waveform Parameters

Symbol	Parameter	Conditions	Min	Max	Unit
$1/(t_{CPMCK})$	Master Clock Frequency	VDDCORE @ 1.20V	—	120	MHz
		VDDCORE @ 1.08V	—	100	MHz

46.11.2 I/O Characteristics

Criteria used to define the maximum frequency of the I/Os:

- Output duty cycle (40%–60%)
- Minimum output swing: 100 mV to V_{DDIO} - 100 mV
- Addition of rising and falling time inferior to 75% of the period

Table 46-56. I/O Characteristics

Symbol	Parameter	Conditions		Min	Max	Unit
FreqMax1	Pin Group 1[1] Maximum output frequency	10 pF	V_{DDIO} = 1.62V	—	70	MHz
		30 pF		—	45	
$PulseminH_1$	Pin Group 1[1] High Level Pulse Width	10 pF		7.2	—	ns
		30 pF		11	—	
$PulseminL_1$	Pin Group 1[1] Low Level Pulse Width	10 pF		7.2	—	ns
		30 pF		11	—	
FreqMax2	Pin Group 2[2] Maximum output frequency	10 pF		—	46	MHz
		25 pF		—	23	
$PulseminH_2$	Pin Group 2 [2] High Level Pulse Width	10 pF		11	—	ns
		25 pF		21.8	—	
$PulseminL_2$	Pin Group 2[2] Low Level Pulse Width	10 pF		11	—	ns
		25 pF		21.8	—	
FreqMax3	Pin Group3[3] Maximum output frequency	10 pF		—	70	MHz
		25 pF		—	35	
$PulseminH_3$	Pin Group 3[3] High Level Pulse Width	10 pF		7.2	—	ns
		25 pF		14.2	—	
$PulseminL_3$	Pin Group 3[3] Low Level Pulse Width	10 pF		7.2	—	ns
		25 pF		14.2	—	
FreqMax4	Pin Group 4[4] Maximum output frequency	10 pF		—	58	MHz
		25 pF		—	29	
$PulseminH_4$	Pin Group 4[4] High Level Pulse Width	10 pF		8.6	—	ns
		25pF		17.2	—	
$PulseminL_4$	Pin Group 4[4] Low Level Pulse Width	10 pF		8.6	—	ns
		25 pF		17.2	—	
FreqMax5	Pin Group 5[5] Maximum output frequency	25 pF		—	25	MHz

Notes:
1. Pin Group 1 = PA14, PA29
2. Pin Group 2 = PA[4], PA[9–11], PA[15–25], PB[0–7], PB[12–13], PC[0–31], PD[2], PD[18–31], PE[0–5]
3. Pin Group 3 = PA[5–8], PA[12–13], PA[26–28], PA[30–31], PB[8–9], PB[14], PD[0–1], PD[3–17]
4. Pin Group 4 = PA[0–3]
5. Pin Group 5 = PB[10–11]

46.11.3 SPI Characteristics

In Figure 46-21 “SPI Master Mode with (CPOL = NCPHA = 0) or (CPOL = NCPHA = 1)” and Figure 46-22 “SPI Master Mode with (CPOL = 0 and NCPHA = 1) or (CPOL = 1 and NCPHA = 0)” below, the MOSI line shifting edge is represented with a hold time = 0. However, it is important to note that for this device, the MISO line is sampled prior to the MOSI line shifting edge. As shown in Figure 46-20 “MISO Capture in Master Mode”, the device sampling point extends the propagation delay (t_p) for slave and routing delays to more than half the SPI clock period, whereas the common sampling point allows only less than half the SPI clock period.

As an example, an SPI Slave working in Mode 0 is safely driven if the SPI Master is configured in Mode 0.

Figure 46-20. MISO Capture in Master Mode

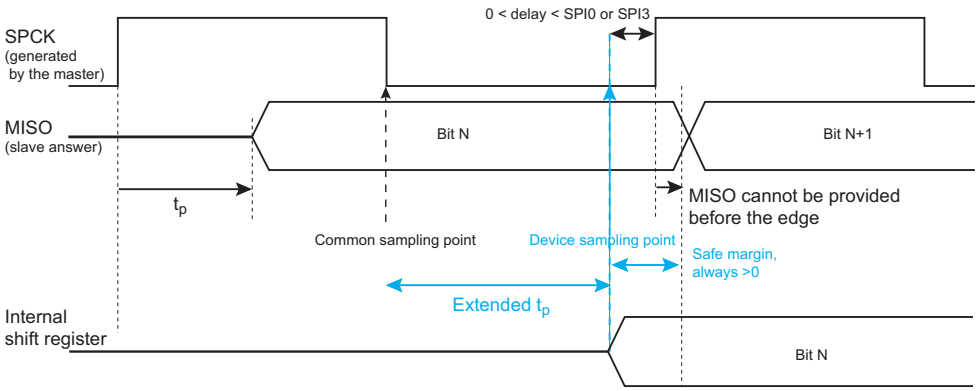


Figure 46-21. SPI Master Mode with (CPOL = NCPHA = 0) or (CPOL = NCPHA = 1)

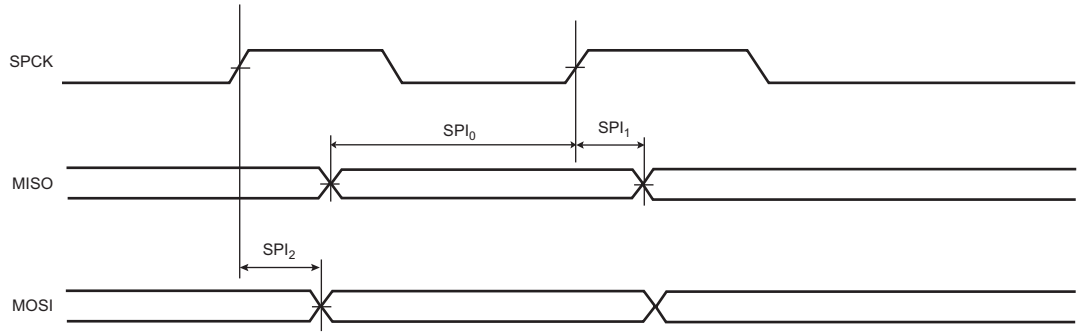


Figure 46-22. SPI Master Mode with (CPOL = 0 and NCPHA = 1) or (CPOL = 1 and NCPHA = 0)

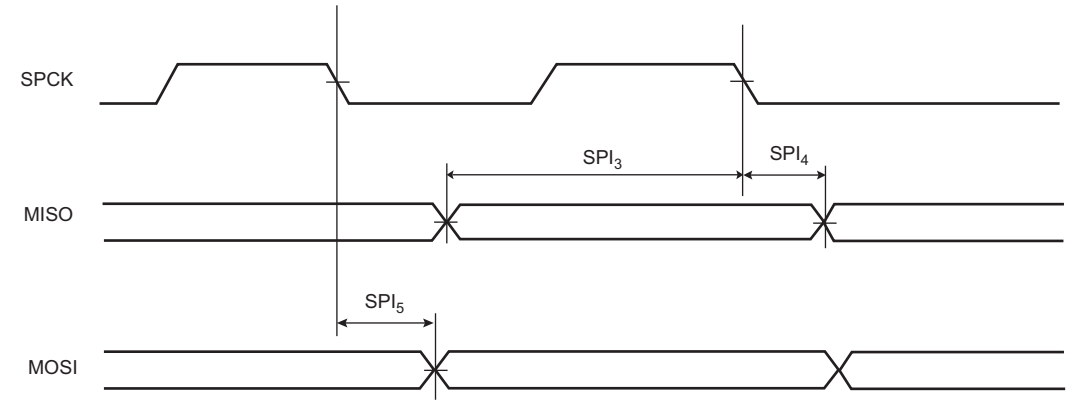


Figure 46-23. SPI Slave Mode with (CPOL = 0 and NCPHA = 1) or (CPOL = 1 and NCPHA = 0)

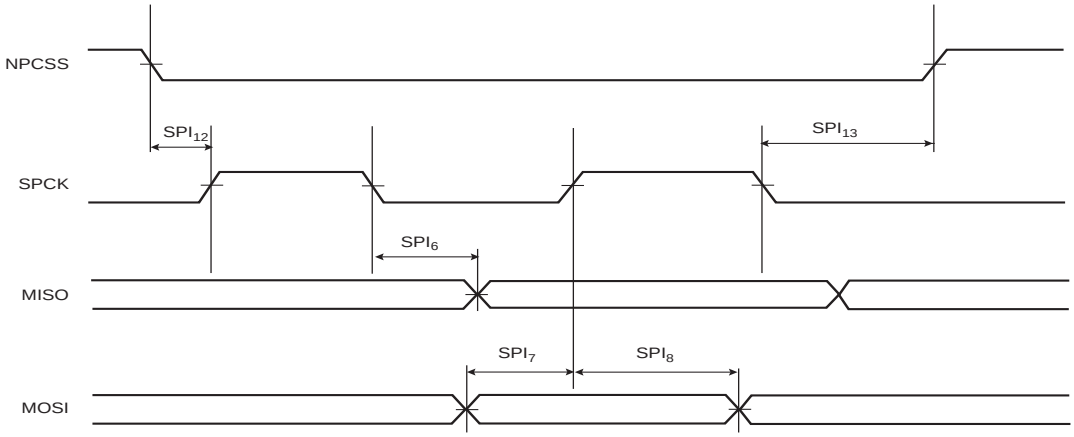
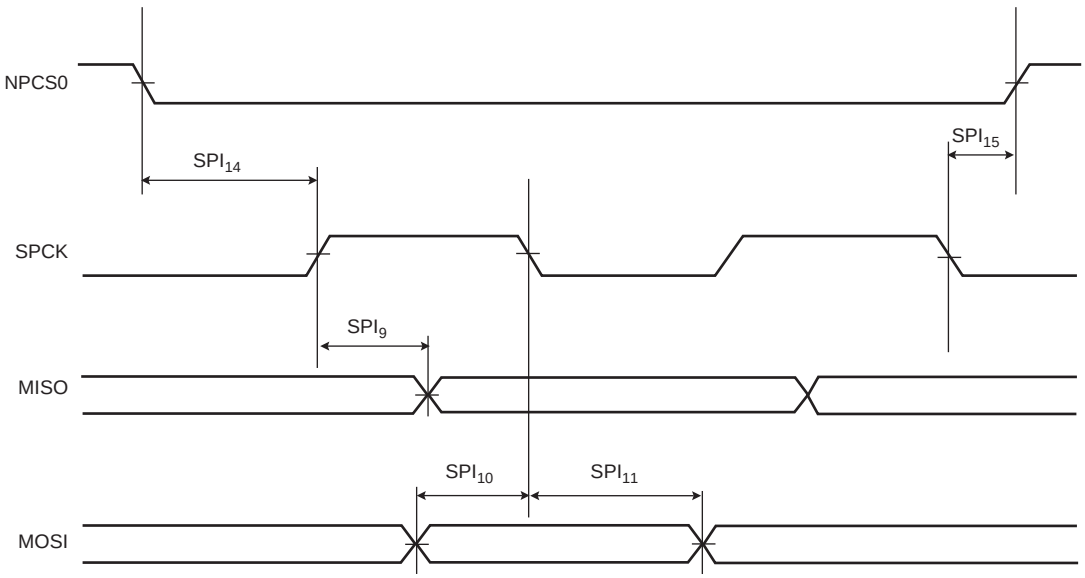


Figure 46-24. SPI Slave Mode with (CPOL = NCPHA = 0) or (CPOL = NCPHA = 1)



46.11.3.1 Maximum SPI Frequency

The following formulas give the 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 46.11.2 “I/O Characteristics”), the maximum SPI frequency is defined by the pin FreqMax value.

Master Read Mode

$$f_{SPCK}Max = \frac{1}{SPI_0(\text{or } SPI_3) + t_{valid}}$$

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 Max, f_{SPCK}Max = 43.4 MHz @ V_{DDIO} = 3.3V.

Slave Read Mode

In slave mode, SPCK is the input clock for the SPI. The maximum 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

$$f_{SPCK}Max = \frac{1}{2x(SPI_{6max}(\text{or } SPI_{9max}) + t_{setup})}$$

For 3.3V I/O domain and SPI_6, f_{SPCK}Max = 21 MHz. t_{SETUP} is the setup time from the master before sampling data.

46.11.3.2 SPI Timings

SPI timings are given for the following domains:

- 3.3V domain: V_{DDIO} from 2.85 to 3.6 V, maximum external capacitor = 40 pF
- 1.8V domain: V_{DDIO} from 1.65 to 1.95 V, maximum external capacitor = 20 pF

Table 46-57. SPI Timings

Symbol	Parameter	Conditions	Min	Max	Unit
SPI_0	MISO Setup time before SPCK rises (master)	3.3V domain	11.0	—	ns
		1.8V domain	12.5	—	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	-3.4	3.0	ns
		1.8V domain	-3.2	1.9	ns
SPI_3	MISO Setup time before SPCK falls (master)	3.3V domain	18.0	—	ns
		1.8V domain	19.8	—	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	-6.4	-1.9	ns
		1.8V domain	-5.9	-2.6	ns
SPI_6	SPCK falling to MISO Delay (slave)	3.3V domain	3.6	11.9	ns
		1.8V domain	4.2	13.9	ns
SPI_7	MOSI Setup time before SPCK rises (slave)	3.3V domain	0	—	ns
		1.8V domain	0	—	ns
SPI_8	MOSI Hold time after SPCK rises (slave)	3.3V domain	2.8	—	ns
		1.8V domain	2.3	—	ns
SPI_9	SPCK rising to MISO Delay (slave)	3.3V domain	3.8	12.1	ns
		1.8V domain	4.2	13.6	ns
SPI_{10}	MOSI Setup time before SPCK falls (slave)	3.3V domain	0	—	ns
		1.8V domain	0	—	ns
SPI_{11}	MOSI Hold time after SPCK falls (slave)	3.3V domain	2.4	—	ns
		1.8V domain	2.5	—	ns
SPI_{12}	NPCS setup to SPCK rising (slave)	3.3V domain	3.2	—	ns
		1.8V domain	3.3	—	ns
SPI_{13}	NPCS hold after SPCK falling (slave)	3.3V domain	0	—	ns
		1.8V domain	0	—	ns
SPI_{14}	NPCS setup to SPCK falling (slave)	3.3V domain	3.8	—	ns
		1.8V domain	3.3	—	ns
SPI_{15}	NPCS hold after SPCK falling (slave)	3.3V domain	0	—	ns
		1.8V domain	0	—	ns

Note that in SPI master mode, the SAM4E does not sample the data (MISO) on the opposite edge where the data clocks out (MOSI), but the same edge is used. See Figure 46-21 and Figure 46-22.

46.11.4 HSMCI Timings

The High Speed MultiMedia Card Interface (HSMCI) supports the MultiMedia Card (MMC) Specification V4.3, the SD Memory Card Specification V2.0, the SDIO V2.0 specification and CE-ATA V1.1.

46.11.5 SMC Timings

Timings are given in the following domains:

- 1.8V domain: V_{DDIO} from 1.65V to 1.95V, maximum external capacitor = 30 pF
- 3.3V domain: V_{DDIO} from 2.85V to 3.6V, maximum external capacitor = 50 pF

Timings are given assuming a capacitance load on data, control and address pads.

In the tables that follow, t_{CPMCK} is MCK period.

46.11.5.1 Read Timings

Table 46-58. SMC Read Signals - NRD Controlled (READ_MODE = 1)

Symbol	Parameter	VDDIO Supply / Min: 1.8V Domain	Min: 3.3V Domain	Max: 1.8V Domain	Max: 3.3V Domain	Unit
NO HOLD Settings (NRD_HOLD = 0)						
SMC_1	Data Setup before NRD High	20.6	18.6	—	—	ns
SMC_2	Data Hold after NRD High	0	0	—	—	ns
HOLD Settings (NRD_HOLD ≠ 0)						
SMC_3	Data Setup before NRD High	15.9	14.1	—	—	ns
SMC_4	Data Hold after NRD High	0	0	—	—	ns
HOLD or NO HOLD Settings (NRD_HOLD ≠ 0, NRD_HOLD = 0)						
SMC_5	A0–A22 Valid before NRD High	(NRD_SETUP + NRD_PULSE) × t_{CPMCK} - 6.8	(NRD_SETUP + NRD_PULSE) × t_{CPMCK} - 6.5	—	—	ns
SMC_6	NCS low before NRD High	(NRD_SETUP + NRD_PULSE - NCS_RD_SETUP) × t_{CPMCK} - 4.6	(NRD_SETUP + NRD_PULSE - NCS_RD_SETUP) × t_{CPMCK} - 5.1	—	—	ns
SMC_7	NRD Pulse Width	NRD_PULSE × t_{CPMCK} - 7.5	NRD_PULSE × t_{CPMCK} - 6.6	—	—	ns

Table 46-59. SMC Read Signals - NCS Controlled (READ_MODE = 0)

Symbol	Parameter	VDDIO Supply / Min: 1.8V Domain	Min: 3.3V Domain	Max: 1.8V Domain	Max: 3.3V Domain	Unit
NO HOLD Settings (NCS_RD_HOLD = 0)						
SMC_8	Data Setup before NCS High	21.8	19.4	—	—	ns
SMC_9	Data Hold after NCS High	0	0	—	—	ns
HOLD Settings (NCS_RD_HOLD ≠ 0)						
SMC_{10}	Data Setup before NCS High	17.1	14.9	—	—	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}	A0–A22 valid before NCS High	(NCS_RD_SETUP + NCS_RD_PULSE) × t_{CPMCK} - 6.9	(NCS_RD_SETUP + NCS_RD_PULSE) × t_{CPMCK} - 6.6	—	—	ns
SMC_{13}	NRD low before NCS High	(NCS_RD_SETUP + NCS_RD_PULSE - NRD_SETUP) × t_{CPMCK} - 5.8	(NCS_RD_SETUP + NCS_RD_PULSE - NRD_SETUP) × t_{CPMCK} - 5.5	—	—	ns

Table 46-59. SMC Read Signals - NCS Controlled (READ_MODE = 0) (Continued)

SMC_{14}	NCS Pulse Width	NCS_RD_PULSE length × t_{CPMCK} - 7.6	NCS_RD_PULSE length × t_{CPMCK} - 6.7	—	—	ns

46.11.5.2 Write Timings

Table 46-60. SMC Write Signals - NWE Controlled (WRITE_MODE = 1)

Symbol	Parameter	VDDIO Supply / Min	1.8V Domain / Min	3.3V Domain / Min	1.8V Domain / Max	3.3V Domain / Max	Unit
HOLD or NO HOLD Settings (NWE_HOLD ≠ 0, NWE_HOLD = 0)							
SMC_{15}	Data Out Valid before NWE High		NWE_PULSE × t_{CPMCK} - 6.2	NWE_PULSE × t_{CPMCK} - 5.9	—	—	ns
SMC_{16}	NWE Pulse Width		NWE_PULSE × t_{CPMCK} - 7.0	NWE_PULSE × t_{CPMCK} - 6.1	—	—	ns
SMC_{17}	A0–A22 valid before NWE low		NWE_SETUP × t_{CPMCK} - 6.6	NWE_SETUP × t_{CPMCK} - 6.2	—	—	ns
SMC_{18}	NCS low before NWE high		(NWE_SETUP - NCS_RD_SETUP + NWE_PULSE) × t_{CPMCK} - 5.9	(NWE_SETUP - NCS_RD_SETUP + NWE_PULSE) × t_{CPMCK} - 5.5	—	—	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} - 9.4	NWE_HOLD × t_{CPMCK} - 7.6	—	—	ns
SMC_{20}	NWE High to NCS Inactive (1)		(NWE_HOLD - NCS_WR_HOLD) × t_{CPMCK} - 6.0	(NWE_HOLD - NCS_WR_HOLD) × t_{CPMCK} - 5.6	—	—	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)		3.3	3.2	—	—	ns

Notes: 1. Hold length = total cycle duration - setup duration - pulse duration. “hold length” is for “NCS_WR_HOLD length” or “NWE_HOLD length”.

Table 46-61. SMC Write NCS Controlled (WRITE_MODE = 0)

Symbol	Parameter	VDDIO Supply / Min	1.8V Domain / Min	3.3V Domain / Min	1.8V Domain / Max	3.3V Domain / Max	Unit
SMC_{22}	Data Out Valid before NCS High		NCS_WR_PULSE × t_{CPMCK} - 6.3	NCS_WR_PULSE × t_{CPMCK} - 6.0	—	—	ns
SMC_{23}	NCS Pulse Width		NCS_WR_PULSE × t_{CPMCK} - 7.6	NCS_WR_PULSE × t_{CPMCK} - 6.7	—	—	ns
SMC_{24}	A0–A22 valid before NCS low		NCS_WR_SETUP × t_{CPMCK} - 6.7	NCS_WR_SETUP × t_{CPMCK} - 6.3	—	—	ns

Table 46-61. SMC Write NCS Controlled (WRITE_MODE = 0)

	VDDIO Supply	1.8V Domain	3.3V Domain	1.8V Domain	3.3V Domain	
Symbol	**Parameter**	**Min**		**Max**		**Unit**
SMC_{25}	NWE low before NCS high	(NCS_WR_SETUP - NWE_SETUP + NCS pulse) × t_{CPMCK} - 5.6	(NCS_WR_SETUP - NWE_SETUP + NCS pulse) × t_{CPMCK} - 5.3	—	—	ns
SMC_{26}	NCS High to Data Out, A0–A25, change	NCS_WR_HOLD × t_{CPMCK} - 10.6	NCS_WR_HOLD × t_{CPMCK} - 9.0	—	—	ns
SMC_{27}	NCS High to NWE Inactive	(NCS_WR_HOLD - NWE_HOLD) × t_{CPMCK} - 7.0	(NCS_WR_HOLD - NWE_HOLD) × t_{CPMCK} - 6.8	—	—	ns

Figure 46-25. SMC Timings - NCS Controlled Read and Write

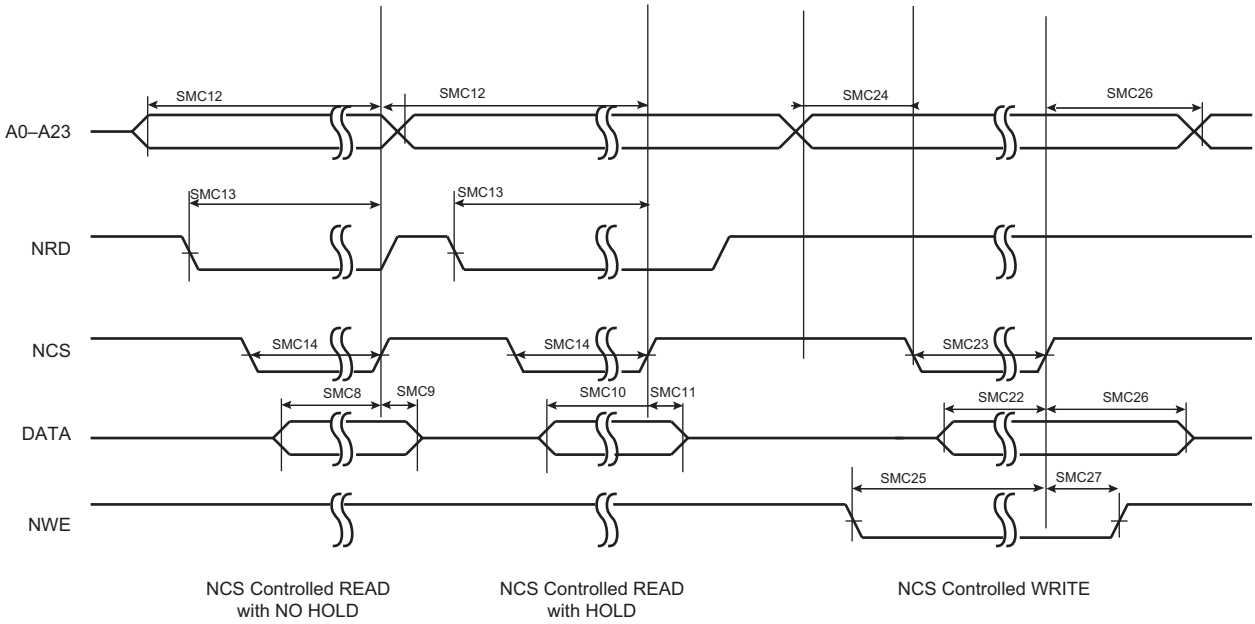
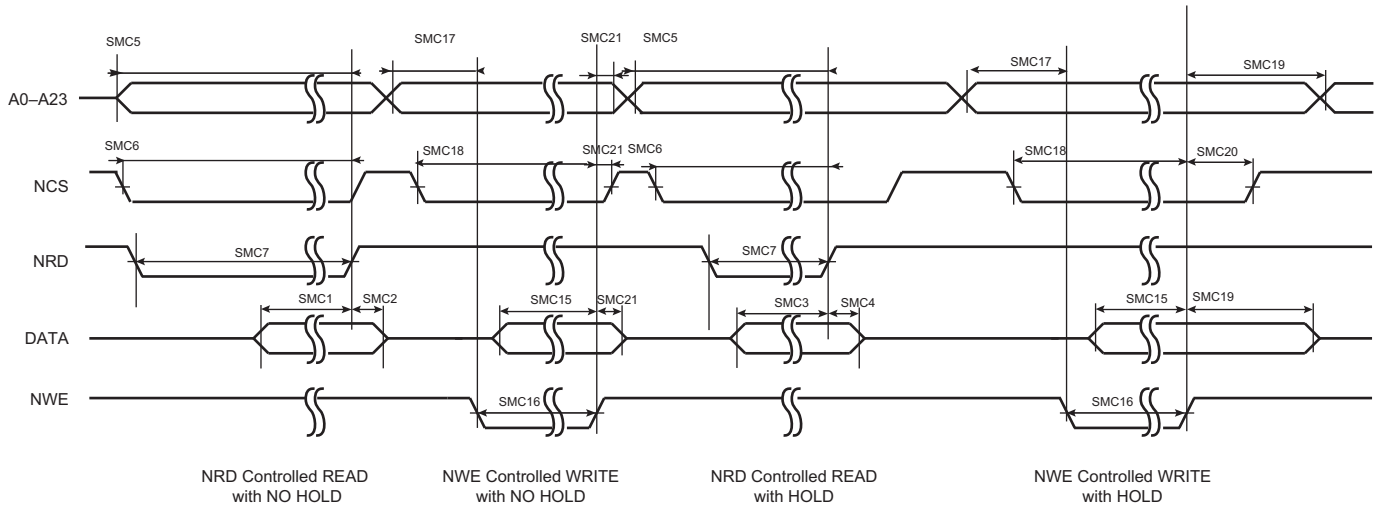


Figure 46-26. SMC Timings - NRD Controlled Read and NWE Controlled Write



46.11.6 USART in SPI Mode Timings

Timings are given in the following domains:

- 1.8V domain: V_{DDIO} from 1.65 to 1.95 V, maximum external capacitor = 20 pF
- 3.3V domain: V_{DDIO} from 2.85 to 3.6 V, maximum external capacitor = 40 pF

Figure 46-27. USART SPI Master Mode

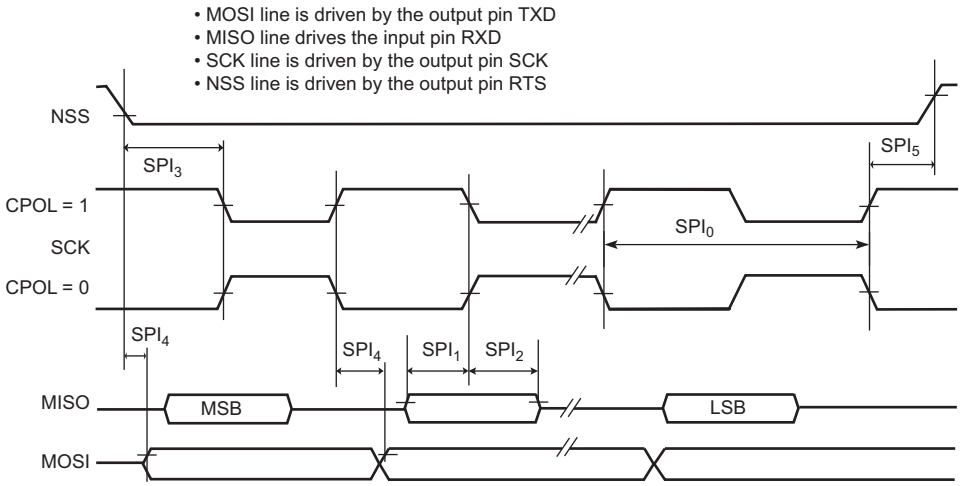


Figure 46-28. USART SPI Slave Mode (Mode 1 or 2)

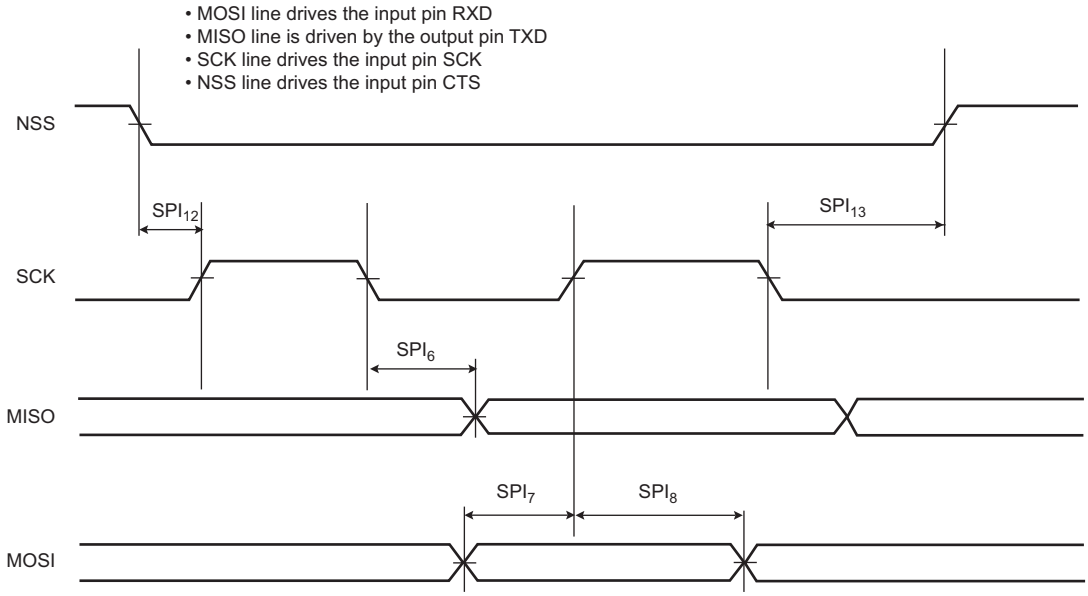
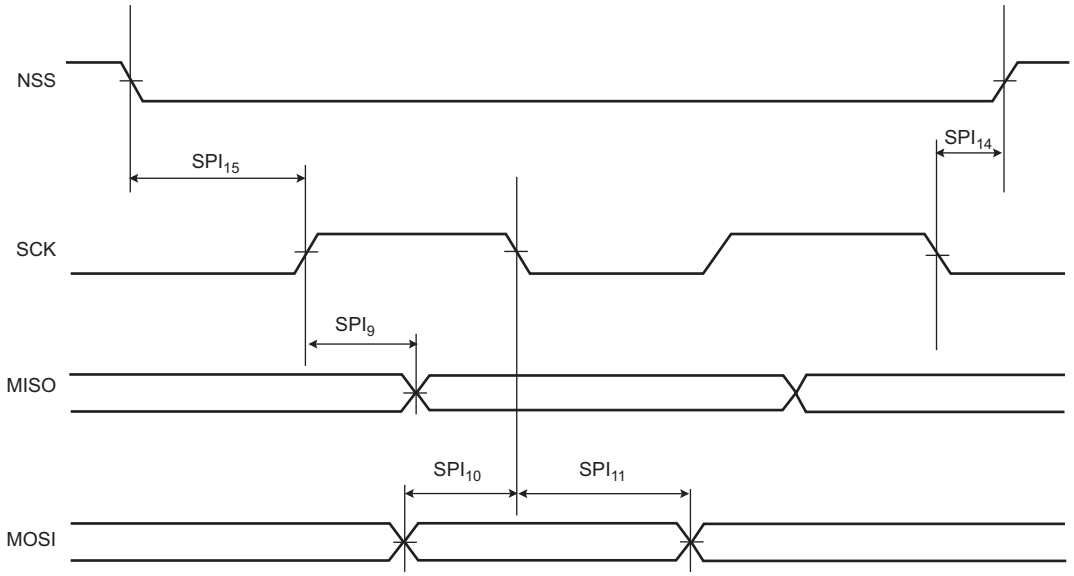


Figure 46-29. USART SPI Slave Mode (Mode 0 or 3)



46.11.6.1 USART SPI TImings

Table 46-62. USART SPI Timings

Symbol	Parameter	Conditions	Min	Max	Unit
Master Mode					
SPI_0	SCK Period	1.8V domain 3.3V domain	MCK/6	—	ns
SPI_1	Input Data Setup Time	1.8V domain 3.3V domain	0.5 × MCK + 3.3 0.5 × MCK + 3.7	—	ns
SPI_2	Input Data Hold Time	1.8V domain 3.3V domain	1.5 × MCK + 0.8 1.5 × MCK + 1.1	—	ns
SPI_3	Chip Select Active to Serial Clock	1.8V domain 3.3V domain	1.5 × SPCK - 1.4 1.5 × SPCK - 1.9	—	ns
SPI_4	Output Data Setup Time	1.8V domain 3.3V domain	- 6.6 - 6.0	9.1 9.8	ns
SPI_5	Serial Clock to Chip Select Inactive	1.8V domain 3.3V domain	1 × SPCK - 4.1 1 × SPCK - 4.6	—	ns
Slave Mode					
SPI_6	SCK falling to MISO	1.8V domain 3.3V domain	3.9 3.2	17.1 15.1	ns
SPI_7	MOSI Setup time before SCK rises	1.8V domain 3.3V domain	2 × MCK + 2.6 2 × MCK + 2.5	—	ns
SPI_8	MOSI Hold time after SCK rises	1.8V domain 3.3V domain	1.3 1.8	—	ns
SPI_9	SCK rising to MISO	1.8V domain 3.3V domain	3.9 3.3	17.2 15.8	ns
SPI_{10}	MOSI Setup time before SCK falls	1.8V domain 3.3V domain	2 × MCK + 2.5 2 × MCK + 3.0	—	ns
SPI_{11}	MOSI Hold time after SCK falls	1.8V domain 3.3V domain	1.4 1.3	—	ns
SPI_{12}	NPCS0 setup to SCK rising	1.8V domain 3.3V domain	2,.5 × MCK + 1.3 2.5 × MCK + 0.4	—	ns
SPI_{13}	NPCS0 hold after SCK falling	1.8V domain 3.3V domain	1.5 × MCK + 1.7 1.5 × MCK + 1.0	—	ns
SPI_{14}	NPCS0 setup to SCK falling	1.8V domain 3.3V domain	2.5 × MCK + 1.2 2.5 × MCK + 0.9	—	ns
SPI_{15}	NPCS0 hold after SCK rising	1.8V domain 3.3V domain	1.5 × MCK + 1.6 1.5 × MCK + 1.5	—	ns

46.11.7 Two-wire Serial Interface Characteristics

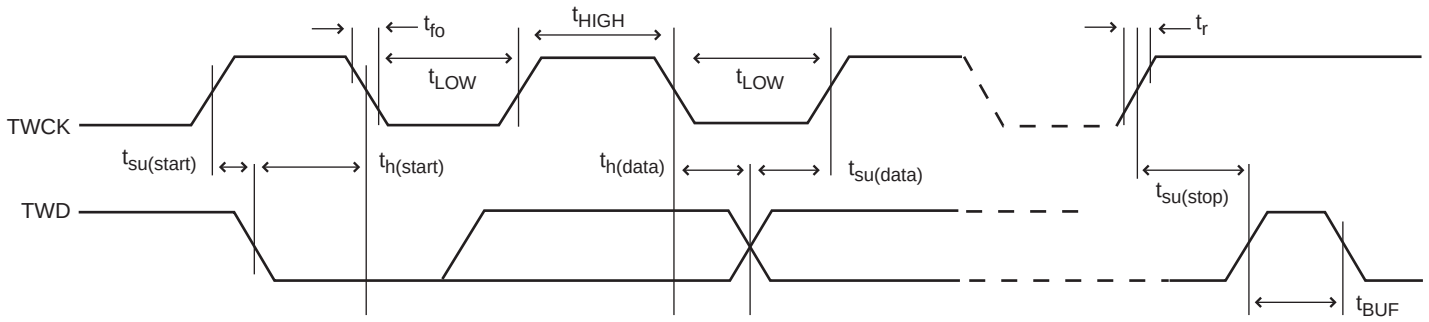
Table 46-63 describes the requirements for devices connected to the Two-wire Serial Bus. For timing symbols refer to Figure 46-30.

Table 46-63. Two-wire Serial Bus Requirements

Symbol	Parameter	Conditions	Min	Max	Unit
V_{IL}	Low-level input voltage	—	-0.3	$0.3 \times V_{DDIO}$	V
V_{IH}	High-level input voltage	—	$0.7 \times V_{DDIO}$	V_{CC} + 0.3	V
V_{hys}	Hysteresis of Schmitt Trigger Inputs	—	0.150	—	V
V_{OL}	Low-level output 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_{of}	Output Fall Time from V_{IHmin} to V_{ILmax}	10 pF < C_b < 400 pF Figure 46-30	$20 + 0.1C_b$ (1)(2)	250	ns
C_i (1)	Capacitance for each I/O Pin	—	—	10	pF
f_{TWCK}	TWCK Clock Frequency	—	0	400	kHz
R_P	Value of Pull-up resistor	$f_{TWCK} \le 100$ kHz	$(V_{DDIO}$ - 0.4V) ÷ 3mA	1000ns ÷ C_b	Ω
		$f_{TWCK} > 100$ kHz		300ns ÷ C_b	Ω
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
$t_{su(start)}$	Set-up 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

Notes:
1. Required only for $f_{TWCK} > 100$ kHz.
2. C_b = capacitance of one bus line in pF. Per I2C Standard, 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

Figure 46-30. Two-wire Serial Bus Timing



46.11.8 Ethernet MAC (GMAC) Characteristics

46.11.8.1 Timing Conditions

Table 46-64. Capacitance Load On Data, Clock Pads

Supply	Corner		
	MAX	STH	MIN
3.3V	20 pf	20 pf	0 pf
1.8V	20 pf	20 pf	0 pf

46.11.8.2 Timing Constraints

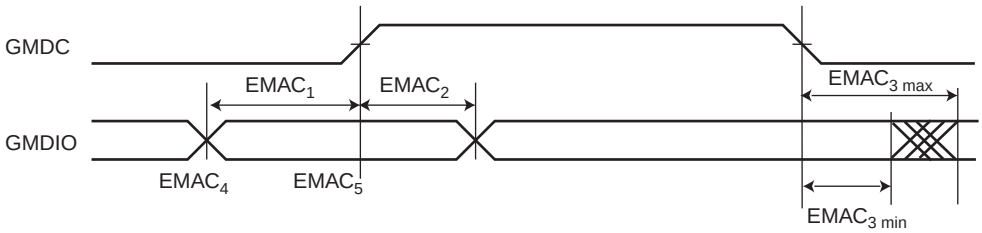
The Ethernet controller must be constrained so as to satisfy the standard timings given in Table 46-65 and Table 46-66, in MAX and STH corners**.**

Table 46-65. EMAC Signals Relative to GMDC

Symbol	Parameter	Min (ns)	Max (ns)
$EMAC_1$	Setup for GMDIO from GMDC rising	10	—
$EMAC_2$	Hold for GMDIO from GMDC rising	10	—
$EMAC_3$	GMDIO toggling from GMDC falling	0 (1)	10 (1)

Note: 1. For EMAC output signals, Min and Max access time are defined. The Min access time is the time between the GMDC falling edge and the signal change. The Max access timing is the time between the GMDC falling edge and the signal stabilization. Figure 46-31 illustrates Min and Max accesses for $EMAC_3$.

Figure 46-31. Min and Max Access Time of EMAC Output Signals

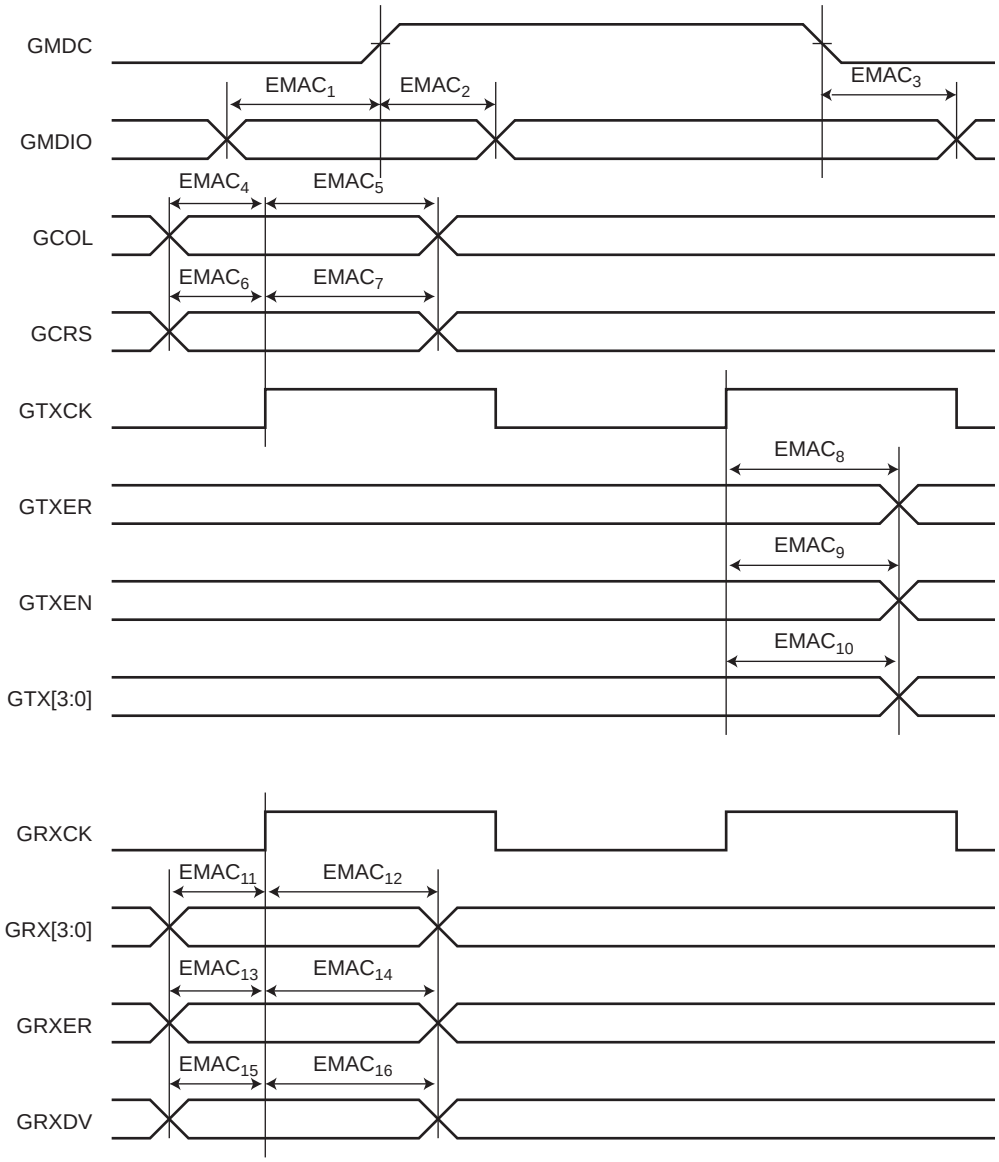


46.11.8.3 MII Mode

Table 46-66. EMAC MII Timings

Symbol	Parameter	Min (ns)	Max (ns)
$EMAC_4$	Setup for GCOL from GTXCK rising	10	—
$EMAC_5$	Hold for GCOL from GTXCK rising	10	—
$EMAC_6$	Setup for GCRS from GTXCK rising	10	—
$EMAC_7$	Hold for GCRS from GTXCK rising	10	—
$EMAC_8$	GTXER toggling from GTXCK rising	10	25
$EMAC_9$	GTXEN toggling from GTXCK rising	10	25
$EMAC_{10}$	GTX toggling from GTXCK rising	10	25
$EMAC_{11}$	Setup for GRX from GRXCK	10	—
$EMAC_{12}$	Hold for GRX from GRXCK	10	—
$EMAC_{13}$	Setup for GRXER from GRXCK	10	—
$EMAC_{14}$	Hold for GRXER from GRXCK	10	—
$EMAC_{15}$	Setup for GRXDV from GRXCK	10	—
$EMAC_{16}$	Hold for GRXDV from GRXCK	10	—

Figure 46-32. EMAC MII Mode Signals



46.11.9 Embedded Flash Characteristics

The embedded flash is fully tested during production test. The flash contents are not set to a known state prior to shipment. Therefore, the flash contents should be erased prior to programming an application.

The maximum operating frequency given in Table 46-67 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.

Table 46-67. Embedded Flash Wait State at 105°C

FWS	Read Operations	Maximum Operating Frequency (MHz)			
		VDDCORE 1.08 V		VDDCORE 1.2 V	
		VDDIO 1.62–3.6 V	VDDIO 2.7–3.6 V	VDDIO 1.62–3.6 V	VDDIO 2.7–3.6 V
0	1 cycle	17	20	17	21
1	2 cycles	34	41	35	43
2	3 cycles	51	62	53	64
3	4 cycles	69	83	71	86
4	5 cycles	86	104	88	107
5	6 cycles	100	–	106	129
6	7 cycles	–	–	124	–

Table 46-68. AC Flash Characteristics

Parameter	Conditions	Min	Typ	Max	Unit
Program Cycle Time	Write page mode	–	1.5	3	ms
	Erase page mode	–	10	50	ms
	Erase block mode (by 4 Kbytes)	–	50	200	ms
	Erase sector mode	–	400	950	ms
Erase Pin Assertion Time	Erase pin high	200	–	–	ms
Full Chip Erase	1 Mbyte	–	9	18	s
	512 Kbytes	–	5.5	11	
Data Retention	Not Powered or Powered	–	20	–	years
Endurance	Write/Erase cycles per page, block or sector @ 105°C	10k	–	–	cycles

47. SAM4E Mechanical Characteristics

The SAM4E series devices are available in TFBGA100, LFBGA144, LQFP100, and LQFP144 packages.

47.1 100-ball TFBGA Package Drawing

Figure 47-1. 100-ball TFBGA Package Drawing

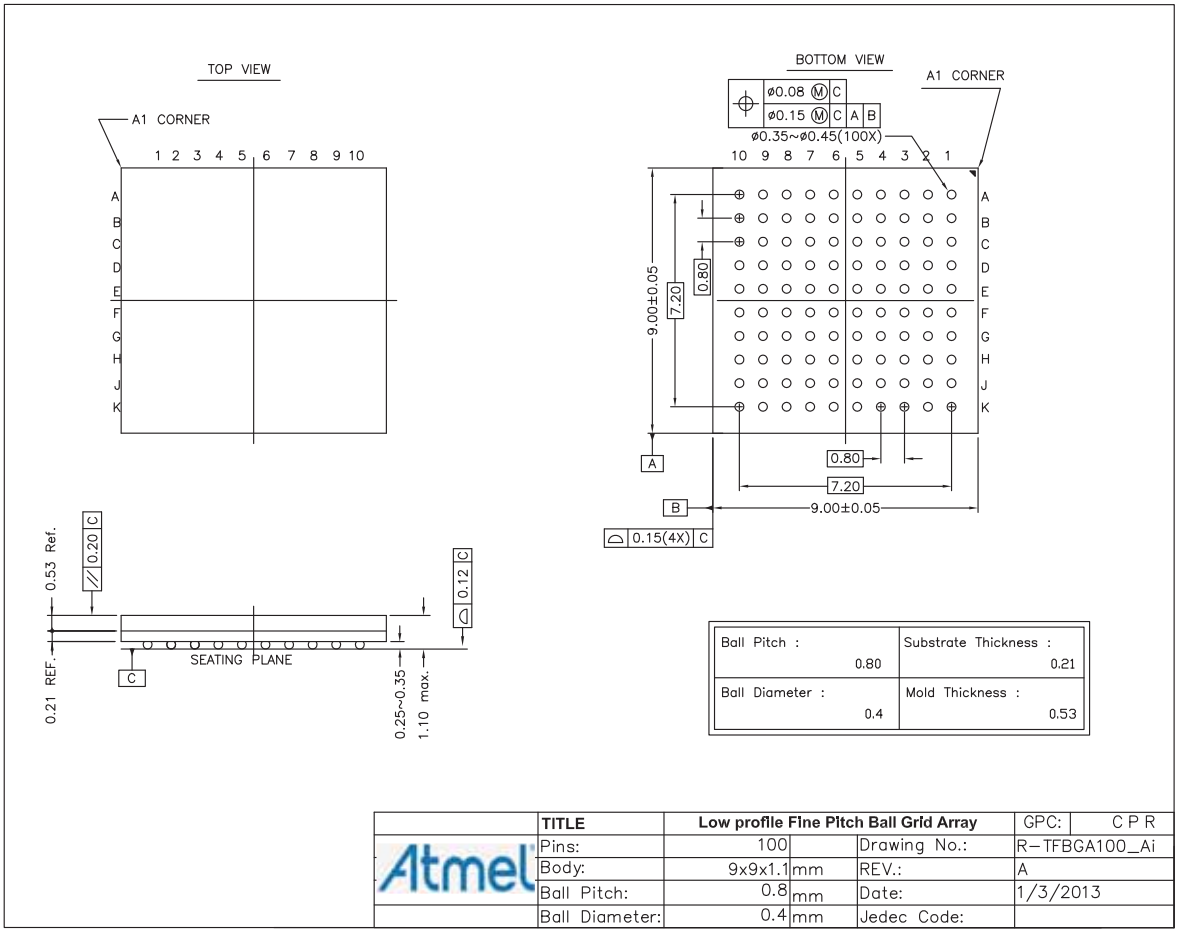


Table 47-1. Device and TFBGA Package Maximum Weight (Preliminary)

SAM4E	150	mg

Table 47-2. TFBGA Package Reference

JEDEC Drawing Reference	MO-275-DDAC-2
JESD97 Classification	e8

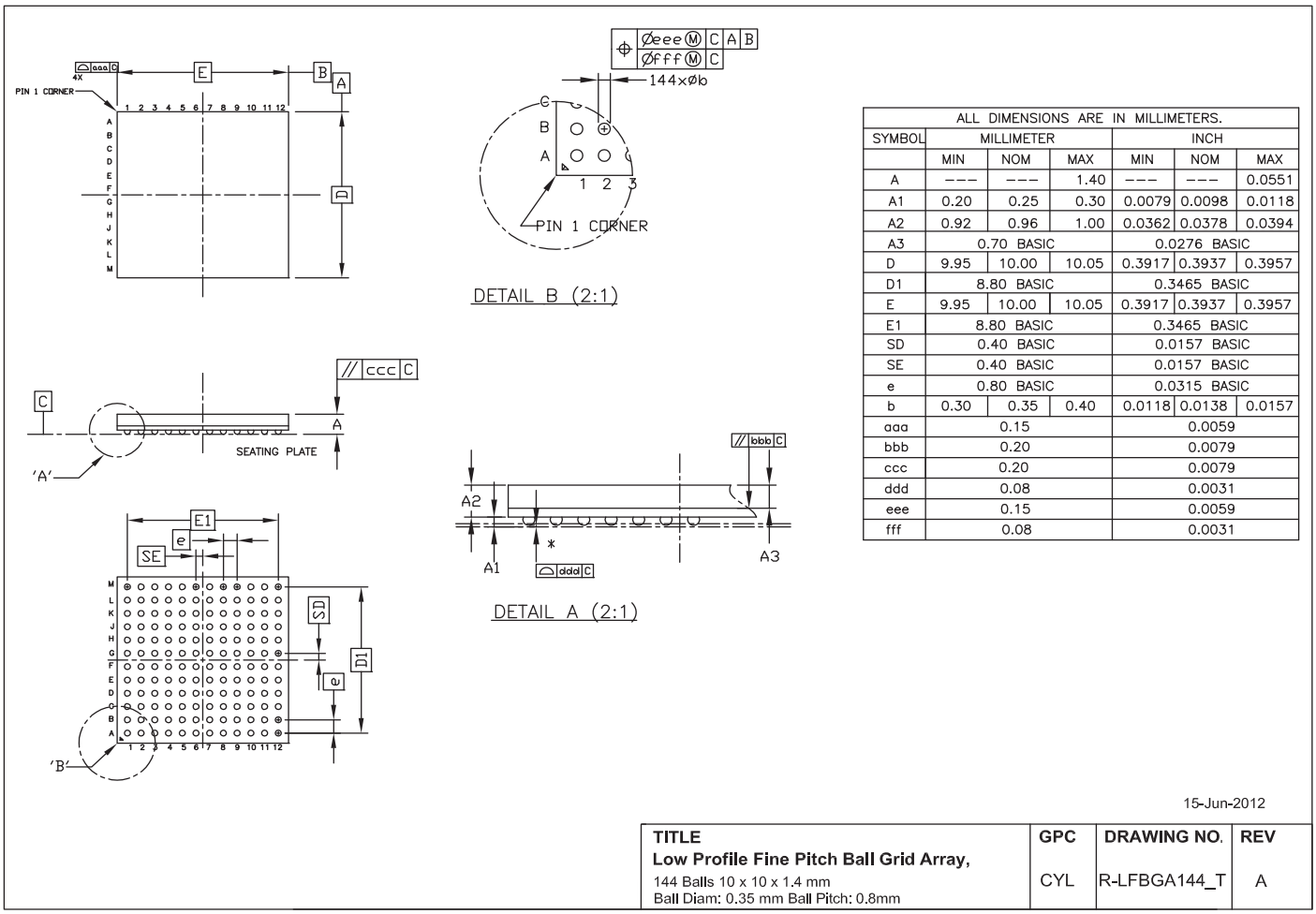
Table 47-3. TFBGA Package Characteristics

Moisture Sensitivity Level	3

This package respects the recommendations of the NEMI User Group.

47.2 144-ball LFBGA Package Drawing

Figure 47-2. 144-ball LFBGA Package Drawing



ALL DIMENSIONS ARE IN MILLIMETERS.						
SYMBOL	MILLIMETER			INCH		
	MIN	NOM	MAX	MIN	NOM	MAX
A	---	---	1.40	---	---	0.0551
A1	0.20	0.25	0.30	0.0079	0.0098	0.0118
A2	0.92	0.96	1.00	0.0362	0.0378	0.0394
A3	0.70 BASIC			0.0276 BASIC		
D	9.95	10.00	10.05	0.3917	0.3937	0.3957
D1	8.80 BASIC			0.3465 BASIC		
E	9.95	10.00	10.05	0.3917	0.3937	0.3957
E1	8.80 BASIC			0.3465 BASIC		
SD	0.40 BASIC			0.0157 BASIC		
SE	0.40 BASIC			0.0157 BASIC		
e	0.80 BASIC			0.0315 BASIC		
b	0.30	0.35	0.40	0.0118	0.0138	0.0157
aaa	0.15			0.0059		
bbb	0.20			0.0079		
ccc	0.20			0.0079		
ddd	0.08			0.0031		
eee	0.15			0.0059		
fff	0.08			0.0031		

TITLE	GPC	DRAWING NO.	REV
Low Profile Fine Pitch Ball Grid Array, 144 Balls 10 x 10 x 1.4 mm Ball Diam: 0.35 mm Ball Pitch: 0.8mm	CYL	R-LFBGA144_T	A

Table 47-4. Device and LFBGA Package Maximum Weight (Preliminary)

SAM4E	200	mg

Table 47-5. LFBGA Package Reference

JEDEC Drawing Reference	MS-275-EEAD-1
JESD97 Classification	e8

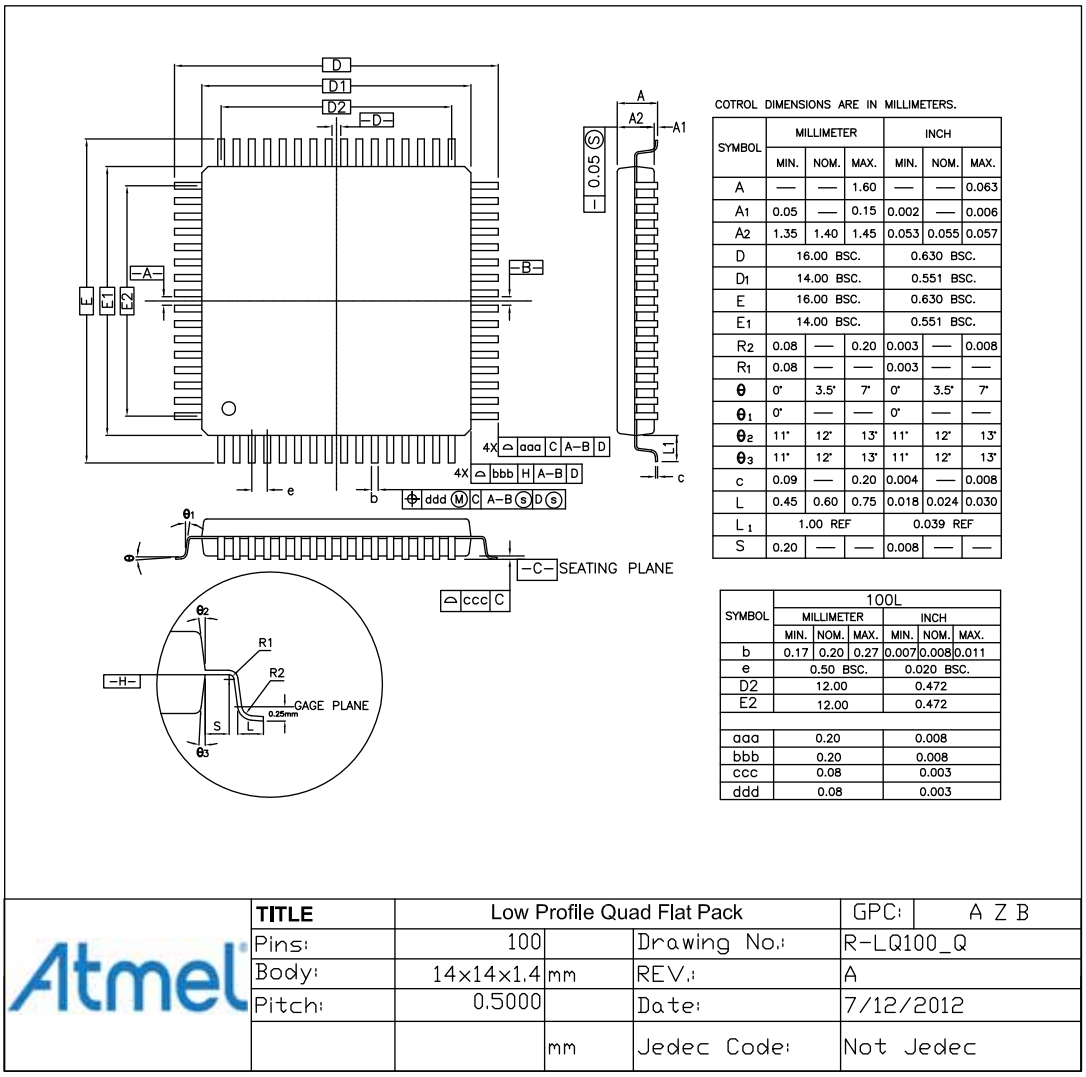
Table 47-6. LFBGA Package Characteristics

Moisture Sensitivity Level	3

This package respects the recommendations of the NEMI User Group.

47.3 100-lead LQFP Package Drawing

Figure 47-3. 100-lead LQFP Package Drawing



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	—	—

SYMBOL	100L					
	MILLIMETER			INCH		
	MIN.	NOM.	MAX.	MIN.	NOM.	MAX.
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		
aaa	0.20			0.008		
bbb	0.20			0.008		
ccc	0.08			0.003		
ddd	0.08			0.003		

Atmel	TITLE	Low Profile Quad Flat Pack		GPC:	A Z B
	Pins:	100		Drawing No.:	R-LQ100_Q
	Body:	14x14x1.4	mm	REV.:	A
	Pitch:	0.5000		Date:	7/12/2012
			mm	Jedec Code:	Not Jedec

Table 47-7. Device and LQFP Package Maximum Weight (Preliminary)

SAM4E	740	mg

Table 47-8. LQFP Package Reference

JEDEC Drawing Reference	MS-026
JESD97 Classification	e3

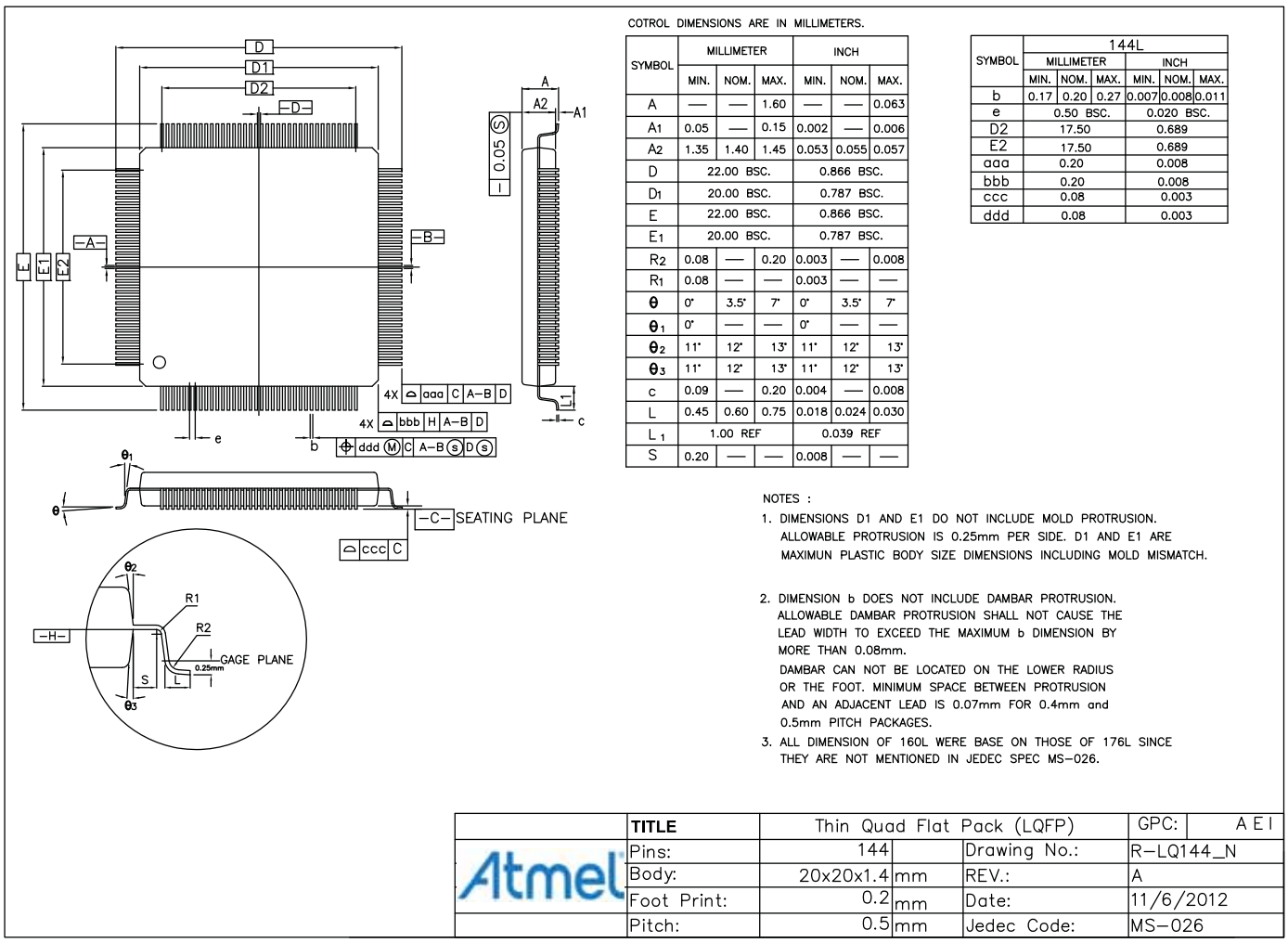
Table 47-9. LQFP Package Characteristics

Moisture Sensitivity Level	3

This package respects the recommendations of the NEMI User Group.

47.4 144-lead LQFP Package Drawing

Figure 47-4. 144-lead LQFP Package Drawing



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	22.00 BSC.			0.866 BSC.		
D1	20.00 BSC.			0.787 BSC.		
E	22.00 BSC.			0.866 BSC.		
E1	20.00 BSC.			0.787 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	—	—

SYMBOL	144L					
	MILLIMETER			INCH		
	MIN.	NOM.	MAX.	MIN.	NOM.	MAX.
b	0.17	0.20	0.27	0.007	0.008	0.011
e	0.50 BSC.			0.020 BSC.		
D2	17.50			0.689		
E2	17.50			0.689		
aaa	0.20			0.008		
bbb	0.20			0.008		
ccc	0.08			0.003		
ddd	0.08			0.003		

NOTES :

1. DIMENSIONS D1 AND E1 DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE PROTRUSION IS 0.25mm PER SIDE. D1 AND E1 ARE MAXIMUN PLASTIC BODY SIZE DIMENSIONS INCLUDING MOLD MISMATCH.
2. 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 0.08mm. DAMBAR CAN NOT BE LOCATED ON THE LOWER RADIUS OR THE FOOT. MINIMUM SPACE BETWEEN PROTRUSION AND AN ADJACENT LEAD IS 0.07mm FOR 0.4mm and 0.5mm PITCH PACKAGES.
3. ALL DIMENSION OF 160L WERE BASE ON THOSE OF 176L SINCE THEY ARE NOT MENTIONED IN JEDEC SPEC MS-026.

	TITLE	Thin Quad Flat Pack (LQFP)		GPC:	A E I
Atmel	Pins:	144		Drawing No.:	R-LQ144_N
	Body:	20x20x1.4	mm	REV.:	A
	Foot Print:	0.2	mm	Date:	11/6/2012
	Pitch:	0.5	mm	Jedec Code:	MS-026

Table 47-10. Device and LQFP Package Maximum Weight (Preliminary)

SAM4E	900	mg

Table 47-11. LQFP Package Reference

JEDEC Drawing Reference	MS-026-C
JESD97 Classification	e3

Table 47-12. LQFP Package Characteristics

Moisture Sensitivity Level	3

This package respects the recommendations of the NEMI User Group.

47.5 Soldering Profile

Table 47-13 gives the recommended soldering profile from J-STD-020C.

Table 47-13. Soldering Profile

Profile Feature	Green Package
Average Ramp-up Rate (217°C to Peak)	3°C/sec. max.
Preheat Temperature 175°C ±25°C	180 sec. max.
Temperature Maintained Above 217°C	60 sec. to 150 sec.
Time within 5°C of Actual Peak Temperature	20 sec. to 40 sec.
Peak Temperature Range	260°C
Ramp-down Rate	6°C/sec. max.
Time 25°C to Peak Temperature	8 min. max.

Note: The package is certified to be backward compatible with Pb/Sn soldering profile.

A maximum of three reflow passes is allowed per component.

47.6 Packaging Resources

Land Pattern Definition.

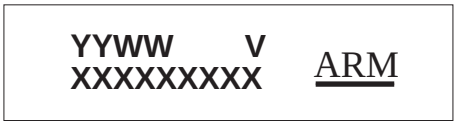
Refer to the following IPC Standards:

- IPC-7351A and IPC-782 (*Generic Requirements for Surface Mount Design and Land Pattern Standards*) http://landpatterns.ipc.org/default.asp
- Atmel Green and RoHS Policy and Package Material Declaration Datasheet available on www.atmel.com

48. 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

49. Ordering Information

Table 49-1. Ordering Codes for SAM4E Devices

Ordering Code	MRL	Flash (Kbytes)	RAM (Kbytes)	Package	Carrier Type	Operating Temperature Range
ATSAM4E16EA-CU	A	1024	128	LFBGA144	Tray	Industrial (-40°C to 85°C)
ATSAM4E16EA-CUR					Reel	
ATSAM4E16EB-CN	B				Tray	Industrial (-40°C to 105°C)
ATSAM4E16EB-CNR					Reel	
ATSAM4E16EA-AU	A			LQFP144	Tray	Industrial (-40°C to 85°C)
ATSAM4E16EA-AUR					Reel	
ATSAM4E16EA-AN	A				Tray	Industrial (-40°C to 105°C)
ATSAM4E16EA-ANR					Reel	
ATSAM4E16EB-AN	B				Tray	
ATSAM4E16EB-ANR					Reel	
ATSAM4E16CA-CU	A			TFBGA100	Tray	Industrial (-40°C to 85°C)
ATSAM4E16CA-CUR					Reel	
ATSAM4E16CB-CN	B				Tray	Industrial (-40°C to 105°C)
ATSAM4E16CB-CNR					Reel	
ATSAM4E16CA-AU	A			LQFP100	Tray	Industrial (-40°C to 85°C)
ATSAM4E16CA-AUR					Reel	
ATSAM4E16CA-AN	A				Tray	Industrial (-40°C to 105°C)
ATSAM4E16CA-ANR					Reel	
ATSAM4E16CB-AN	B				Tray	
ATSAM4E16CB-ANR					Reel	
ATSAM4E8EA-CU	A	512		LFBGA144	Tray	Industrial (-40°C to 85°C)
ATSAM4E8EA-CUR					Reel	
ATSAM4E8EB-CN	B				Tray	Industrial (-40°C to 105°C)
ATSAM4E8EB-CNR					Reel	
ATSAM4E8EA-AU	A			LQFP144	Tray	Industrial (-40°C to 85°C)
ATSAM4E8EA-AUR					Reel	
ATSAM4E8EA-AN	A				Tray	Industrial (-40°C to 105°C)
ATSAM4E8EA-ANR					Reel	
ATSAM4E8EB-AN	B				Tray	
ATSAM4E8EB-ANR					Reel	
ATSAM4E8CA-CU	A			TFBGA100	Tray	Industrial (-40°C to 85°C)
ATSAM4E8CA-CUR					Reel	
ATSAM4E8CB-CN	B				Tray	Industrial (-40°C to 105°C)
ATSAM4E8CB-CNR					Reel	

Table 49-1. Ordering Codes for SAM4E Devices (Continued)

Ordering Code	MRL	Flash (Kbytes)	RAM (Kbytes)	Package	Carrier Type	Operating Temperature Range
ATSAM4E8CA-AU	A	512	128	LQFP100	Tray	Industrial (-40°C to 85°C)
ATSAM4E8CA-AUR					Reel	
ATSAM4E8CA-AN	A				Tray	Industrial (-40°C to 105°C)
ATSAM4E8CA-ANR					Reel	
ATSAM4E8CB-AN	B				Tray	
ATSAM4E8CB-ANR					Reel	

50. Errata on SAM4E Devices

50.1 Errata SAM4E Rev. A Parts

The errata are applicable to the devices listed in the table below:

Table 50-1. Revision A parts

Chip Name	Revision	CHIPID_CIDR	CHIPID_EXID
SAM4E16E	A	0xA3CC_0CE0	0x0012_0200
SAM4E8E	A	0xA3CC_0CE0	0x0012_0208
SAM4E16C	A	0xA3CC_0CE0	0x0012_0201
SAM4E8C	A	0xA3CC_0CE0	0x0012_0209

50.1.1 Watchdog

50.1.1.1 Watchdog Not Stopped in Wait Mode

When the Watchdog is enabled and the bit WAITMODE = 1 is used to enter Wait mode, the watchdog is not halted. If the time spent in Wait mode is longer than the Watchdog time-out, the device will be reset if Watchdog reset is enabled.

Problem Fix/Workaround

When entering Wait mode, the Wait For Event (WFE) instruction of the processor Cortex-M4 must be used with the SLEEPDEEP bit of the System Control Register (SCB_SCR) of the Cortex-M = 0.

50.1.2 Brownout Detector

50.1.2.1 Unpredictable Behavior if BOD is Disabled, VDDCORE is Lost and VDDIO is Connected

In Active mode or in Wait mode, if the Brownout Detector is disabled (SUPC_MR.BODDIS = 1) and power is lost on VDDCORE while VDDIO is powered, the device might not be properly reset and may behave unpredictably.

Problem Fix/Workaround

When the Brownout Detector is disabled in Active or in Wait mode, VDDCORE always needs to be powered.

50.1.3 Flash

50.1.3.1 Flash: Incorrect Flash Read May Occur Depending on VDDIO Voltage and Flash Wait State

Flash read issues leading to wrong instruction fetch or incorrect data read may occur under the following operating conditions:

VDDIO < 2.4V and Flash wait state[(1)] ≥ 1

If the core clock frequency does not require the use of the Flash wait state [(2)] (FWS = 0 in EEFC_FMR), or if only data reads are performed on the Flash (e.g., if the code is running out of SRAM), there are no constraints on

VDDIO voltage. The usable voltage range for VDDIO is defined in Table 46-2 “DC Characteristics” in Section 46. “SAM4E Electrical Characteristics”.

Notes: 1. Defined in FWS field in EEFC_FMR.
2. See “Embedded Flash Characteristics” in Section 46. “SAM4E Electrical Characteristics” for the maximum core clock frequency at zero (0) wait state.

Problem Fix/Workaround

Two workarounds are available:

- Reduce the device speed to decrease the number of wait states to 0.
- Copy the code from Flash to SRAM at 0 wait states and then run the code out of SRAM.

50.1.4 Floating Point Unit (FPU)

50.1.4.1 FPU: IXC flag interrupt

The FPU exhibits six exceptions that are logically ORed and connected to the interrupt controller. If the IXC (Inexact result) flag occurrence is frequent, this leads to a very high rate of interrupts which severely affects FPU performance.

Problem Fix/Workaround

Disable the FPU Error interrupt. After each FPU operation, check whether an error occurred by polling the FPU Status register (FPSCR).

50.2 Errata SAM4E Rev.B Parts

The errata are applicable to the devices listed in the table below:

Table 50-2. Revision B parts

Chip Name	Revision	CHIPID_CIDR	CHIPID_EXID
SAM4E16E	B	0xA3CC_0CE1	0x0012_0200
SAM4E8E	B	0xA3CC_0CE1	0x0012_0208
SAM4E16C	B	0xA3CC_0CE1	0x0012_0201
SAM4E8C	B	0xA3CC_0CE1	0x0012_0209

50.2.1 Watchdog

50.2.1.1 Watchdog Not Stopped in Wait Mode

When the Watchdog is enabled and the bit WAITMODE = 1 is used to enter Wait mode, the watchdog is not halted. If the time spent in Wait mode is longer than the Watchdog time-out, the device will be reset if Watchdog reset is enabled.

Problem Fix/Workaround

When entering Wait mode, the Wait For Event (WFE) instruction of the processor Cortex-M4 must be used with the SLEEPDEEP bit of the System Control Register (SCB_SCR) of the Cortex-M = 0.

50.2.2 Brownout Detector

50.2.2.1 Unpredictable Behavior if BOD is Disabled, VDDCORE is Lost and VDDIO is Connected

In Active mode or in Wait mode, if the Brownout Detector is disabled (SUPC_MR.BODDIS = 1) and power is lost on VDDCORE while VDDIO is powered, the device might not be properly reset and may behave unpredictably.

Problem Fix/Workaround

When the Brownout Detector is disabled in Active or in Wait mode, VDDCORE always needs to be powered.

Table of Contents

51. Revision History

In the tables that follow, the most recent version of the document appears first.

Table 51-1. SAM4E Datasheet Rev. 11157H 31-Mar-2016 Revision History

Doc. Date	Changes
31-Mar-2016	Updated Figure 2-1. SAM4E 144-pin Block Diagram (replaced AFEx_AD0..11 with AFEx_AD0..14) Section 4. "Package and Pinout" GNDPLL, GNDCORE, GNDANA and GNDIO replaced with GND for BGA packages in Table 4-1 "SAM4E 100-ball TFBGA Pinout" and Table 4-2 "SAM4E 144-ball LFBGA Pinout" Adde note 2 to XIN32 and XOUT32 in Table 6-1 "System I/O Configuration Pin List" Section 11. "Cortex-M4 processor" : Removed SCB_AFSR in Section 11-33 "System Control Block (SCB) Register Mapping"
	Section 12. "Debug and Test Features": Updated Section 12.6.9 "IEEE® 1149.1 JTAG Boundary Scan"
	Section 13. "Reset Controller (RSTC)" Updated Section 13.4.3.3 "Watchdog Reset": Replaced "is set" with "is written to 1" and "is reset" with "is written to 0". Updated Section 13.4.2.1 "NRST Signal or Interrupt" Reworked Section 13.1 "Description" and Section 13.2 "Embedded Characteristics"
	Section 15. "Real-time Clock (RTC)" Updated Section 15.2 "Embedded Characteristics" Figure 15-5, "Calibration Circuitry Waveforms" corrected two instances of "3,906 ms" to "3.906 ms" Table 15-2 "Register Mapping": added offset 0xCC as reserved Section 15.6.1 "RTC Control Register": updated descriptions of value '0' for bits UPDTIM and UPDCAL and updated CALEVSEL bit description Deleted "All non-significant bits read zero." from the following registers: - Section 15.6.3 "RTC Time Register" - Section 15.6.4 "RTC Calendar Register" Reworked Figure 15-5, "Calibration Circuitry Waveforms" Modified Section 15.5.6 "Updating Time/Calendar"
	Section 16. "Watchdog Timer (WDT)" Replaced "Idle mode" with "Sleep mode (idle mode)" in Section 16.1 "Description" and with "Sleep mode" in Section 16.4 "Functional Description" Section 16.5.1 "Watchdog Timer Control Register": added note on modification of WDT_CR values. Section 16.5.2 "Watchdog Timer Mode Register": updated note on modification of WDT_MR values. Section 16.4 "Functional Description" and Section 16.5.2 "Watchdog Timer Mode Register" (WDDIS bit description) : modified information on WDDIS bit setting" Section 16.4 "Functional Description" : Modified paragraph starting with "The reload of the WDT must occur..."
	Section 20. "Enhanced Embedded Flash Controller (EEFC)" Section 20.4.3.6 "Calibration Bit": changed information on oscillators that are calibrated in production
	Section 21. "Fast Flash Programming Interface (FFPI)" Figure 21.3.3. Entering Parallel Programming Mode: deleted note on device clocking. Section 21.3 "Parallel Fast Flash Programming": in block diagrams, changed input source for XIN. Table 21-1 Signal Description List: deleted comment for XIN. Section 21.3.3 "Entering Parallel Programming Mode": reworded steps 2 and 3.

Table 51-1. SAM4E Datasheet Rev. 11157H 31-Mar-2016 Revision History

Doc. Date	Changes
31-Mar-2016	Section 22. “Cortex-M Cache Controller (CMCC)” Updated Section 22.2 “Embedded Characteristics”
	Section 25. “DMA Controller (DMAC)” Section 25.7 “DMAC Software Requirements”: deleted bullet referencing hardware handshake interface protocol Moved Section 25.6.7 “Register Write Protection” into Section 25.6 “Functional Description” Section 25.6.5 “Programming a Channel”: “DMAC_SARx, DMAC_DARx, DMAC_CTLx, and DMAC_LLPx” corrected to “DMAC_SADDRx, DMAC_DADDRx, DMAC_CTRLAx, DMAC_CTRLBx, and DMAC_DSCRx” Section 25-3 “Multiple Buffers Transfer Management”: - added links to footnotes - deleted footnote “Channel stalled is true if the relevant BTC interrupt is not masked
	Section 27. “Static Memory Controller (SMC)” Modified Figure 27-3 “NAND Flash Signal Multiplexing on SMC Pins” and added Note 1 below the figure Section 27.10 “Scrambling/Unscrambling Function”: added details on access for SMC_KEY1 and SMC_KEY2 registers. Section 27.16.6 “SMC Off-Chip Memory Scrambling Key1 Register” and Section 27.16.7 “SMC Off-Chip Memory Scrambling Key2 Register”: added Note (1) to clarify Write-once access
	Section 28. “Clock Generator” Section 29.17 “Register Write Protection”: added “PMC Clock Generator Main Clock Frequency Register” to list of protectable registers Updated Figure 28-1 “Clock Generator Block Diagram” Section 29.11 “Fast Startup”: inserted warning “The duration of the WKUPx pins active level must be greater than four main clock cycles.”
	Section 34. “Serial Peripheral Interface (SPI)” Section 34.8.1 “SPI Control Register”: added bit REQCLR Section 34-5 “Register Mapping”: for Chip Select Register, replaced fixed offset with equation Modified transmission condition description in Section 34.7.3 “Master Mode Operations”
	Section 37. “Universal Synchronous Asynchronous Receiver Transmitter (USART)” Section 8.6 “USART Interrupt Enable Register (SPI_MODE)”: added bit NSSE (register bit 19) in Section 37.7.6 “USART Interrupt Enable Register (SPI_MODE)”, Section 37.7.8 “USART Interrupt Disable Register (SPI_MODE)”, and Section 37.7.10 “USART Interrupt Mask Register (SPI_MODE)”. Section 37.7.12 “USART Channel Status Register (SPI_MODE)”: added bit NSSE (register bit 19) and bit NSS (register bit 23). Section 37-2 “Baud Rate Generator”: added label “Selected Clock” to USCLKS multiplexer output and corrected value in "The frequency of the signal provided on SCK must be at least... Section “Baud Rate Calculation Example”: in baud rate calculation formula, replaced “$f_{peripheral\ clock}$” with “Selected Clock” Figure 37-3, “Fractional Baud Rate Generator”: added label “Selected Clock” to USCLKS mux output Section 37.6.1.3 “Baud Rate in Synchronous Mode or SPI Mode”: in second paragraph, replaced “$f_{peripheral\ clock}$” with “Selected Clock” At end of Section 37.6.1.2 “Fractional Baud Rate in Asynchronous Mode”, added warning “When the value of field FP is greater than 0...” Cont’d

Table 51-1. SAM4E Datasheet Rev. 11157H 31-Mar-2016 Revision History

Doc. Date	Changes
31-Mar-2016	Section 37.7.15 “USART Baud Rate Generator Register”: added warning “When the value of field FP is greater than 0...” to FP field description: Section 37.6.3.4 “Manchester Decoder”: corrected “MANE flag” with “MANERR” flag. Section 37.6.8.5 “Character Transmission”: corrected bit names: RTSEN to RCS, RTSDIS to FCS and added content “An additional condition...on the receiver side”. Section 37.7.1 “USART Control Register”: updated RTSDIS bit description. Section 37.7.3 “USART Mode Register”: updated descripiton for row 0xE, SPI_MASTER
	Section 38. “Timer Counter (TC)” Section 38.2 “Embedded Characteristics”: rephrased "Total number of TC channels" to read "Total number of TC channels implemented on this device Reformatted and renamed Table 38-2 “Channel Signal Description” Section 38.6.3 “Clock Selection”: updated notes (1) and (2) Section 38.6.9 “Transfer with PDC in Capture Mode”: added “in Capture mode” and updated Figure 38-5. Example of Transfer with PDC in Capture Mode Replaced TIOA, TIOB, TCLK with TIOAx, TIOBx, TCLK Section 38.6.16.4 “Position and Rotation Measurement”: added sentence on clearing the internal counter Added Section 38.6.16.6 “Detecting a Missing Index Pulse”
	Section 39. “Pulse Width Modulation Controller (PWM)” Updated Figure 39-1, “Pulse Width Modulation Controller Block Diagram” Added reference to Section 39.5.4 “Fault Inputs” in register descriptions Updated Section 39.6.2.2 “Comparator” Corrected PWM period formulas in Section 39.7.43 “PWM Channel Period Register” and Section 39.7.44 “PWM Channel Period Update Register” Section 39.6.5.1 “Initialization”: modified “Enable of the interrupts...” list item
	Section 42. “Ethernet MAC (GMAC)” Updated Section 42.1 “Description” Section 42.5.2 “Power Management”: deleted reference to PMC_PCER. Section 42.5.3 “Interrupt Sources”: deleted reference to ‘Advanced Interrupt Controller’. Replaced by ‘interrupt controller’. Section 42.6.13 “IEEE 1588 Support”: deleted reference to GMAC_TSS and removed reference to ‘output pins’ in 2nd paragraph Section 42.7.1.2 “Receive Buffer List” and Section 42.7.1.3 “Transmit Buffer List”: added note at end of sections on queue pointer intialization.

Table 51-1. SAM4E Datasheet Rev. 11157H 31-Mar-2016 Revision History

Doc. Date	Changes
31-Mar-2016	Section 43. “Analog Front-End Controller (AFEC)” Section 43.7.13 “AFEC Interrupt Status Register”: defined EOCAL bit as ‘cleared on read’ Section 43.6.1 “Analog Front-End Conversion”: updated section and added equations for AFE conversion time. Updated Section 43-2 “Sequence of AFE Conversions when Tracking Time > Conversion Time” Added sentence on write protection below the register table for: Section 43.7.21 “AFEC Channel Offset Compensation Register” Section 43.7.22 “AFEC Temperature Sensor Mode Register” Section 43.6.16 “Register Write Protection”: added ”AFEC Channel Differential Register” to the list of write-protected registers. Section 43.7.2 “AFEC Mode Register”: updated descriptions of fields TRACKTIM and TRANSFER Updated Warning in Section 43.6.10 “AFE Timings” Section 43.2 “Embedded Characteristics”: deleted bullet on conversion rate (redundant with Electrical Characteristics) Deleted Section 7.5 ”Conversion Results Format”. Section 43.6.7 “Comparison Window”: deleted paragraph on conversion sign. Section 43.7.3 “AFEC Extended Mode Register” bits 28 and 29 now reserved (were ‘SIGNMODE’) Section 43.7.21 “AFEC Channel Offset Compensation Register”: added note on configuration of AOFF. Section 43.7.19 “AFEC Channel Selection Register”: updated CSEL bit description. Section 43.6.9 “Input Gain and Offset”: updated information on AOFF field.
	Section 31. “Controller Area Network (CAN)” Updated MIDvA and MIDvB description in Section 31.9.15 “CAN Message Acceptance Mask Register” Updated Section 31.6.3 “Interrupt Sources” (replaced “Advanced Interrupt Controller” with ”interrupt controller”) and Figure 9-1, “Possible Initialization Procedure” (replaced “AIC” with “Interrupt Controller”) Section 31.8.3.2 “Transmission Handling”: in sixth paragraph, “CAN_MACR” remplaced with “CAN_ACR
	Section 46. “SAM4E Electrical Characteristics” Section 46.11.8.3 “MII Mode” : Removed note 1 below Table 46-66 “EMAC MII Timings” Deleted $t_{TRACKTIM}$ and t_s in Table 46.7.3 “ADC Timings” Modified Section 46.11.3 “SPI Characteristics”

Table 51-2. SAM4E Datasheet Rev. 11157G 12-Feb-2016 Revision History

Doc. Date	Changes
12-Feb-2016	Added MRLB (Rev. B) devices: - Updated Table 32-1 “SAM4E Chip ID Registers”. - Updated Section 50. “Errata on SAM4E Devices” (added Section 50.1.4 “Floating Point Unit (FPU)”, Section 50.2 “Errata SAM4E Rev.B Parts” and CHIPID information). - Table 49-1, “Ordering Codes for SAM4E Devices”.

Table 51-3. SAM4E Datasheet Rev. 11157F Revision History

Doc. Date	Changes
27-Apr-15	Editorial and formatting changes throughout Deleted reset values and/or address offsets from individual register description sections (information found in “Register mapping” tables)
	Modified Section 1. “Features” and Figure 2-1, “SAM4E 144-pin Block Diagram” Updated Table 10-1, “Peripheral Identifiers”
	Section 11., “ARM Cortex-M4 Processor” Updated Table 11-11 “Faults” Table 11-35 “System Timer (SYST) Register Mapping”: corrected SYST_CSR reset value Modified Figure 11-1 “Typical Cortex-M4F Implementation”
	Section 12., “Debug and Test Features” Updated Section 12.6.3 “ERASE Pin”
	Section 13., “Reset Controller (RSTC)” Updated Section 13.4.1, “Reset Controller Overview”, Section 13.5.2, “Reset Controller Status Register” and Section 13.5.3, “Reset Controller Mode Register”
	Section 14. “Real-time Timer (RTT)” Modified Section 14.4 “Functional Description” Updated Section 14.5.3 “Real-time Timer Value Register” and Section 14.5.4 “Real-time Timer Status Register”
	Section 15., “Real-time Clock (RTC)” Modified Section 15.3 “Block Diagram” Updated “Section 15.1 “Description” and Section 15.5 “Functional Description” (removed references to the 20th century) Section 15.5.5 “RTC Internal Free Running Counter Error Checking”: replaced “RTC status clear control register” with “Status Clear Command Register” Modified Section 15.5.7 “RTC Accurate Clock Calibration” Added TDERR field in Section 15.6.11 “RTC Interrupt Mask Register”
	Section 16., “Watchdog Timer (WDT)” Modified Figure 16-2 “Watchdog Behavior” and Section 16.5.3 “Watchdog Timer Status Register”
	Section 17., “Reinforced Safety Watchdog Timer (RSWDT)” Updated Section 17.1 “Description”, Section 17.2 “Embedded Characteristics” and Section 17.4 “Functional Description”
	Section 18., “Supply Controller (SUPC)” Updated Figure 18-1 “Supply Controller Block Diagram”. Modified Section 18.4.2, “Slow Clock Generator” Updated Section 18.5.5, “Supply Controller Mode Register”, Section 18.5.8, “Supply Controller Status Register” and Section 18.5.7, “Supply Controller Wake-up Inputs Register” Section 18.4.7.3, “Low-power Tamper Detection and Anti-Tampering” Modified Figure 18-4 “Wake-up Sources” and added a paragraph and Figure 18-5 “Entering and Exiting Backup Mode with a WKUP Pin” in Section 18.4.7.2, “Wake-up Inputs”

Table 51-3. SAM4E Datasheet Rev. 11157F Revision History (Continued)

Doc. Date	Changes
27-Apr-15	Section 1. “Enhanced Embedded Flash Controller (EEFC)” Updated Table 1-8 “Register Mapping” Updated Section 1.3.2 “Interrupt Sources” Updated Section 1.4.3.2 “Write Commands” Updated Section 1.5.1 “EEFC Flash Mode Register”, Section 1.5.2 “EEFC Flash Command Register” and Section 1.5.3 “EEFC Flash Status Register”, Updated Figure 1-3 “Code Read Optimization for FWS = 0” and Figure 1-4 “Code Read Optimization for FWS = 3”. Modified Section 1.4.3.3 “Erase Commands”, Section 1.4.3.8 “Unique Identifier Area”, Section 1.4.3.9 “User Signature Area” and Section 1-7 “Command State Chart”
	Section 22. “Cortex-M Cache Controller (CMCC)” Modified Section 22.4.3 ”Cache Performance Monitoring”, Section 22.5.1 ”Cache Controller Type Register” and Section 22.5.2 ”Cache Controller Configuration Register” Section 22.5.7 ”Cache Controller Monitor Configuration Register”: changed access from Write-only to Read/Write. Updated Section 22.5.3 ”Cache Controller Control Register”, Section 22.5.4 ”Cache Controller Status Register”, Section 22.5.5 ”Cache Controller Maintenance Register 0”, Section 22.5.8 ”Cache Controller Monitor Enable Register” and Section 22.5.9 ”Cache Controller Monitor Control Register”.
	Section 25. “DMA Controller (DMAC)” Updated Section 25.2 “Embedded Characteristics”: added bullet “Register Write Protection” Added Section 25.5 “Product Dependencies” Modified Section 25.6.3.1 “Software Handshaking” Updated Section 25.6.6 “Disabling a Channel Prior to Transfer Completion” and Section 25.6.6.1 “Abnormal Transfer Termination” Modified Section 25.8 “Register Write Protection” Updated Table 25-4, “Register Mapping”: replaced reserved offset range “0x01EC- 0x1FC” with “0x1EC–0x1FC” Updated Section 25.9.19 “DMAC Write Protection Mode Register” and Section 25.9.20 “DMAC Write Protection Status Register”
	Section 26., “Peripheral DMA Controller (PDC)” Modified Section 26.5.10, “Transfer Status Register”
	Section 27., “Static Memory Controller (SMC)” Updated Table 27-1 “I/O Line Description”. Updated Section 27.9.5 “Register Write Protection” and added information on write protection to Section 27.16.1 “SMC Setup Register”, Section 27.16.2 “SMC Pulse Register”, Section 27.16.3 “SMC Cycle Register” and Section 27.16.5 “SMC OCMS Mode Register”. Updated Section 27.16.8 “SMC Write Protection Mode Register” and Section 27.16.9 “SMC Write Protection Status Register”. Updated Section 27.10 “Scrambling/Unscrambling Function”.

Table 51-3. SAM4E Datasheet Rev. 11157F Revision History (Continued)

Doc. Date	Changes
27-Apr-15	**Section 28. "Clock Generator"**. Modified Section 28.2 "Embedded Characteristics" and Section 29.2 "Embedded Characteristics". Added Section 28.5.5 "Bypassing the 3 to 20 MHz Crystal Oscillator". Updated Section 28.4.2 "32768 Hz Crystal Oscillator" Updated Section 28.5.6 "Main Clock Frequency Counter" and Section 28.5.7 "Switching Main Clock between the RC Oscillator and the Crystal Oscillator" Modified Section 28.6.1 "Divider and Phase Lock Loop Programming" Modified Table 29-2, "Register Mapping" Modified Section 29.7 "USB Clock Controller", Section 29.11 "Fast Startup" and Section 29.13 "Main Clock Failure Detector" Updated Section 29.17.7 "PMC Clock Generator Main Oscillator Register", Section 29.17.8 "PMC Clock Generator Main Clock Frequency Register" and Section 29.17.9 "PMC Clock Generator PLLA Register"
	Section 30. "Advanced Encryption Standard (AES)" Section 30.4.2 "Operation Modes": updated content relative to 1 megabyte limitation Modified Section 30.4.5.1 "Manual and Auto Modes" Modified Section 30.5.6 "AES Interrupt Status Register"
	Section 32. "Chip Identifier (CHIPID)" Section 32.3.1 "Chip ID Register": Updated EPROC, SRAMSIZ and NVPSIZ descriptions
	Section 31. "Controller Area Network (CAN)" Modified Table 31-6, "Register Mapping" Modified Section 31.9.1 "CAN Mode Register", Section 31.9.5 "CAN Status Register", Section 31.9.12 "CAN Write Protection Mode Register", Section 31.9.13 "CAN Write Protection Status Register", Section 31.9.18 "CAN Message Status Register" and Section 31.9.21 "CAN Message Control Register"
	Section 33., "Parallel Input/Output Controller (PIO)" Section 33.6.38 "PIO Additional Interrupt Modes Mask Register": modified P0–P31 bit description Modified Figure 33-1 "Block Diagram" and Figure 33-9 "PIO Controller Connection with CMOS Digital Image Sensor" Replaced instances of "div_slclk" with "div_slck"; replaced instances of "slow_clock" with "slck" Modified Table 33-5 "Register Mapping" Removed section "External Interrupt Lines" and Figure 4-4 Application Block Diagram. Updated Figure 33-10 "Parallel Capture Mode Waveforms (DSIZE = 2, ALWYS = 0, HALFS = 0)",Figure 33-11 "Parallel Capture Mode Waveforms (DSIZE = 2, ALWYS = 1, HALFS = 0)", Figure 33-12 "Parallel Capture Mode Waveforms (DSIZE = 2, ALWYS = 0, HALFS = 1, FRSTS = 0)" and Figure 33-13 "Parallel Capture Mode Waveforms (DSIZE = 2, ALWYS = 0, HALFS = 1, FRSTS = 1)" Updated Section 33.5.16 "Register Write Protection" and Section 33.6.32 "PIO Pad Pull-Down Status Register":" and Section 33.5.3 "Peripheral A or B or C or D Selection"
	Section 34. "Serial Peripheral Interface (SPI)" Updated Figure 34.3. Block Diagram and Figure 34-7. Master Mode Flow Diagram Section 34.7.3.6 "SPI Peripheral DMA Controller (PDC)" ("Transfer size"): replaced "8-bit to 16-bit data" with "9-bit to 16-bit data" Modified Section 34.7.3.5 "Peripheral Selection" and Section 34.7.3.9 "Peripheral Deselection without DMA nor PDC" Updated Section 34.7.1 "Modes of Operation", Section 34.7.3 "Master Mode Operations", Section 34.8.1 "SPI Control Register", Section 34.8.2 "SPI Mode Register", Section 34.8.5 "SPI Status Register" and Section 34.8.9 "SPI Chip Select Register"

Table 51-3. SAM4E Datasheet Rev. 11157F Revision History (Continued)

Doc. Date	Changes
27-Apr-15	Section 35. “Two-wire Interface (TWI)” Modified Section 35.1 “Description” Modified Table 35-1, “Atmel TWI Compatibility with I2C Standard” Modified Section “Clock Synchronization Sequence” and added Section “Clock Stretching Sequence” Section 35.7.3.3 “Programming Master Mode”: replaced all occurrences of "TWIHS_" with "TWI_". Replaced instance of acronym “TWI2” with “TWI” Section 35.8 “Two-wire Interface (TWI) User Interface” Replaced Sections “Application Block Diagram” with updated Section 35.5 “I/O Lines Description” Updated Figure 35-9 “Master Read Wait State with Multiple Data Bytes”, Figure 35-23. Read Access Ordered by a Master, Figure 35-24. Write Access Ordered by a Master, Figure 35-25. Master Performs a General Call Figure 35-30. Read Write Flowchart in Slave Mode, Figure 35-27. Clock Synchronization in Write Mode and Figure 35-30. Read Write Flowchart in Slave Mode Replaced all instances of “(Optional) Wait for the TXCOMP flag in TWI_SR before disabling the peripheral clock if required.” with “(Only if peripheral clock must be disabled) Wait for the TXCOMP flag to be raised in TWI_SR” Modified Section 35.7.3.3 “Master Transmitter Mode” and Section “Read Sequence” (note on clearing TXRDY flag) Section 35.7.3.4 “Master Receiver Mode”: Modified “Warning”. Modified Section 35.8.5 “TWI Clock Waveform Generator Register” and Section 35.8.6 “TWI Status Register”
	Section 36. “Universal Asynchronous Receiver Transmitter (UART)” Updated Section 36.5.1 “Baud Rate Generator”, Section 36.5.2.4 “Receiver Overrun” and Section 36.6.9 “UART Baud Rate Generator Register”
	Section 37. “Universal Synchronous Asynchronous Receiver Transceiver (USART)” Updated Section 37.2 “Embedded Characteristics”, Section 37.3 “Block Diagram” and Section 37.6.10 “Register Write Protection” Modified Section 37.5.1 “I/O Lines” and Table 37-15 “Register Mapping” Modifed Section 37.7.1 “USART Control Register”, Section 37.7.3 “USART Mode Register”, Section 37.7.4 “USART Mode Register (SPI_MODE)”, Section 37.7.5 “USART Interrupt Enable Register”, Section 37.7.6 “USART Interrupt Enable Register (SPI_MODE)”, Section 37.7.7 “USART Interrupt Disable Register”, Section 37.7.8 “USART Interrupt Disable Register (SPI_MODE)”Section 37.7.9 “USART Interrupt Mask Register”, Section 37.7.10 “USART Interrupt Mask Register (SPI_MODE)”Section 37.7.11 “USART Channel Status Register”, and Section 37.7.12 “USART Channel Status Register (SPI_MODE)”, Section 37.7.15 “USART Baud Rate Generator Register”, Section 37.7.16 “USART Receiver Time-out Register”, Section 37.7.17 “USART Transmitter Timeguard Register” and Section 37.7.18 “USART FI DI RATIO Register” Updated symbols used to express time to JEDEC standards Section 37.6.3.3 “Asynchronous Receiver” Section 37.6.1 “Baud Rate Generator”, Section 37.6.4 “ISO7816 Mode” Section “Transmit Character Repetition”: replaced “ITERATION bit” with “ITER bit” Removed RXIDLEV bit (bit 31 now reserved) from “USART Manchester Configuration Register” Modified information on Hardware handshaking

Table 51-3. SAM4E Datasheet Rev. 11157F Revision History (Continued)

Doc. Date	Changes
27-Apr-15	Section 38. “Timer Counter (TC)” Modified Section 38.1 “Description”, Section 38.5.2 “Power Management” and Section 38.6.19 “Register Write Protection” Moved Table 38-1, “Timer Counter Clock Assignment” Modified ‘Name’ line in Section 38.7.2 “TC Channel Mode Register: Capture Mode” and Section 38.7.3 “TC Channel Mode Register: Waveform Mode” Modified Section 38.7.10 “TC Status Register”, Section 38.7.14 “TC Extended Mode Register”Section 38.7.16 “TC Block Mode Register”, Section 38.7.20 “TC QDEC Interrupt Status Register” Section 38.7.22 “TC Write Protection Mode Register” Modified Section 38.5.3 “Interrupt Sources” and Section 38.6.14 “Synchronization with PWM”, Section 38.6.16.4 “Position and Rotation Measurement”, Section 38.6.16.5 “Speed Measurement” Section 38.6.16 “Quadrature Decoder”: removed subsection “Missing Pulse Detection and Auto-correction”
	Section 39. “Pulse Width Modulation Controller (PWM)” Modified Section “Method 3: Automatic write of duty-cycle values and automatic trigger of the update” Updated Section 39.2 “Embedded Characteristics” throughout: corrected register name PWM_CPRx to PWM_CPRDx (PWM_CPRx does not exist) Section 39.5.3 “Interrupt Sources” Section 39.6 “Functional Description” Section 39.6.2.9 “Synchronous Channels” Section 39.7.24 “PWM Fault Mode Register”, Section 39.7.25 “PWM Fault Status Register”: changed field descriptions. Section 39.7.26 “PWM Fault Clear Register”, Section 39.7.28 “PWM Fault Protection Enable Register” Section 39.7.40 “PWM Channel Mode Register” Figure 39-17 “Comparison Unit Block Diagram” Table 39-7 “Register Mapping”: deleted reset value from PWM_SCUPUPD (this register is write-only) Added Figure 39-20 “Event Line Generation Waveform (Example)” Section 39.6.4 “PWM Event Lines”: in first sentence, replaced “i.e.” with “e.g.”
	Section 40. “High Speed Multimedia Card Interface (HSMCI)” Modified Section 40.1 “Description”: removed sentence “Only one slot can be selected at a time (slots are multiplexed)” Added Section 40.14.19 “HSMCI FIFOx Memory Aperture” Updated Section 40.14.12 “HSMCI Status Register” Updated Table 40-4, “Bus Topology” (4-bit instead of 8-bit)
	Section 41. “USB Device Port (UDP)” Table 41-6 Register Mapping: update footnote No. 1. Modified Section 41.7.4, “UDP Interrupt Enable Register” Section , “Using Endpoints With Ping-pong Attribute”: Replaced Bank 0 with Bank 1 in step 12.
	Section 42. “Ethernet MAC (GMAC)” Modified Section 42.2 “Embedded Characteristics”, Section 42.5.3 “Interrupt Sources” Modified Section 42.3 “Block Diagram” Added Section 42.5 “Product Dependencies”. Modified Section 42.6.3.1 “Receive AHB Buffers” and Section 42.6.3.2 “Transmit AHB Buffers”, Section 42.6.6.1 “Receiver Checksum Offload”, Section 42.6.15.2 “802.3 Pause Frame Transmission”, Section 42.6.16.2 “PFC Pause Frame Transmission”

Table 51-3. SAM4E Datasheet Rev. 11157F Revision History (Continued)

Doc. Date	Changes
27-Apr-15	Updated Section 42.6.4 “MAC Transmit Block”, Section 42.6.5 “MAC Receive Block”, Section 42.6.7 “MAC Filtering Block”, Section 42.6.13 “IEEE 1588 Support” and Section 42.6.14 “Time Stamp Unit” Modified Section 42.7.1.8 “Receiving Frames” Modified “GMAC Type ID Match x Registers” descriptions Modified Section 42.8 “Ethernet MAC (GMAC) User Interface” Modified Table 42-17 Register Mapping and updated register description sections accordingly Updated Section 42.8.1 “GMAC Network Control Register”, Section 42.8.6 “GMAC Transmit Status Register”, , Section 42.8.9 “GMAC Receive Status Register”, Section 42.8.13 “GMAC Interrupt Mask Register” and Section 42.8.14 “GMAC PHY Maintenance Register”
	Section 45. “Analog Comparator Controller (ACC)” Section 45.2 “Embedded Characteristics”: Changed ADVREF to ‘External Voltage Reference’ Updated Figure 45-1 “Analog Comparator Controller Block Diagram” and Table 45-1, “ACC Signal Description” Modified Section 45.5.1 “I/O Lines” and Section 45.7.2 “ACC Mode Register”, Section 45.7.6 “ACC Interrupt Status Register” Corrected signal names Removed Table “List of External Analog Data Inputs” and note referring to this table. Changed all occurrences of ‘MCK’ to ‘peripheral clock’.
	Section 43. “Analog Front-End Controller (AFEC)” Updated Figure 43-1 “Analog Front-End Controller Block Diagram” Modified Section 43.5.1 “I/O Lines”, Section 43.5.2 “Power Management” and Section 43.6.12 “Temperature Sensor” Section 43.6.4 “Conversion Results” and Section 43.7.7 “AFEC Channel Disable Register”: updated warning text Section 43.6.8 “Comparison Window”: Replaced CPM_ALL bit name with CMPALL. Modified Section 43.6.13 “Enhanced Resolution Mode and Digital Averaging Function” and Section 43.6.14 “Automatic Calibration” Changed ‘AFEC Clock’ to ‘AFE clock’ and MCK to ‘peripheral clock’. Modified Figure 43-4 “EOCx and DRDY Flag Behavior” and Figure 43-5 “EOCx, GOVRE and OVREx Flag Behavior” Removed “Section 7.3.1 Enhanced Resolution Mode” Updated Section 43.7.2 “AFEC Mode Register”, Section 43.7.13 “AFEC Interrupt Status Register” and Section 43.7.20 “AFEC Channel Data Register”
	Section 44. “Digital-to-Analog Converter Controller (DACC)” Modified Figure 44-1 “DACC Block Diagram” , Section 44.2 “Embedded Characteristics” and Table 44-3, “Register Mapping” Removed references to Sleep mode and refresh period Modified Section 44.6.6 “DACC Timings” Updated Section 44.7.2 “DACC Mode Register”, Section 44.7.12 “DACC Write Protection Mode Register”, Section 44.7.13 “DACC Write Protection Status Register”
	Section 46. “SAM4E Electrical Characteristics” Updated Table 46-18 “32.768 kHz Crystal Oscillator Characteristics” (removed C_{LEXT}) Updated Table 46-2 “DC Characteristics” (conditions for V_{OL} and V_{OL})

Table 51-4. SAM4E Datasheet Rev. 11157E Revision History

Doc. Date	Changes
13-Feb-15	Editorial and formatting changes throughout
	"Description": Corrected "nine general-purpose 16-bit Timers" to "3 three-channel general-purpose 32-bit timers" Replaced "one RTC" with "a low-power RTC, a low-power RTT, 256-bit General Purpose Backup Registers" Added paragraph relating to low-power modes Added paragraph relating to Real-time Event Management
	Section 1. "Features" Updated description of "Low-power Modes" Under "Peripherals": - changed "32-bit Real-time Timer 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": renamed "EMAC" to "GMAC"
	Section 2. "Block Diagram" Removed Figure "SAM4E 100-pin Block Diagram" Inserted sentence "See Table 1-1 for detailed configurations of memory size, package and features of the SAM4E devices" Revised Figure 2-1 "SAM4E 144-pin Block Diagram"
	Section 3. "Signal Description" Table 3-1 "Signal Description List": removed Ethernet MAC signals GREFCK, GTXDV, and GCRSDV; redistributed links to footnote 1
	Section 5. "Power Considerations" Table 5-1 "Low-power Mode Configuration Summary": replaced "Backup Registers" with "GPBR" in column header Added Section 5.2 "Power-up Considerations" Figure 5-2 "Single Supply": - changed main supply range from 1.8V-3.6V to 1.62–3.6 V - renamed "ADC" to "AFEC" - below figure, added Analog Comparator to AFEC/DAC restrictions Figure 5-3 "Core Externally Supplied": - renamed "ADC" to "AFEC" - below figure, added Analog Comparator to AFEC/DAC restrictions Section 5.6.3 "Sleep Mode": added "with bit LPM = 0 in PMC_FSMR" to end of second paragraph Section 5.7 "Wake-up Sources": removed figure "Wake-up Source" Section 5.8 "Fast Start-up": in last sentence, changed "switches the master clock on this 4 MHz clock" to "switches the master clock on this 4 MHz clock by default"; removed figure "Fast Start-up Sources"
	Section 6. "Input/Output Lines" Section 6.2 "System I/O Lines": rewrote first paragraph Table 6-1 "System I/O Configuration Pin List": changed column header "SYSTEM_IO Bit Number" to "CCFG_SYSIO Bit No."

Table 51-4. SAM4E Datasheet Rev. 11157E Revision History (Continued)

Doc. Date	Changes
13-Feb-15	Section 7. “Memories” Inserted Section 7.1 “Product Mapping” (was previously section 7. “Product Mapping”) Figure 7-1 “SAM4E Product Mapping”: renamed “EFC” to “EEFC”; removed reserved space block between addresses 0x400E1600 and 0x400E1800 of System Controller map
	Section 8. “Real-time Event Management” Section 8.1 “Embedded Characteristics”: renamed instance of “ADC” to “AFEC” Revised Table 8-1 “Real-time Event Mapping List”
	Section 9. “System Controller” Deleted first two sentences “The System Controller is a set of peripherals...” and “See the system controller block diagram...” Removed Figure 10-1. “System Controller Block Diagram” **Removed Section 10.3 “Reset Controller” (reset controller is described in** Section 13. “Reset Controller (RSTC)”**)**
	Section 10. “Peripherals” Table 10-1 “Peripheral Identifiers”: renamed “EFC” to “EEFC”; renamed “EMAC” to “GMAC”; updated instance descriptions Table 10-2 “Multiplexing on PIO Controller A (PIOA)”: added footnotes providing information on selecting extra functions and system functions Table 10-3 “Multiplexing on PIO Controller B (PIOB)”: added footnotes providing information on selecting extra functions and system functions Table 10-4 “Multiplexing on PIO Controller C (PIOC)”: added footnotes providing information on selecting extra functions Table 10-5 “Multiplexing on PIO Controller D (PIOD)”: - removed signal GREFCK from PD0/Peripheral A - removed signal GCRSDV from PD4/Peripheral A
	Section 12. “Debug and Test Features” **Section 12.6.1 “Test Pin”**: at end of section, deleted sentence “For more on the manufacturing and test mode, refer to the “Debug and Test” section of the product datasheet” **Section 12.6.4 “Debug Architecture”**: in first paragraph, corrected “Cortex-M4 embeds four functional units” to “Cortex-M4 embeds five functional units” **Section 12.6.5 “Serial Wire JTAG Debug Port (SWJ-DP) Pins”**: - in second paragraph, deleted sentence “Please refer to the “Debug and Test” section of the product datasheet.” - in sixth paragraph, deleted sentence “For more information about SW-DP and JTAG-DP switching, please refer to the “Debug and Test” section of the datasheet.”
	Section 23. “SAM-BA Boot Program for SAM4E Microcontrollers” Section 23.6 “SAM-BA Monitor”**: rephrased introductory sentence** **Section 23.6.3 “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 23.6.4 “In Application Programming (IAP) Feature”:** replaced two instances of “MC_FSR register” with “EEFC_FSR”
	Section 24. “Bus Matrix (MATRIX)” Table 24-2 “Master to Slave Access”: inserted “PDC0” as name of master 2

Table 51-4. SAM4E Datasheet Rev. 11157E Revision History (Continued)

Doc. Date	Changes
13-Feb-15	Section 27. “Static Memory Controller (SMC)” Section 27.1 “Description”**:** replaced instance of “CM4P2” with “SAM4E” Section 27.7.1 “Implementation Examples”**:** replaced instance of “CM4P2” with “SAM4E”
	Section 46. “SAM4E Electrical Characteristics” Updated and harmonized parameter symbols Table 46-2 “DC Characteristics”: updated footnotes Table 46-3 “1.2V Voltage Regulator Characteristics”: replaced two footnotes with single footnote in V_{DDIN} conditions; deleted “Cf. External Capacitor Requirements” from CD_{IN} and CD_{OUT} conditions Table 46-4 “Core Power Supply Brownout Detector Characteristics”: added parameter “Reset Period” Table 46-7 “Zero-Power-On Reset Characteristics”: modified parameter name “Reset Time-out Period” to “Reset Period” Section 46.3.2.1 “Sleep Mode”: deleted sentence “Table 47-10 shows the current consumption in typical conditions” Figure 46-6 “Current Consumption in Sleep Mode (AMP1) versus Master Clock Ranges (refer to Table 46-10)”: replaced comma with dot as decimal separator in mA values Table 46-15 “Power Consumption on VDDCORE (VDDIO = 3.3V, VDDCORE = 1.08V, TA = 25°C)”: renamed peripheral “EMAC” to “GMAC” Table 46-18 “32.768 kHz Crystal Oscillator Characteristics”: added parameter “Allowed Crystal Capacitance Load” Figure 46-11 “32.768 kHz Crystal Oscillator Schematics”: added label “$C_{crystal}$” Table 46-20 “3 to 20 MHz Crystal Oscillator Characteristics”: removed parameter “Maximum External Capacitor on XIN and XOUT”; added parameter “Allowed Crystal Capacitance Load” Table 46-22 “XIN Clock Electrical Characteristics (In Bypass Mode)”: added parameters “Internal Parasitic Capacitance During Standby” and “Internal Parasitic Resistance During Standby” Added Figure 46-13 “XIN Clock Timing” Table 46-27 “Analog Power Supply Characteristics”: redirected link in first footnote to section “Low Voltage Supply” (was linked to section “ADC Channel Input Impedance”) Table 46-29 “ADVREF Electrical Characteristics”: redirected link in first footnote to section “Low Voltage Supply” (was linked to section “ADC Channel Input Impedance”) Figure 46-19 “Simplified Acquisition Path”: added caption “ADC Input”; replaced caption “12-bit ADC Core” with “12-bit ADC” Added “Symbol” column to Table 46-50 “Static Performance Characteristics”, Table 46-51 “Dynamic Performance Characteristics”, Table 46-52 “Analog Outputs”, and Table 46-53 “Analog Comparator Characteristics” Section 46.10 “Temperature Sensor”: specified instances of “27°C” as ambient temperature Table 46-54 “Temperature Sensor Characteristics”: deleted “After TSON = 1” from Startup Time conditions Section 46.11.3.1 “Maximum SPI Frequency”: - replaced “frequency above the pin FreqMax value” with “frequency above the maximum pad speed” in “Master Write Mode” - updated content in “Master Read Mode” - replaced “25 MHz” with “21 MHz” in “Slave Write Mode” Table 46-57 “SPI Timings”: removed footnotes defining 1.8V and 3.3V domains (this information is now found at the beginning of Section 46.11.3.2 “SPI Timings”) Section 46.11.5 “SMC Timings”: in timings tables, removed footnotes defining 1.8V and 3.3V domains (this information is already provided at the beginning of the section) Table 46-62 “USART SPI Timings”: removed footnotes defining 1.8V and 3.3V domains (this information is now found at the beginning of Section 46.11.6 “USART in SPI Mode Timings”) Table 46-63 “Two-wire Serial Bus Requirements”: added parameter “Bus free time between a STOP and START condition”

Table 51-4. SAM4E Datasheet Rev. 11157E Revision History (Continued)

Doc. Date	Changes
13-Feb-15	Section 46. “SAM4E Electrical Characteristics” (cont’d) Section 46.11.8.1 “Timing Conditions”: at end of section, deleted sentence “These values may be product dependent and should be confirmed by the specification” Section 46.11.9 “Embedded Flash Characteristics”: - inserted paragraph explaining that flash contents should be erased prior to programming an application - in second paragraph, corrected “field FWS of the MC_FMR” to “field FWS of the EEFC_FMR” - replaced four “Embedded Flash Wait State” tables with single Table 46-67 “Embedded Flash Wait State at 105°C”
	Section 49. “Ordering Information” Table 49-1 “Ordering Codes for SAM4E Devices”: removed “Package Type” column (package type information is provided on the Atmel website)

Table 51-5. SAM4E Datasheet Rev. 11157D 12-Jun-14 Revision history

Doc. Date	Changes
12-Jun-2014	Modified Title of the document (SAM4E Series)
	Changed structure of the document (order of sections: GMAC, DAC...)
	Ethernet MAC (EMAC) replaced with Ethernet MAC (GMAC) and EMAC signals replaced with GMAC signals throughout the document (example: GTXCK instead of ETXCK, etc.).
	Section 1. “Features” Added tamper detection Modified note 1 (removed “or using internal voltage regulator”) Table 1-1 “Configuration Summary”: Modified information on Timer channels
	Section 2-1 “SAM4E 144-pin Block Diagram” Updated Figure 2-1 “SAM4E 144-pin Block Diagram” and Figure 2-2 “SAM4E 144-pin Block Diagram” (Tamper detection added ; AFE block ; WKUP pins) Timer Counter B and C added in Figure 2-1 “SAM4E 144-pin Block Diagram”
	Section 3. “Signal Description” Modified Table 3-1 “Signal Description List” (“DATRG” instead of “DACTRG”, WKUP[15:0] added, FFPI signals modified)
	Section 4. “Package and Pinout” Modified Table 4-1 “SAM4E 100-ball TFBGA Pinout”, Table 4-2 “SAM4E 144-ball LFBGA Pinout”, Table 4-3 “SAM4E 100-lead LQFP Pinout” and Table 4-4 “SAM4E 144-lead LQFP Pinout”
	Section 5. “Power Considerations” Modified notes after Figure 5-2 “Single Supply” and Figure 5-3 “Core Externally Supplied”: 2.0V replaced with 2.4V (VDDIN minimum value for FAE)
	Section 7.2 “Embedded Memories” Modified Section 7.2.3.1 “Flash Overview” (paragraph below Section 7-4 “Flash Size”) Updated information on the ERASE pin in Section 7.2.3.5 “Security Bit Feature”
	Section 10. “Peripherals” Removed note 1 in Table 10-2 “Multiplexing on PIO Controller A (PIOA)”, Table 10-3 “Multiplexing on PIO Controller B (PIOB)” and Table 10-4 “Multiplexing on PIO Controller C (PIOC)” Removed reset values for Write-only registers
	Section 11. “ARM Cortex-M4 Processor” Minor formatting and editorial changes throughout Updated 2nd instruction line, in Section 11.5.3 “Power Management Programming Hints” Section 11.9.1.2 “CPUID Base Register”: updated ‘Constant’ field description Section 11.9.1.5 “Application Interrupt and Reset Control Register”: updated ‘VECTCLRACTIVE’ and ‘VECTRESET’ field descriptions Section 11.9.1.7 “Configuration and Control Register”: updated ‘USERSETMPEND’ field description Section 11.9.1.16 “MemManage Fault Address Register”: updated ‘ADDRESS’ field description Section 11.9.1.16 “MemManage Fault Address Register”: updated ‘ADDRESS’ field description Section 11.10.1.1 “SysTick Control and Status”: updated ‘TICKINT’ and ‘ENABLE’ field descriptions Section 11.10.1.2 “SysTick Reload Value Registers””: updated ‘RELOAD’ field description Section 11.10.1.3 “SysTick Current Value Register”: updated ‘CURRENT' field description

Table 51-5. SAM4E Datasheet Rev. 11157D 12-Jun-14 Revision history (Continued)

Doc. Date	Changes
12-Jun-2014	Section 11.10.1.4 “SysTick Calibration Value Register”: updated register reset value; updated ‘TENMS’ and ‘SKEW’ field descriptions Section 11.12.2.1 “Coprocessor Access Control Register”: replaced ‘CPn’ field description with ‘CP10 and CP11’ field descriptions Section 11.12.2.3 “Floating-point Context Address Register”: updated ‘ADDRESS’ field description Section 11.12.2.5 “Floating-point Default Status Control Register”: updated descriptions of fields ‘AHP’, ‘DN’, ‘FZ’, and ‘RMode’ Replaced “£” with “≤” in operation description in equations (Section 11.6.7.1 “SSAT and USAT”) Updated Section 11.8.2.1 “NVIC Programming Hints” Updated Section 12.6.3 “ERASE Pin”. Modified Section 12.6.5 “Serial Wire JTAG Debug Port (SWJ-DP) Pins” (added references to PA7)
	Section 13. “Reset Controller (RSTC)” Minor editorial and formatting changes throughout Section 13.4.2.2 “NRST External Reset Control” replaced “ext_nreset” with “exter_nreset” Section 13.4.4.1 “General Reset”: replaced “A general reset occurs when a Power-on-reset is detected” with “A general reset occurs when a VDDIO Power-on-reset is detected” Updated Figure 13-3 “General Reset State” Section 13.4.4.2 “Backup Reset”: replaced “core_backup_reset” with “vddcore_nreset”; reworded content to improve comprehension Section 13.5.1 “Reset Controller Control Register” updated EXTRST value 1 description Updated Section 13.5.2 “Reset Controller Status Register” Updated Section 13.5.3 “Reset Controller Mode Register” Modified Section 13.2 “Embedded Characteristics” and Section 13.4.4.4 “Software Reset”
	Section 14. “Real-time Timer (RTT)” General editorial and formatting changes throughout Figure 14-1, “Real-time Timer” replaced “16-bit Divider” with “16-bit Prescaler Revised Section 14.4 “Functional Description” Section 14.5.1 “Real-time Timer Mode Register”: updated RTPRES field description Section 14.5.4 “Real-time Timer Status Register”: updated RTTINC bit description Updated Section 14.4 “Functional Description”. Modified ALMV description in Section 14.5.2 “Real-time Timer Alarm Register”. Updated Figure 14-2 “RTT Counting”
	Section 15. “Real-time Clock (RTC)” Section 15.1 “Description””: updated to explain need for accurate external 32.768 kHz clock Harmonized write protection register naming throughout Updated Section 15.2 “Embedded Characteristics” Section 15.5.6 “Updating Time/Calendar””: reworded second paragraph for clarity Section 15.5.7 “RTC Accurate Clock Calibration”: replaced sentence “The period interval between 2 correction events is programmable in order to cover the possible crystal oscillator clock variations” with “According to the CORRECTION, NEGPPM and HIGHPPM values configured in the RTC Mode Register (RTC_MR), the period interval between two correction events differs” Updated Section 15.6.2 “RTC Mode Register” Modified Section 15.6.1 “RTC Control Register”, Section 15.6.5 “RTC Time Alarm Register” and Section 15.6.6 “RTC Calendar Alarm Register” : added sentence “This register can only be written if the WPEN bit is cleared in the System Controller Write Protection Mode Register (SYSC_WPMR)”

Table 51-5. SAM4E Datasheet Rev. 11157D 12-Jun-14 Revision history (Continued)

Doc. Date	Changes
12-Jun-2014	Section 1. “Watchdog Timer (WDT)” Figure 1-2, “Watchdog Behavior”, “WDT_CR = WDRSTT” replaced with “WDT_CR.WDRSTT=1”
	Section 17. “Reinforced Safety Watchdog Timer (RSWDT)” General formatting and editorial changes throughout Section 17.2 “Embedded Characteristics”: added bullet “Windowed Watchdog” Figure 17-2 “Watchdog Behavior” replaced “RSWDT_CR = WDRSTT” with “RSWDT_CR.WDRSTT = 1” Added notes in Section 17.5.2 “Reinforced Safety Watchdog Timer Mode Register” and updated Section 17.4 “Functional Description”. KEY is now decribed with a table in Section 17.5.1 “Reinforced Safety Watchdog Timer Control Register”
	Section 18. “Supply Controller (SUPC)” Added Tamper detection and Anti-tampering (Section 18.2 “Embedded Characteristics”, Section 18.4.7.3 “Low-power Tamper Detection and Anti-Tampering”) “Low-power Debouncer Inputs” section restructured: content modified and included in Section 18.4.7.3 “Low-power Tamper Detection and Anti-Tampering” Updated Section 18.3 “Block Diagram” and Figure 18-4 “Wake-up Sources” Updated Section 18.4.2 “Slow Clock Generator”, Section 18.4.4 “Supply Monitor” Updated Section 18.4.4 “Supply Monitor” Section 18.4.6.2 “Brownout Detector Reset” : Reworked 1st paragraph Added Section 18.4.8 “Register Write Protection” and Section 18.4.9 “Register Bits in Backup Domain (VDDIO)” In Section 18.5.9 “System Controller Write Protection Mode Register”: updated register name and bit descriptions. Section 18-5 “Low-power Debouncer (Push-to-Make Switch, Pull-up Resistors)”, Section 18-6 “Low-power Debouncer (Push-to-Break Switch, Pull-down Resistors)” and Section 18-7 “Using WKUP Pins Without RTCOUTx Pins”: Modified pin names. Updated Section 18.5.3 “Supply Controller Control Register”, Section 18.5.4 “Supply Controller Supply Monitor Mode Register”, Section 18.5.6 “Supply Controller Wake-up Mode Register”, Section 18.5.7 “Supply Controller Wake-up Inputs Register”, Section 18.5.8 “Supply Controller Status Register” and Section 18.5.9 “System Controller Write Protection Mode Register” (added information on VDDIO domain and WPEN bit) Section 18.4.7.2 “Wake-up Inputs” corrected WKUPPLx pins to WKUPTx pins. WKUP0, WKUP15 references changed to WKUPx.
	Section 19. “General Purpose Backup Registers (GPBR)” Minor editorial changes Section 19-1 “Register Mapping”: added reset value 0x00000000 for all registers SYS_GPBRx Section 19.3.1 “General Purpose Backup Register x”: inserted sentence “These registers are reset at first power-up and on each loss of VDDBU” below bitmap
	Section 20. “Enhanced Embedded Flash Controller (EEFC)” Reworked section Section 20.4.3.2 “Write Commands” and all sub-sections with figures Figure 20-7 “Full Page Programming” to Figure 20-9 “Programming Bytes in the Flash” Modified Section 20.4.3.3 “Erase Commands” In Section 20.5.2 “EEFC Flash Command Register”, changed the description of FARG field Replaced NVIC by “interrupt controller” everywhere in the document. Revised all figures in the section.
	Section 21. “Fast Flash Programming Interface (FFPI)” Modified Table 21-1 “Signal Description List” (removed references to PGMEN2)

Table 51-5. SAM4E Datasheet Rev. 11157D 12-Jun-14 Revision history (Continued)

Doc. Date	Changes
12-Jun-2014	**Section 22. "Cortex M Cache Controller (CMCC)"** Modified access rights for "Cache Controller Monitor Configuration Register" and "Cache Controller Monitor Enable Register" Modified reset value for "Cache Controller Monitor Status Register" Removed reset values for write-only registers
	Section 24. "SAM-BA Boot Program for SAM4E Microcontrollers" Modified frequency values in Section 24.2 "Embedded Characteristics" and Section 24.3 "Hardware and Software Constraints" ("," replaced with ".")
	Section 25. "DMA Controller (DMAC)" Modified Section 25.2 "Embedded Characteristics" (added Section 25.3 "DMA Controller Peripheral Connections") ARB_CFG described with a table in Section 25.8.1 "DMAC Global Configuration Register" WPKEY described with a table in Section 25.8.19 "DMAC Write Protect Mode Register"
	Section 26. "Peripheral DMA Controller (PDC)" Replaced "on- and/or off-chip" with "target" in Section 26.1 "Description" and Section 26.5.2 "Memory Pointers". Added Section 26.3 "Peripheral DMA Controller Connections" Added last paragraph to Section 26.5.1 "Configuration" specifying that the peripheral clock must be enabled for a PDC transfer
	Section 29. "Power Management Controller (PMC)" Added Section 28.5.5 "Switching Main Clock between the Main RC Oscillator and Fast Crystal Oscillator". Reworked Section 29.13 "Main Clock Failure Detector" for clarity. Updated the list of write protected registers Reworked Section 29.11 "Fast Startup" and added Section 29.12 "Startup from Embedded Flash" Enhanced Section 29.14 "Programming Sequence" Section 29.16 "Register Write Protection": Changed section title and re-worked content. In Section 29.17.20 "PMC Write Protection Mode Register" and Section 29.17.21 "PMC Write Protection Status Register": Changed register names and modified bit and field descriptions. Updated Figure 28-1, "Clock Generator Block Diagram", Figure 28-3, "Main Clock Block Diagram"Figure 28-4, "Divider and PLL Block Diagram" Section 29.17.8 "PMC Clock Generator Main Clock Frequency Register": Added equation to MAINF description.
	Section 30. "Advanced Encryption Standard (AES)" Editorial and minor formatting changes throughout Section 30.4.1 "Operation Modes" updated text at end of section Restructured Section 30.4.3 "Start Modes" to include Section 30.4.3.3 "DMA Mode" Restructured Section 30.4.4 "Last Output Data Mode" Section 30-3 "DMA transfer with LOD = 0": repositioned rising edge of BTC (channel 0) Updated Section 30-4 "Last Output Data Mode Behavior versus Start Modes" In Section 30.6.2 "AES Mode Register", updated LOD field description: Section 30.6.2 "AES Mode Register": updated formula in PROCDLY field description In Section 30.6.10 "AES Initialization Vector Register x", updated IV field description.
	Section 31. "Chip Identifier (CHIPID)" Corrected title for Section 31.3 "Chip Identifier (CHIPID) User Interface"

Table 51-5. SAM4E Datasheet Rev. 11157D 12-Jun-14 Revision history (Continued)

Doc. Date	Changes
12-Jun-2014	**Section 32. "Controller Area Network (CAN)"** Minor editorial formatting changes throughout MCK replaced with Peripheral clock Section 32.6.3 "Interrupt" and Section 32.8.1 "CAN Controller Initialization": Replaced 2 "AIC" occurences with "interrupt controller. Updated Section 32.9.12 "CAN Write Protection Mode Register" Section 32-10 "Possible Initialization Procedure" replaced instance of "AIC" with "Interrupt Controller" Table 32-4 "Receive Mailbox Objects": added missing title Table 32-5 "Transmit Mailbox Objects": added missing title Section 32.7.4.1 "CAN Bit Timing Configuration": moved three bullets describing the phase segments to precede the bullet "TIME QUANTAM"
	Section 33. "Parallel Input/Output Controller (PIO)" Minor editorial and formatting changes throughout Replaced all instances of "PIO clock" and "PIO controller clock" with "peripheral clock" "MCK" replaced with "Peripheral clock" as needed Section 33.5.1 "Pull-up and Pull-down Resistor Control" Changed information to specify that pull-up or pull-down can be set. Updated Section 33.5.3 "Peripheral A or B or C or D Selection" Section 33.5.10 "Input Edge/Level Interrupt": edited, reorganized and reformatted example of interrupt generation Figure 33-3 "I/O Line Control Logic": updated connectivity between clocks and glitch/debouncing filter block; renamed "Resynchronization Stage" to "Peripheral Clock Resynchronization Stage" Moved Section 33.5.15 "I/O Lines Programming Example" to appear before Section 33.5.16 "Register Write Protection" Section 33.5.16 "Register Write Protection": Changed section title and revised content. Updated Section 33.6.46 "PIO Write Protection Mode Register", Section 33.6.47 "PIO Write Protection Status Register" Section 33.6.51 "PIO Parallel Capture Interrupt Enable Register" added bit configuration values Section 33.6.52 "PIO Parallel Capture Interrupt Disable Register": added bit configuration values Section 33.6.53 "PIO Parallel Capture Interrupt Mask Register": added bit configuration values Section 33-6 "Input Debouncing Filter Timing": inserted "(div_slclk)" under "Divided Slow Clock" waveform label
	Section 34. "Serial Peripheral Interface (SPI)" Reworked content. MCK replaced with peripheral clock Updated Section 34.3 "Block Diagram", Section 34-3 "SPI Transfer Format (NCPHA = 1, 8 bits per transfer)" and Section 34-4 "SPI Transfer Format (NCPHA = 0, 8 bits per transfer)" Section 34.2 "Embedded Characteristics": added bullet "Register Write Protection" Modified Section 34.7.3 "Master Mode Operations", Modified Section 34.7.5 "Register Write Protection", Section 34.8.10 "SPI Write Protection Mode Register" and Section 34.8.11 "SPI Write Protection Status Register"

Table 51-5. SAM4E Datasheet Rev. 11157D 12-Jun-14 Revision history (Continued)

Doc. Date	Changes
12-Jun-2014	Section 35. “Two-wire Interface (TWI)” Minor editorial and formatting changes throughout Added “Register Write Protection” in Section 35.2 “Embedded Characteristics” Updated Figure 35-1, “Block Diagram” Updated Section 35.6 “Product Dependencies”, Section 35.7.3.5 “Master Receiver Mode”, Section 35.7.3.7 “Using the Peripheral DMA Controller (PDC)” Restructured Section 35.7 “Functional Description” Table 35-7 “Register Mapping”: replaced TWI_THR reset value “0x00000000” with “–” Section 35.7.3.3 “Programming Master Mode” added one note Clock Synchronization in Write Mode” in Section 35.7.5.5 “Data Transfer”: at end of last sentence, changed “in Read mode” to “in Write mode” Added Section 35.7.5.6 “Using the Peripheral DMA Controller (PDC) in Slave Mode” Updated Section 35.7.6 “Register Write Protection” (changed title and content), Section 35.8.1 “TWI Control Register” Section 35.8.5 “TWI Clock Waveform Generator Register”: replaced t_{mck} with $t_{peripheral\ clock}$ in CLDIV and CHDIV field descriptions Modified Section 35.8.6 “TWI Status Register”: replaced the description of “NACK”, used in master mode, with a new text (address byte is now referenced too) Updated Section 35.8.7 “TWI Interrupt Enable Register” (added first paragraph) Section 35.8.8 “TWI Interrupt Disable Register””: removed reset value from this write-only register Section 35.8.11 “TWI Transmit Holding Register”: removed reset value from this write-only register Section 35.8.12 “TWI Write Protection Mode Register”: replaced list of protectable registers with link to Section 35.7.6 “Register Write Protection” Modified Section 35.8.13 “TWI Write Protection Status Register” Replaced instances of “shift register” with “internal shifter”
	Section 36. “Universal Asynchronous Receiver Transmitter (UART)” Minor editorial/language changes throughout. Changed ‘MCK’ to ‘peripheral clock’ Updated Figure 36-1, “UART Functional Block Diagram” Corrected the offset for PDC registers in Section 36.6 “Universal Asynchronous Receiver Transmitter (UART) User Interface”. Added Section 36.5.5 “Register Write Protection”, and Section 36.6.10 “UART Write Protection Mode Register”. Updated Section 36.6 “Universal Asynchronous Receiver Transmitter (UART) User Interface” table with Write Protection Register.

Table 51-5. SAM4E Datasheet Rev. 11157D 12-Jun-14 Revision history (Continued)

Doc. Date	Changes
12-Jun-2014	Section 37. “Universal Synchronous Asynchronous Receiver Transmitter (USART)” Minor formatting and editorial changes throughout Replaced all references to ‘MCK’ with ’peripheral clock’ Modified Figure 37-1 “USART Block Diagram”: Removed ‘SLCK’. Added ‘Bus clock’. Section 37-2 “I/O Line Description”: removed sentences: ‘Note that it is not recommended to use the USART interrupt line in edge sensitive mode.’ and ‘Configuring the USART does not require the USART clock to be enabled.’ Updated Section 37.2 “Embedded Characteristics”: added bullet: ‘Digital Filter on Receive Line’ Table 37-2 “I/O Line Description”: corrected RXD type from Input to I/O. Section 37.3 “Block Diagram”: removed table “SPI Operating Mode” (information is already present in Table 37-2 “I/O Line Description”) Section 37.7 “Functional Description” Removed list of peripheral characteristics that was redundant with Section 37.2 “Embedded Characteristics”. In Section 37.7.3.4 “Manchester Decoder”, updated information on RXIDLEV bit in 4th paragraph. Section 37.7.5.3 “IrDA Demodulator”: replaced instances of “T” with “t” when used to express time Section 37.7.8.5 “Character Transmission”: INACK replaced by WRDBT. Section 37.7.10 “Register Write Protection”: Changed section title and reworked content. Updated Section 37.8.3 “USART Mode Register” Section 37-13 “IrDA Baud Rate Error”: added missing units of measure to column headers In Table 9-1, “Section 37-16 “Register Mapping”” changed register name to Manchester Configuration Register to be consistent throughout the document. Section 37.8.18 “USART FI DI RATIO Register” modified FI_DI_RATIO field from 16 bits to 11 bits. In Section 37.8.21 “USART Manchester Configuration Register”: added RXIDLEV as bit 31 and added bit description. Section 37.8.22 “USART Write Protection Mode Register” and Section 37.8.23 “USART Write Protection Status Register”: Changed register names and modified bit and field descriptions. Figure 37-37 “Example of RTS Drive with Timeguard”: Figure modified with RTS rising edge prior to start bit

Table 51-5. SAM4E Datasheet Rev. 11157D 12-Jun-14 Revision history (Continued)

Doc. Date	Changes
12-Jun-2014	Section 1. “Timer Counter (TC)” Editorial and formatting changes throughout Master clock” or “MCK” replaced with “peripheral clock”. Removed references to FILTER bit (register bit 19 now reserved in Section 39.7.16 “TC Block Mode Register”) Figure 1-16 “Synchronization with PWM” added value ‘1’ to all multiplexers Figure 1-18 “Input Stage”: replaced “FILTER” with “MAXFILTER > 0” Updated Figure 1-1 “Timer Counter Block Diagram” Updated Figure 1-5 “Example of Transfer with PDC” Erroneous description of TCCLKS table, rows 0 to 4 reworked in Section 1.7.2 “TC Channel Mode Register: Capture Mode” and Section 1.7.3 “TC Channel Mode Register: Waveform Mode”" Updated Section 1.7.16 “TC Block Mode Register” Section 1.6.16.3 “Direction Status and Change Detection”: rewrote sixth paragraph for clarity Section 1.6.16.4 “Position and Rotation Measurement” rewrote first paragraph for clarity Section 1.6.16.3 “Direction Status and Change Detection” replaced sentence “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” Added Section 1.6.16.6 “Missing Pulse Detection and Auto-correction” Added configuration bit AUTOC in Section 1.7.16 “TC Block Mode Register” Section 1.6.18 “Register Write Protection” changed title (was “Write Protection System”); revised content Section 1.7.22 “TC Write Protection Mode Register”: modified register name (was “TC Write Protect Mode Register”); updated WPEN field description (replaced list of protectable registers with link to Section 1.6.18 “Register Write Protection”) Replaced “0xFFFF” with “2^n-1” (with “n” representing counter size) in Section 1.6.12.1 “WAVSEL = 00”, Section 1.6.12.3 “WAVSEL = 01”, Figure 1-10 “WAVSEL = 10 without Trigger”, Section 1-14 “WAVSEL = 11 without Trigger”, Figure 1-11 “WAVSEL = 10 with Trigger” and Figure 1-15 “WAVSEL = 11 with Trigger”:
	Section 1. “Pulse Width Modulation Controller (PWM)” Editorial and formatting changes throughout. Updated Table 1-4 “Fault Inputs” Modified Section 1.6.2.2 “Comparator” Section 1.6.6 “Register Write Protection”: at end of section, replaced sentence “The WPVS and PWM_WPSR fields are automatically reset after reading the PWM_WPSR register” with “The WPVS and WPVSRC fields are automatically cleared after reading the PWM_WPSR” Section 1.7.9 “PWM Sync Channels Mode Register”: removed table row for value 3 “reserved” in UPDM field description WPKEY/WPCMD are now described with tables in Section 1.7.34 “PWM Write Protection Control Register” Corrected reset value of PWM_FPV2 in Table 1-7 “Register Mapping” (was 0x0000_0000; is 0x003F_003F) Deleted instances of “(fault input bit varies from 0 to Z-1)” from field descriptions in Section 1.7.24 “PWM Fault Mode Register”, Section 1.7.25 “PWM Fault Status Register” on page 1139, Section 1.7.26 “PWM Fault Clear Register” and Section 1.7.28 on page 1142 Updated Section 1.7.34 “PWM Write Protection Control Register” on page 1148

Table 51-5. SAM4E Datasheet Rev. 11157D 12-Jun-14 Revision history (Continued)

Doc. Date	Changes
12-Jun-2014	Section 40. “High Speed MultiMedia Card Interface (HSMCI)” Minor formatting and editorial changes throughout Figure 40-1, “Block Diagram (4-bit configuration)”: added “(4-bit configuration)” to title; added missing note below figure Modified Section 40.8.1 “Command - Response Operation” Section 40.13 “Register Write Protection” changed title (was “Write Protection Registers”); revised content Section 40.14.17 “HSMCI Write Protection Mode Register”: modified register name (was HSMCI Write Protect Mode Register); replaced list of protectable registers with cross-reference to section “Register Write Protection” Section 40.14.18 “HSMCI Write Protection Status Register” modified register name (was HSMCI Write Protect Status Register) and updated description
	Section 41. “USB Device Port (UDP)” Minor editorial and formatting changes throughout Figure 41-1 “Block Diagram”: added “interrupt line” below “udp_int” Section 41.2 “Embedded Characteristics” on page 1158: replaced bullet “Integrated Pull-up on DP” with “Integrated Pull-up on DPP” added bullet “Integrated Pull-down on DDM”* Section 41.5.1 “USB Device Transceiver” on page 1161: reworded content for clarity Section 41-5 “USB Transfer Events” on page 1163: restructured table and reorganized contents Section 41.6.3 “Controlling Device States” on page 1173: replaced “may not consume more than 500 µA” with “must not consume more than 2.5 mA” Section 41.6.3.6 “Entering in Suspend State” on page 1174: replaced “must drain less than 500uA” with “must drain no more than 2.5 mA” Section 41.7.10 “UDP Endpoint Control and Status Register (CONTROL_BULK)”: changed EPTYPE[2:0] field configuration values from binary to decimal Section 41.7.11 “UDP Endpoint Control and Status Register (ISOCHRONOUS)””: changed EPTYPE[2:0] field configuration values from binary to decimal
	Section 42. “Ethernet MAC (GMAC)” Minor editorial and formatting changes throughout Updated Section 42.6.1.6 “Interrupts”: in first paragraph, deleted content “Depending on the overall system ... CPU enters the interrupt handler” and “Note that in the default ... be write-one-toclear if desired” In Section 42.6.2 “Statistics Registers”, deleted sentence “In order to reduce overall design area, the Statistics Registers may be optionally removed in the configuration file if they are deemed unnecessary for a particular design.” Section 42.7.1 “Network Control Register” removed bit RDS (“Read Snapshot” function not supported) Section 42.7.32 “Stacked VLAN Register”: added missing description to field ESVLAN Updated Section 42.7.27 “Type ID Match 1 Register”, Section 42.7.28 “Type ID Match 2 Register”, Section 42.7.29 “Type ID Match 3 Register” and Section 42.7.30 “Type ID Match 4 Register” (added EINDx bits and updated TID bit description Section 42.7.81 “1588 Timer Sync Strobe Seconds [31:0] Register” and Section 42.7.83 “1588 Timer Seconds [31:0] Register”: updated title and register name (GMAC_TSSSL and GMAC_TSL instead of GMAC_TSSS and GMAC_TS Updated Section 42.5.2 “1588 Time Stamp Unit”

Table 51-5. SAM4E Datasheet Rev. 11157D 12-Jun-14 Revision history (Continued)

Doc. Date	Changes
12-Jun-2014	Section 43. “Analog Comparator Controller (ACC)” Updated Figure 43-1 “Analog Comparator Controller Block Diagram” Added Table 43-1 “List of analog inputs” Updated Table 43-2 “ACC Pin List” ADx analog inputs replaced with AFEx_ADx external analog data inputs Section 43.1 “Description”, Section 43.6.2 “Analog Settings” and Section 43.6.4 “Fault Mode”: Updated section for clarity. Replaced Section “Write Protection System” with Section 43.6.5 “Register Write Protection”. Updated Section 43.7.8 “ACC Write Protection Mode Register” and Section 43.7.9 “ACC Write Protection Status Register”.
	Section 44. “Analog-Front-End Controller (AFEC)” General editorial and formatting changes throughout Updated Section 44.3 “Block Diagram” Updated Section 44.6.3 “Conversion Resolution” Added Section 44.6.5 “Conversion Results Format” Added the last paragraph (“Depending on the sign of the conversion...”) in ”Section 44.6.8 “Comparison Window” Updated Section 44.6.9 “Differential Inputs”, Section 44.6.13 “Enhanced Resolution Mode and Digital Averaging Function”, added two paragraphs (“Note that,...” and “As the consequence,...”) Updated Section 44.6.17 “Register Write Protection” Section 44-7 “Analog Full Scale Ranges in Single Ended/Differential Applications Versus Gain”: replaced instances of “vrefin” with “V_{ADVREF}” Updated Table 44-7 “Register Mapping”, added new registers and updated offsets for reserved registers ”Section 44.7.3 “AFEC Extended Mode Register”: added SIGNMODE (bits 29:28) and AFEMODE (bits 21:20) Section 44.7.14 “AFEC Overrun Status Register” added a note Section 44.7.15 “AFEC Compare Window Register”: -LOWTHRES: updated the field description -HIGHTRES: updated the field description Section 44.7.17 “AFEC Channel Calibration DC Offset Register” In OFFx bit description, replaced instances of “Vrefin/2” with “V_{ADVREF}/2” Modified DATA field description in Section 44.7.20 “AFEC Channel Data Register” Section 44.7.23 “AFEC Temperature Compare Window Register”: -TLOWTHRES: updated the field description -THIGHTRES: updated the field description Section 44.7.25 “AFEC Write Protection Mode Register” - modified the section title/register name (was “AFEC Write Protect Mode Register”)/content Section 44.7.26 “AFEC Write Protection Status Register”: updated WPVSRC field description

Table 51-5. SAM4E Datasheet Rev. 11157D 12-Jun-14 Revision history (Continued)

Doc. Date	Changes
12-Jun-2014	Section 44. “Digital-to-Analog Converter Controller (DACC)” Editorial and formatting changes throughout. MCK or Master clock replaced with Peripheral clock. Removed references to Sleep mode and refresh period Renamed “Features” chapter as “Embedded Characteristics” Updated Section 44.2 “Embedded Characteristics”” In Section 44.7.2 “DACC Mode Register”: - REFRESH bit replaced with ONE bit - Removed FASTWAKEUP bit and SLEEP bit Re-worked Section 44.6.7 “Register Write Protection” and associated registers and bit/field descriptions in Section 44.7.7 “DACC Interrupt Enable Register”, Section 44.7.8 “DACC Interrupt Disable Register” and Section 44.7.9 “DACC Interrupt Mask Register”: modified bit descriptions. Section 44.7.12 “DACC Write Protection Mode Register”” and Section 44.7.13 “DACC Write Protection Status Register”
	Section 46. “SAM4E Electrical Characteristics” Updated whole section Added reference to note 1 in Table 46-12 “Typical Current Consumption in Wait Mode (1)” title I_O conditions modified in Table 46-2 “DC Characteristics” VDDIN replaced with V_{VDDIN} for VDDIN voltage values VDDIO replaced with V_{VDDIO} for VDDIO voltage values Modified Section 46.3.1 “Backup Mode Current Consumption” Modified Figure 46-7, “Measurement Setup for Wait Mode” Updated Section 46.7 “12-bit AFE (Analog Front End) Characteristics” and Figure 46-15 “12-bit AFE (Analog Front End) Diagram” Updated Section 46.8 “12-bit DAC Characteristics” Added Erase Pin Assertion Time in Table 46-68 “AC Flash Characteristics”
	“Marking”section moved to **Section 48.**
	Section 50. “Errata on SAM4E Devices” Added Section 50.1.3 “Flash”

Table 51-6. SAM4E Datasheet Rev. 11157C 25-Jul-2013 Revision History

Doc. Date	Changes
25-Jul-2013	Introduction In "Features" : - added information on Two-wire Interface in Peripherals section and Wake-on-LAN for EMAC - changed operating temperature range to 105°C Updated Table 1-1 "Configuration Summary" with TWI information In Section 4. "Package and Pinout", added the FFPI signals to Table 4-1 "SAM4E 100-ball TFBGA Pinout", Table 4-2 "SAM4E 144-ball LFBGA Pinout", Table 4-3 "SAM4E 100-lead LQFP Pinout" and Table 4-4 "SAM4E 144-lead LQFP Pinout". Updated Section 5.5 "Low-power Modes". Added information on WFE. In Section 6.1 "General Purpose I/O Lines" on page 21, added information on GPIOs as analog input. Removed Section 7. "Processor and Architecture". Removed Sections 11-4 to 11-16. Reordered introduction sections. Removed Note regarding PIOs and 144-pin package at bottom of Table 11-5, "Multiplexing on PIO Controller D (PIOD)," on page 40 and Table 11-6, "Multiplexing on PIO Controller E (PIOE)," on page 41.
	RSTC: In Section 15.3.6 "Reset Controller Status Register", RSTTYP information corrected.
	RTT: Added notes in Section 16.4 "Functional Description", Section 16.5.1 "Real-time Timer Mode Register" and in Section 16.5.2 "Real-time Timer Alarm Register".
	RTC: In Section 18.5.3 "Alarm", added new information and note. In Section 18.5.7 "RTC Accurate Clock Calibration", updated information on temperature range. In Section 18.6.5 "RTC Time Alarm Register" and Section 18.6.6 "RTC Calendar Alarm Register", added notes.
	WDT: In Section 19.1 "Description", added information on slow clock at 32 kHz. In Section 19.2 "Embedded Characteristics", added that Watchdog Clock is independent from Processor Clock. Moved note (WDD, WDV) from Section 19.5.3 "Watchdog Timer Status Register" to Section 19.5.2 "Watchdog Timer Mode Register".
	EFC: In Section 22.4.3.5 "Lock Bit Protection", added notes on FARG exceeding limits. Updated existing note in Section 22.4.3.6 "GPNVM Bit". Added Section 22.4.3.3 "Optimized Partial Programming". Added note on programming limitations in Section 22.4.3.2 "Write Commands".
	FFPI: Removed information throughout on Serial Fast Flash Programming not available for device.
	CMCC: In Table 24.5 "Cortex M Cache Controller (CMCC) User Interface", updated reset value of CMCC_SR.

Table 51-6. SAM4E Datasheet Rev. 11157C 25-Jul-2013 Revision History (Continued)

Doc. Date	Changes
25-Jul-2013	PMC: Section 30.1.4.2 “Slow Clock Crystal Oscillator”, replaced “...in MOSCSEL bit of CKGR_MOR,...” with “...in XTALSEL bit of SUPC_CR,...” in the last phrase of the 3d paragraph. Section 30.1.4.2 “Slow Clock Crystal Oscillator”, added references on the OSCSEL bit of PMC_SR in the 3d paragraph. Register names in Clock Generator: Replaced “PLL_MCKR” with “PMC_MCKR” and “PLL_SR” with “PMC_SR” in Section 30.1.5.5 “Software Sequence to Detect the Presence of Fast Crystal”. In Section 30.1.6.1 “Divider and Phase Lock Loop Programming”, 3rd bullet, replaced PMC_IER with PMC_SR. Deleted previous 4th bullet (was useless sentence “Disable and then enable the PLL...”). In Figure 30-3 “Main Clock Block Diagram” and Section 30.1.5.3 “3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator” paragraph 5, replaced MOSCXTCNT with MOSCXTST. Added code example in step 1. of Section 30.2.13 “Programming Sequence”. Corrected reset value of CKGR_MOR register in Table 30-2, “Register Mapping”.
	USART: In Table 37-10 “Maximum Timeguard Length Depending on Baud Rate” and in Table 37-11 “Maximum Time-out Period”, modified 33400 baudrate to 38400. Updated Figure 37-22 “Parity Error” with corrected stop bit value.
	TC: In Section 38.1 “Description”, corrected reference to TIOA1 with TIOB1. In Section 38.7.3 “TC Channel Mode Register: Waveform Mode”, added note for ENETRG description.
	HSMCI: In Figure 39-8 “Read Functional Flow Diagram”, Figure 39-9 “Write Functional Flow Diagram” and Figure 39-10 “Multiple Write Functional Flow Diagram”, corrected HSMCI_MR to HSMCI_BLKR when referring to Block Length field that is not available in HSMCI_MR. Removed related Note 2. In Section 39.14.7 “HSMCI Block Register”, BLKLEN bit description, removed reference on accessiblity in Mode Register.

Table 51-6. SAM4E Datasheet Rev. 11157C 25-Jul-2013 Revision History (Continued)

Doc. Date	Changes
25-Jul-2013	Electrical Characteristics Operating temperature is extended to 105°C. Changed/updated in: - Table 46-1 “Absolute Maximum Ratings*” - Section 46.2 “DC Characteristics” - Table 46-5 “VDDIO Supply Monitor” - Table 46-16 “32 kHz RC Oscillator Characteristics” - Table 46-17 “4/8/12 MHz RC Oscillators Characteristics” - Table 46-45 “Temperature Sensor Characteristics” - Table 46-58 “Embedded Flash Wait State VDDCORE set at 1.08V and VDDIO 1.62V to 3.6V @105C” - Table 46-59 “Embedded Flash Wait State VDDCORE set at 1.08V and VDDIO 2.7V to 3.6V @105C” - Table 46-60 “Embedded Flash Wait State VDDCORE set at 1.2V and VDDIO 1.62V to 3.6V @ 105C” - Table 46-61 “Embedded Flash Wait State VDDCORE set at 1.20V and VDDIO 2.7V to 3.6V @ 105C” - Table 46-62 “AC Flash Characteristics”. Added Program cycle time/Write page mode values. In Section 46.3 “Power Consumption”, added bullet with conditions of power consumption values. In Section 46.3.1.1 “Configuration A: Embedded Slow Clock RC Oscillator Enabled” and Section 46.3.1.2 “Configuration B: 32768 kHz Crystal Oscillator Enabled”, added bullet on BOD disabled. New values in Table 46-9 “Power Consumption for Backup Mode Configuration A and B” In Section 46.3.2.1 “Sleep Mode”, added bullet on VDDIO. In Section 46.3.2.2 “Wait Mode”, added bullet on VDDIO. New values in Table 46-12 “Typical Current Consumption in Wait Mode”.
	Ordering Information Table 48-1 “Ordering Codes for SAM4E Devices” updated with new ordering codes for parts at 105°C and for tape & reel
	Errata Added Section 49. “Errata on SAM4E Devices” that includes Section 49.2.1.1 “Watchdog Not Stopped in Wait Mode” and Section 49.2.2.1 “Unpredictable Behavior if BOD is Disabled, VDDCORE is Lost and VDDIO is Connected”

Table 51-7. SAM4E Datasheet Rev. 11157B 25-April-2013 Revision History

Doc. Date	Changes
25-Apr-2013	Introduction: Updated the section structure and added references to 100-ball TFBGA and 100-lead LQFP packages in: - Section 1. “Features” - Table 1-1 “Configuration Summary” - Figure 2-1 “SAM4E 100-pin Block Diagram” - Section 4.1 “100-ball TFBGA Package and Pinout” - Section 4.3 “100-lead LQFP Package and Pinout” - Table 11-1 “PIO available according to Pin Count” Added Analog Comparator (ACC) and Reinforced Safety Watchdog Timer (RSWDT) blocks in Figure 2-2 “SAM4E 144-pin Block Diagram”. Updated the description of power supply pins in Section 5.1 “Power Supplies”. Updated Figure 11-3 “Power Management Controller Block Diagram”. Removed RC80M references in Figure 10-1 “System Controller Block Diagram” and Figure 11-2 “Clock Generator Block Diagram”. Add data on consumption and wake-up time in Table 5-1 “Low-power Mode Configuration Summary”. Removed “AT91SAM” from the document title and further on in the entire document (where appropriate). Replaced “Cortex™” references with “Cortex®“ in “Description” and further on in the entire document. Section 11.14 “Chip Identification”, replaced “Table 11-1. SAM4E Chip ID Register” with a cross-reference to the corresponding Table 14-1 “SAM4E Chip ID Registers” (Section 14. “Chip Identifier (CHIPID)”). Removed package dimension references in Section 4. “Package and Pinout”. Added a phrase on the flash write commands usage in Section 8.1.3.1 “Flash Overview” (the last paragraph). Updated package information in Section 11.2 “Peripheral Signal Multiplexing on I/O Lines”: - replaced “100/144 pin version” with “144 pin version” in Table 11-4 “Multiplexing on PIO Controller C (PIOC)” - removed “144 pin version” in Table 11-5 “Multiplexing on PIO Controller D (PIOD)” Updated Figure 7-1 “SAM4E Product Mapping”. Replaced GRX by GRX1 on line PD6 in Table 11-5 “Multiplexing on PIO Controller D (PIOD)”.
	CHIPID: Section 14.3 “Chip Identifier (CHIPID) User Interface”, updated the ARCH bitfield table in “ARCH: Architecture Identifier” (removed rows with not relevant information: 0x43, 0x88, 0x89, 0x8A, 0x93, 0x94, and 0x95). Section 14.2 “Embedded Characteristics”, replaced ‘0x0011_0201’ with ‘0x0012_0201’ and ‘0x0011_0209’ with ‘0x0012_0209’ in Table 14-1 “SAM4E Chip ID Registers”. Section 14.3.2 “Chip ID Extension Register”, updated value in the Flash Size table and removed package references in the Product Number table.
	RTT: Section 16.3 “Block Diagram”, replaced ‘CLKSRC’ source reference with ‘RTC1HZ’ in Figure 16-1 “Real-time Timer”. Updated the 4th and the 8th paragraphs in Section 16.4 “Functional Description” (“Setting the RTC 1 HZ clock to 1...” and “The RTTINC bit in RTT_SR is set...” respectively). Section 16.5.1 “Real-time Timer Mode Register”, added notes in “RTTDIS: Real-time Timer Disable” and “RTC1HZ: Real-Time Clock 1Hz Clock Selection”.
	RSWDT: Added a new component: Section 17. “Reinforced Safety Watchdog Timer (RSWDT)”.

Table 51-7. SAM4E Datasheet Rev. 11157B 25-April-2013 Revision History (Continued)

Doc. Date	Changes
25-Apr-2013	RTC: Section 18.2 “Embedded Characteristics”, added a new bullet "Safety/Security features”, right indented the 2 following bullets. Section 18.5 “Functional Description”, added the last paragraph (“The RTC can generate...”).
	SUPC: Updated Figure 20-1 “Supply Controller Block Diagram”. Updated the 2-nd paragraph in Section 20.4.7.4 “Low-power Tamper Detection Inputs” and placed this section just after Section 20.4.7.3 “Low-power Debouncer Inputs”.
	EFC: Typo fixed in Section 22.4.3.5 “GPNVM Bit” and added title in Section 22.4.3.6 “Calibration Bit”. Added notes when FARG exceeds limits in Section 22.4.3.4 “Lock Bit Protection” and reworked the existing note in Section 22.4.3.5 “GPNVM Bit”.
	FFPI: Removed duplicate and erroneous figures in: - Section 23.3.1 “Device Configuration” - Section 23.3.4.1 “Write Handshaking” - Section 23.3.4.2 “Read Handshaking” - Section 23.3.5.8 “Get Version Command” Fixed the section structure.
	MATRIX: Removed references to Special Function Registers (SFR) and to Bus Matrix Priority Registers B for Slaves. Updated sections: - Section 26.1 “Description” - Section 26.2 “Embedded Characteristics” - Section 26.12 “Bus Matrix (MATRIX) User Interface”: - Table 26-3 “Register Mapping” - Section 26.12.1 - Section 26.12.4 - Section 26.12.7 Updated register names to “MATRIX_MCFGx [x=0..6]” and so on in Section 26.12.1 “Bus Matrix Master Configuration Registers”, Section 26.12.2 “Bus Matrix Slave Configuration Registers”, and Section 26.12.3 “Bus Matrix Priority Registers A For Slaves”. Replaced the WPKEY bitfield description with the corresponding table in Section 26.12.7 “Write Protect Mode Register”.
	PMC: Section 30.2.16.9 “PMC Clock Generator PLLA Register”, removed “x8” in “PLLACOUNT: PLLA Counter” bitfield description. Section “”, replaced the KEY bitfield description with a table. Section 30.2.16.20 “PMC Write Protect Mode Register”, replaced the WPKEY bitfield description with a table. Updated the last paragraph in Section 30.1.5.2 “Fast RC Oscillator Clock Frequency Adjustment” and added the corresponding note in Section 30.2.16.8 “PMC Clock Generator Main Clock Frequency Register”.
	AES: Updated Section 31.4.4 “DMA Mode” and Section 31.4.7 “DMA Mode” (“PDC Mode” --> “DMA Mode”). Section 31.6.2 “AES Mode Register”, replaced the CKEY bitfield description with a table.

Table 51-7. SAM4E Datasheet Rev. 11157B 25-April-2013 Revision History (Continued)

Doc. Date	Changes
25-Apr-2013	PIO: Section 33.5 “Functional Description”, added pull-down resistor and registers in Figure 33-5 “Input Glitch Filter Timing”. Section 33.7.46 “PIO Write Protect Mode Register”, replaced the WPKEY bitfield description with a table. Added missing dashes for reserved registers in Table 33-3 “Register Mapping”. Replaced “DIVx” with “DIV” in Section 33.7.29 “PIO Slow Clock Divider Debouncing Register”. Updated the SCHMITTx bitfield description in Section 33.7.48 “PIO Schmitt Trigger Register” and updated the Delayx bitfield description in Section 33.7.49 “PIO I/O Delay Register”.
	SPI: Section 34.7.3.2 “Master Mode Flow Diagram”, added TDRE references in Figure 34-8 “PDC Status Register Flags Behavior”. Section 34.7.4 “SPI Slave Mode”, updated the next-to-last paragraph (“Then, a new data is loaded...”). Replaced offset 0x4C with 0x40, 0xE8 with 0xEC in Section 34.8 “Serial Peripheral Interface (SPI) User Interface”.
	USART: Added a paragraph on IRDA_FILTER programming criteria in Section 37.7.5.3 “IrDA Demodulator” and in the corresponding bitfield description in Section 37.8.20 “USART IrDA FILTER Register”. Section 37.8.18 “USART FI DI RATIO Register”, expanded FI_DI_RATIO bitfield to 16 bits in the register table. Added RXBUFF and TXBUFE bitfields and their descriptions in: - Section 37.8.6 “USART Interrupt Enable Register (SPI_MODE)” - Section 37.8.8 “USART Interrupt Disable Register (SPI_MODE)” - Section 37.8.10 “USART Interrupt Mask Register (SPI_MODE)” - Section 37.8.12 “USART Channel Status Register (SPI_MODE)”
	TC: Fixed a typo in Section 38.1 “Description”: “TIOA1” --> “TIOB1”.
	ACC: Added Table 42-2 “Analog Comparator Inputs”.

Table 51-7. SAM4E Datasheet Rev. 11157B 25-April-2013 Revision History (Continued)

Doc. Date	Changes
25-Apr-2013	AFEC: Fixed a typo in Section 43.7.25 “AFEC Write Protect Mode Register”: replaced ‘(“AFE” in ASCII)’ with ‘(“ADC” in ASCII)’ in the WPEN and WPKEY bitfield descriptions. Updated register tables (replaced bitfield data with “-” for bits from 16 to 23) in: - Section 43.7.6 “AFEC Channel Enable Register” - Section 43.7.7 “AFEC Channel Disable Register” - Section 43.7.8 “AFEC Channel Status Register” - Section 43.7.17 “AFEC Channel Calibration DC Offset Register” Updated the acronym from ‘AFE’ to ‘AFEC’ in the entire document (except of ‘AFE Controller’). Updated Section 43.2 “Embedded Characteristics”. Reworked Section Section 43.6.9 “Input Gain and Offset”: - updated the first and the last paragraphs - removed Table 42-7 Offset of the Sample and Hold Unit: OFFSET DIFF and Gain (G) - updated Figure 43-7 “Analog Full Scale Ranges in Single Ended/Differential Applications Versus Gain”. Rewritten Section 43.6.13 “Automatic Calibration”. Updated Section 43.7 “Analog-Front-End Controller (AFEC) User Interface”: - AFEC_CSELR, AFEC_COCR are declared as Read-write registers - “Channel DC Offset Register’ --> ‘Channel Calibration DC Offset Register” - updated “ANACH: Analog Change” bitfield description table in Section 43.7.2 “AFEC Mode Register” - updated the OFFx bitfield description and added a note in Section 43.7.17 “AFEC Channel Calibration DC Offset Register”. Updated the last paragraph in Section 43.6.3.1 “Enhanced Resolution Mode”.
	GMAC: Replaced "at all three speeds" by "at all supported speeds" in Section 44.1 “Description” and Section 44.2 “Embedded Characteristics”. Section 44.7.1 “Network Control Register”, removed the LB bitfield and its description and updated the LBL bitfield description (removed phrases: "Bit 11 of GMAC_R ... loopback mode." and "Local loopback functionality is optional."). Section 44.7.4 “User Register”, removed the BPDG, HDFC and RMII bitfields and their descriptions. Removed references to external FIFO/external FIFO interface in the entire document.

Table 51-7. SAM4E Datasheet Rev. 11157B 25-April-2013 Revision History (Continued)

Doc. Date	Changes
25-Apr-2013	Electrical Characteristics: Added references to 100-ball TFBGA and 100-lead LQFP packages in Table 46-1 “Absolute Maximum Ratings*”. Updated data in: - Table 46-2 “DC Characteristics” - Table 46-3 “1.2V Voltage Regulator Characteristics” - Section 46.3.1.2 “Configuration B: 32768 kHz Crystal Oscillator Enabled” - Section 46.3.2.1 “Sleep Mode” - Section 46.3.2.2 “Wait Mode”, including Table 46-12 “Typical Current Consumption in Wait Mode” - Table 46-13 “Active Power Consumption with VDDCORE @ 1.2V running from Embedded Memory (IDDCORE- AMP1)” - Table 46-15 “Power Consumption on VDDCORE(1)” - Table 46-16 “32 kHz RC Oscillator Characteristics” - Table 46-27 “Analog Power Supply Characteristics” - Table 46-28 “Channel Conversion Time and ADC Clock” - Table 46-29 “External Voltage Reference Input” - Section 46.7.1 “ADC Resolution” - Section 46.7.2 “Static Performance Characteristics” - Section 46.7.3 “Dynamic Performance Characteristics” - Notes in Table 46-47 “I/O Characteristics” Added: - Figure 46-6 “Current Consumption in Sleep Mode (AMP1) versus Master Clock Ranges (Condition from Table 46-10)” - Figure 46-9 “Active Power Consumption with VDDCORE @ 1.2V” - Section 46.3.3.2 “SAM4E Active Total Power Consumption” - Figure 46-14 “12-bit AFE (Analog Front End) Diagram” Updated temperature range from “-40°C - +125°C” to “-40°C - +85°C” in Table 46-16 “32 kHz RC Oscillator Characteristics”. Added missing titles in: - Figure 46-11 “32.768 kHz Crystal Oscillator Schematics” - Figure 46-12 “3 to 20 MHz Crystal Oscillator Schematics”

Table 51-7. SAM4E Datasheet Rev. 11157B 25-April-2013 Revision History (Continued)

Doc. Date	Changes
25-Apr-2013	Replaced “RES = 1” with “RES = 0” in Table 46-30 “ADC Resolution following Digital Averaging”. Updated Table 46-33 “Gain and Error Offset, 12-bit Mode, VDDIN 2.4V to 3.6V Supply Voltage Conditions”. Section 46.7.1.2 “Conditions @ 25 degrees with Gain =4”, replaced “f_S = 1 kHz” with “f_S = 1 MHz”. Section 46.11.3.2 “SPI Timings”, updated for better presentation the paragraph “Note that in SPI master mode,...”. Updated notes in Table 46-52 “SMC Write NCS Controlled (WRITE_MODE = 0)” and in Table 46-57 “EMAC MII Timings”. Section 46.8 “12-bit DAC Characteristics”, updated data in Table 46-41 “Static Performance Characteristics” and Table 46-42 “Dynamic Performance Characteristics”. Updated data in Table 46-62 “AC Flash Characteristics”.
	Mechanical Characteristics: Updated the section structure and added references on 100-ball TFBGA and 100-lead LQFP packages in: - Section 47.1 “100-ball TFBGA Package Drawing” - Section 47.3 “100-lead LQFP Package Drawing”
	Ordering Codes: Added references to 100-ball TFBGA and 100-lead LQFP packages in Table 48-1 “Ordering Codes for SAM4E Devices”.

Table 51-8. SAM4E Datasheet Rev. 11157A 14-Jan-2013 Revision History

Doc. Date	Changes
14-Jan-2013	Initial release

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**

X-ON Electronics

Largest Supplier of Electrical and Electronic Components

Click to view similar products for ARM Microcontrollers - MCU ***category:***

Click to view products by Microchip ***manufacturer:***

Other Similar products are found below :

R7FS3A77C2A01CLK#AC1 MB96F119RBPMC-GSE1 MB9BF122LPMC1-G-JNE2 MB9BF122LPMC-G-JNE2 MB9BF128SAPMC-GE2 MB9BF218TBGL-GE1 MB9BF529TBGL-GE1 26-21/R6C-AT1V2B/CT 5962-8506403MQA MB9AF342MAPMC-G-JNE2 MB96F001YBPMC1-GSE1 MB9BF121KPMC-G-JNE2 VA10800-D000003PCA CP8547AT CY9AF156NPMC-G-JNE2 MB9BF104NAPMC-G-JNE1 ADUCM410BCBZ-RL7 GD32f303RGT6 NHS3152UK/A1Z MK26FN2M0CAC18R EFM32TG230F32-D-QFN64 EFM32TG232F32-D-QFP64 EFM32TG825F32-D-BGA48 MB9AFB44NBBGL-GE1 MB9BF304RBPMC-G-JNE2 MB9BF416RPMC-G-JNE2 MB9AF155MABGL-GE1 MB9BF306RBPMC-G-JNE2 MB9BF618TBGL-GE1 ATSAMS70N21A-CN MK20DX64VFT5 MK50DX128CMC7 MK51DN256CMD10 MK51DX128CMC7 MK53DX256CMD10 MKL25Z32VFT4 LPC1754FBD80 STM32F030K6T6TR ATSAM3N0AA-MU ATSAM3N0CA-CU ATSAM3SD8BA-MU ATSAM4LC2BA-UUR ATSAM4LC4BA-MU ATSAM4LS2AA-MU ADuC7023BCPZ62I-R7 ATSAM4LS4CA-CFU STR711FR0T6 XMC1302Q040X0200ABXUMA1 STM32L431RCT6 ADUCM3027BCPZ-R7