SAM4CP16C

Atmel | SMART Power Line Communications Device

DATASHEET

Description

Atmel SAM4CP16C belongs to Atmel® | SMART energy portfolio. It is based on SAM4C, a high performance 32-bit, dual core ARM® Cortex®-M4 RISC processor embedding a G3 PLC [Power Line Communication] modem. The two cores Cortex-M4 are able to operate at a maximum speed of 120 MHz, featuring 1 Mbyte of embedded Flash, 128 kBytes of SRAM and on-chip cache for each core.

SAM4CP16C is a flexible, compact and high-efficient device for a wide range of Smart Grid applications such as Smart Metering (Smart Meters and Data Concentrators), Lighting, Industrial / Home Automation, Home and Building Energy Management Systems, Solar Energy and Plug-in Hybrid Electric Vehicle (PHEV) Charging Stations.

The unique dual ARM Cortex-M4 architecture allows implementation of signal processing, application and communications firmware in independent partitions, supported by a powerful embedded PLC modem and an extensive set of embedded cryptographic features. SAM4CP16C can be combined with external Atmel devices for metrology, representing a flexible and highly efficient platform for smart metering applications.

The peripheral set includes advanced cryptographic engine, anti-tamper, floating point unit (FPU), 5x USARTs, 2x UARTs, 2x TWIs, 6 x SPI, as well as 1 PWM timer, 2x three channel general-purpose 16-bit timers, integrated true RTC, a 10-bit ADC, and a 46 x 5 Segmented LCD controller.

SAM4CP16C operates from 1.62V to 3.6V and is available in 176-pin LQFP package.

sales@prom-elec.com / ООО ПРОМЭЛЕКТРОНИКА

1. Features

- Application/Master Core (CM4P0)
  - ARM Cortex-M4 running at up to 120 MHz(1)
  - Memory Protection Unit (MPU)
  - DSP Instruction
  - Thumb®-2 instruction set
  - Instruction and Data Cache Controller with 2 Kbytes Cache Memory
  - Memories
    - 1024 Kbytes of Embedded Flash for Program Code (I-Code bus) and Program Data (D-Code bus) with Built-in ECC (2-bit error detection and 1-bit correction per 128 bits)
    - 128 Kbytes of Embedded SRAM (SRAM0) for Program Data (System bus)
    - 8 Kbytes of ROM with embedded boot loader routines (UART) and In-Application Programming (IAP) routines
- Co-processor (CM4P1), provides ability to separate application, communication or metrology functions
  - ARM Cortex-M4F running at up to 120 MHz(1)
  - IEEE® 754 Compliant, Single-precision Floating-Point Unit (FPU)
  - DSP Instruction
  - Thumb-2 instruction set
  - Instruction and Data Cache Controller with 2 Kbytes of Cache Memory
  - Memories
    - 16 Kbytes of Embedded SRAM (SRAM1) for Program Code (I-Code bus) and Program Data (D-Code bus and System bus)
    - 8 Kbytes of Embedded SRAM (SRAM2) for Program Data (System bus)
- Symmetrical/Asynchronous Dual Core Architecture
  - Interrupt-based Interprocessor Communication
  - Asynchronous Clocking
  - One Interrupt Controller (NVIC) for each core
  - Each Peripheral IRQ routed to each NVIC Input
- G3-PLC Modem
  - Implements G3 CENELEC-A, FCC and ARIB profiles (ITU-T G.9903, June ’14)
  - Power Line Carrier Modem for 50 Hz and 60 Hz mains
  - G3-PLC coherent and differential modulation schemes available
  - Automatic Gain Control and continuous amplitude tracking in signal reception
  - Zero cross detection
  - Embedded PLC Analog Front End (AFE), requires only external discrete high efficient Class D Line Driver for signal injection
- Cryptography
  - High-performance AES 128 to 256 with various modes (GCM, CBC, ECB, CFB, CBC-MAC, CTR)
  - TRNG (up to 38 Mbit/s stream, with tested Diehard and FIPS)
  - Classical Public Key Crypto accelerator and associated ROM library for RSA, ECC, DSA, ECDSA
  - Integrity Check Module (ICM) based on Secure Hash Algorithm (SHA1, SHA224, SHA256), DMA-assisted
- Safety
  - Up to four Physical Anti-tamper Detection I/Os with Time Stamping and Immediate Clear of General Backup Registers
  - Security Bit for Device Protection from JTAG Accesses

- Shared System Controller
  - Power Supply
    - Embedded core and LCD voltage regulator for single-supply operation
    - Power-on-Reset (POR), Brownout Detector (BOD) and Dual Watchdog for safe operation
    - Ultra-low-power Backup mode (<0.5 µA Typical @ 25ºC)
  - Clock
    - Optional 3 to 20 MHz quartz or ceramic resonator oscillators with clock failure detection
    - Ultra-low-power 32.768 kHz crystal oscillator for RTC with frequency monitoring
    - 24 MHz quartz or ceramic resonator for GPLC peripheral
    - High-precision 4/8/12 MHz factory-trimmed internal RC oscillator with on-the-fly trimming capability
    - One high-frequency PLL up to 240 MHz, one 8 MHz PLL with internal 32 kHz input, as source for high-frequency PLL
    - Low-power slow clock internal RC oscillator as permanent clock
  - Ultra low-power RTC with Gregorian and Persian Calendar, Waveform Generation in Backup mode and Clock Calibration Circuitry for 32.768 kHz Crystal Frequency Compensation Circuitry
  - Up to 23 Peripheral DMA (PDC) Channels
- Shared Peripherals
  - One Low-power Segmented LCD Controller
    - Display capacity of 46 Segments and 5 common terminals
    - Software-selectable LCD output voltage (Contrast)
    - Low current consumption in Low-power mode
    - Can be used in Backup mode
  - Up to five USARTs with ISO7816, IrDA®, RS-485, SPI and Manchester Mode
  - Two 2-wire UARTs with one UART (UART1) supporting optical transceiver providing an electrically isolated serial communication with hand-held equipment, such as calibrators, compliant with ANSI-C12.18 or IEC62056-21 norms
  - Two 400 kHz Master/Slave and Multi-Master Two-wire Interfaces (I^2C compatible)
  - Up to six Serial Peripheral Interfaces (SPI)
  - Two 3-Channel 16-bit Timer/Counters with Capture, Waveform, Compare and PWM modes
  - Quadrature Decoder Logic and 2-bit Gray Up/Down Counter for Stepper Motor
  - 4-channel 16-bit Pulse Width Modulator
  - 32-bit Real-time Timer
- Analog Conversion Block
  - 8-channel, 500 kS/s, Low-power 10-bit SAR ADC with Digital Averager providing 12-bit Resolution at 30 kS/s
  - Software-controlled On-chip Reference ranging from 1.6V to 3.4V
  - Temperature Sensor and Backup Battery Voltage Measurement Channel
- Debug
  - Star Topology AHB-AP Debug Access Port Implementation with Common SW-DP / SWJ-DP Providing Higher Performance than Daisy-chain Topology
  - Debug Synchronization between both Cores (cross triggering to/from each core for Halt and Run Mode)
- I/O
  - Up to 69 I/O lines with External Interrupt Capability (edge or level sensitivity), Schmitt Trigger, Internal Pull-up/pull-down, Debouncing, Glitch Filtering and On-die Series Resistor Termination
- Packages
  - 176-lead LQFP, 24 x 24 mm, pitch 0.5 mm

Note: 1. 120 MHz: -40°C/+85°C, VDDCORE = 1.2V

1.1 Configuration Summary

Table 1-1 summarizes the SAM4CP16C device configurations.

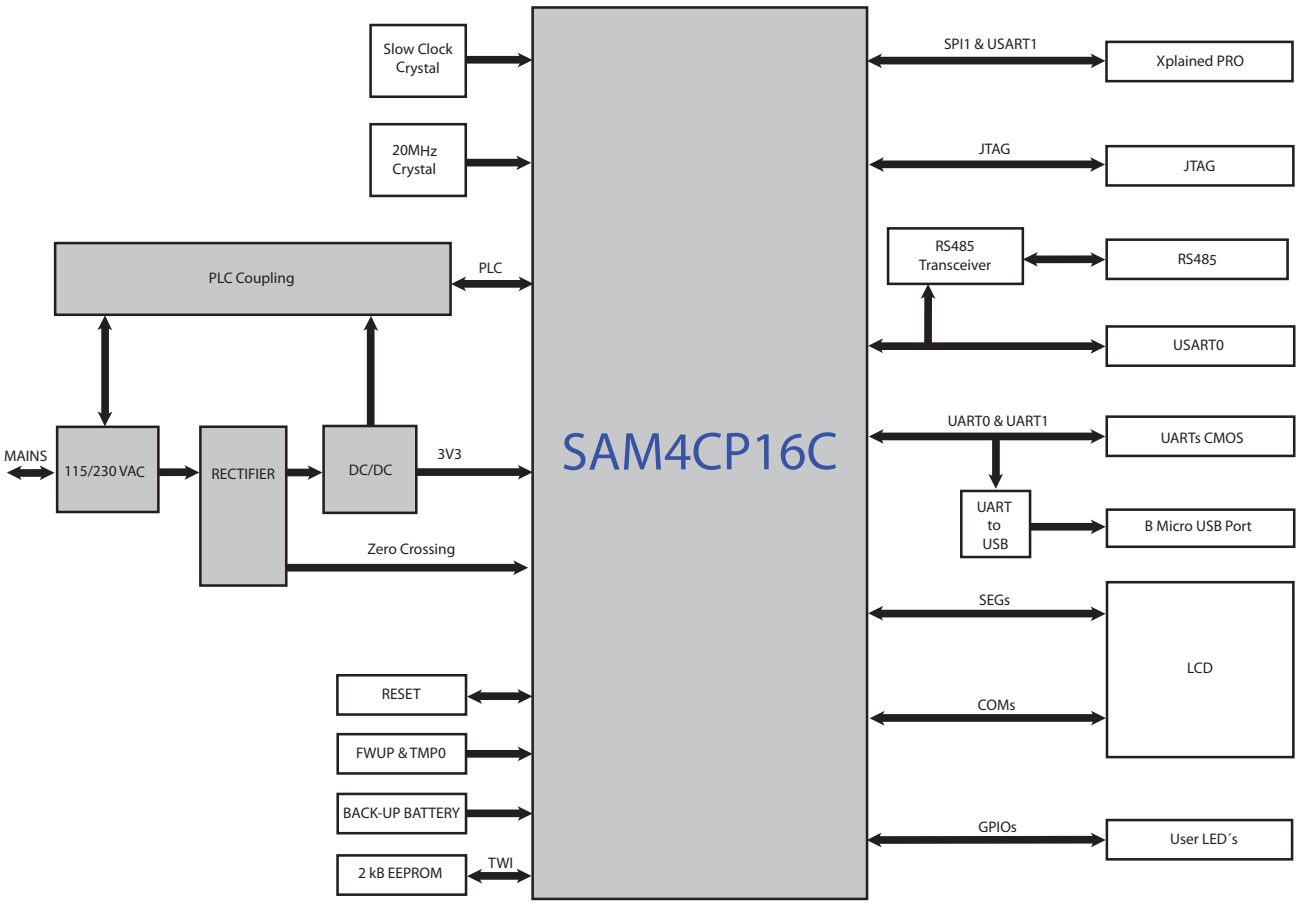
Table 1-1. Configuration Summary

Feature	SAM4CP16C
Flash	1024 Kbytes
SRAM	128 + 16 + 8 Kbytes
Package	LQFP 176
Number of PIOs	69
16-bit Timer	6 channels
16-bit PWM	4 channels
UART / USART	2/5
SPI (1)	6
TWI	2
10-bit ADC Channels(2)	7
Cryptography	AES, CPKCC, ICM (SHA), TRNG
Segmented LCD	46 segments x 5 commons
Anti-Tampering Inputs	4
Flash Page Size	512
Flash Pages	2048
Flash Lock Region Size	8192
Flash Lock Bits	128

Notes: 1. Using SPI mode of USARTs (1 SPI Controller + 5 USARTs with SPI mode).
2. One channel is reserved for internal temperature sensor and one channel for VDDBU measurement.

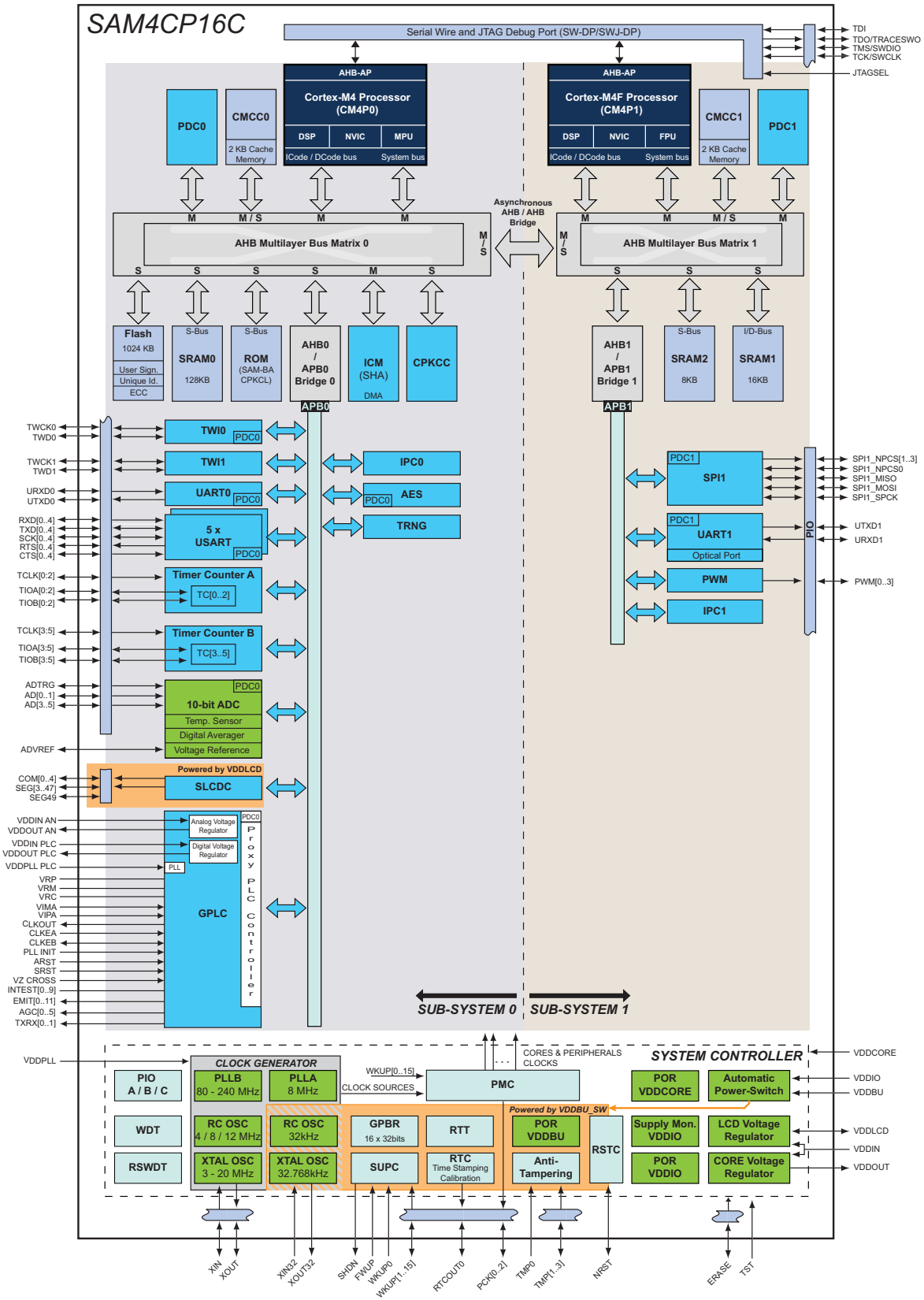
1.2 SAM4CP application block diagram

Figure 1-1. Block diagram of example of SAM4CP application



2. Block Diagram

Figure 2-1. SAM4CP16C 176-pin Block Diagram



3. Signal Description

Table 3-1 provides 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	See Table 5-1 on page 14	Power			3.0V to 3.6V
VDDBU		Power			1.6V to 3.6V
VDDIN		Power			2.5V to 3.6V(1)
VDDLCD		Power			2.4V to 3.6V(2)
VDDOUT		Power			1.2V
VDDPLL		Power			1.08V to 1.32V
VDDCORE		Power			1.08V to 1.32V
VDDPLL PLC		Power			1.2V
VDDIN PLC		Power			3.0V to 3.6V
VDDOUT PLC		Power			1.2V
VDDIN AN		Power			3.0V to 3.6V
VDDOUT AN		Power			1.2V
GND		Power			
AGND		Power			
Clocks, Oscillators and PLLs					
XIN	Main Crystal Oscillator Input	Analog Digital		VDDIO	
XOUT	Main Crystal Oscillator Output				
XIN32	Slow Clock Crystal Oscillator Input	Analog Digital		VDDBU	
XOUT32	Slow Clock Crystal Oscillator Output				
PCK0 - PCK2	Programmable Clock Output	Output		VDDIO	
CLKEA	PLC External Clock Input	Input		VDDIO	
CLKEB	PLC External Clock Input/Output	I/O		VDDIO	
CLKOUT	12 MHz External Clock Output	Output		VDDIO	
Supply Controller					
FWUP	Force Wake-up input	Input	Low	VDDBU	External Pull-up needed
TMP0	Anti-tampering Input 0	Input		VDDBU	
TMP1 - TMP3	Anti-tampering Inputs 1 to 3	Input		VDDIO	

Table 3-1. Signal Description List (Continued)

Signal Name	Function	Type	Active Level	Voltage reference	Comments
SHDN	Active Low Shutdown Control	Output		VDDBU	0:The device is in Backup mode 1:The device is running (not in Backup mode)
WKUP0	Wake-up Input 0	Input		VDDBU	
WKUP1 - WKUP15	Wake-up Inputs 1 to 15	Input		VDDIO	
Real Time Clock					
RTCOUT0	Programmable RTC waveform output	Output		VDDIO	To use this pin, the JTAG interface must be used in SWD mode
Serial Wire/JTAG Debug Port - SWJ-DP					
TCK/SWCLK	Test Clock/Serial Wire Clock	Input		VDDIO	
TDI	Test Data In	Input			
TDO/TRACESWO	Test Data Out / Trace Asynchronous Data Out	Output			
TMS/SWDIO	Test Mode Select Input / Serial Wire Input/Output	Input / I/O			
JTAGSEL	JTAG Selection	Input	High	VDDBU	Permanent Internal pull-down[3]
Flash Memory					
ERASE	Flash and NVM Configuration Bits Erase Command	Input	High	VDDIO	Permanent Internal pull-down[3]
Reset/Test					
NRST	Synchronous Microcontroller Reset	I/O	Low	VDDIO	Permanent Internal pull-up[3]
TST	Test Select	Input		VDDBU	Permanent Internal pull-down[3]
ARST	PLC Asynchronous Reset	Input	Low	VDDIO	Permanent Internal pull-up[4]
SRST	PLC Synchronous Reset	Input	Low	VDDIO	Permanent Internal pull-up[4]
PLL INIT	PLC PLL Initialization Signal	Input	Low	VDDIO	Permanent Internal pull-up[4]
GPLC (G3 Power Line Communications) Transceiver					
EMIT0 - EMIT11	PLC Transmission ports[5]	Output		VDDIO	Different configurations allowed depending on external topology and net behaviour

Table 3-1. Signal Description List (Continued)

Signal Name	Function	Type	Active Level	Voltage reference	Comments
AGC0 - AGC5	PLC Automatic Gain Control	Output		VDDIO	
TXRX0 - TXRX1	PLC Ext. coupling TxRx control	Output		VDDIO	
VZ CROSS	Mains Zero-Cross Detection Signal[6]	Input		VDDIO	Permanent Internal pull-down[4]
VIMA	Negative Differential Voltage Input	Input		VDDOUT AN	
VIPA	Positive Differential Voltage Input	Input		VDDOUT AN	
VRP	Internal Reference “Plus” Voltage	Output		VDDOUT AN	
VRM	Internal Reference “Minus” Voltage	Output		VDDOUT AN	
VRC	Common-mode Voltage	Output		VDDOUT AN	
INTEST0	PLC Internal Test	Input		VDDIO	Permanent Internal pull-up[4] This pin must be connected to INTEST5 (pin 144)
INTEST1	PLC Internal Test	Input		VDDIO	Permanent Internal pull-up[4] This pin must be connected to INTEST6 (pin 45)
INTEST2	PLC Internal Test	Input		VDDIO	Permanent Internal pull-up[4] This pin must be connected to INTEST7 (pin 176)
INTEST3	PLC Internal Test	Output		VDDIO	This pin must be connected to INTEST8 (pin 111)
INTEST4	PLC Internal Test	Output		VDDIO	This pin must be connected to INTEST9 (pin 20)
INTEST5	PLC Internal Test	Output		VDDIO	This pin must be connected to INTEST0 (pin 94)
INTEST6	PLC Internal Test	Output		VDDIO	This pin must be connected to INTEST1 (pin 95)
INTEST7	PLC Internal Test	Output		VDDIO	This pin must be connected to INTEST2 (pin 97)

Table 3-1. Signal Description List (Continued)

Signal Name	Function	Type	Active Level	Voltage reference	Comments
INTEST8	PLC Internal Test	Input		VDDIO	This pin must be connected to INTEST3 (pin 99)
INTEST9	PLC Internal Test	Input		VDDIO	This pin must be connected to INTEST4 (pin 4)
PIO Controller - PIOA - PIOB - PIOC					
PA0 - PA4, PA9 - PA31	Parallel IO Controller A	Digital I/O		VDDIO	
PB0 - PB29, PB31	Parallel IO Controller B				
PC0 - PC9	Parallel IO Controller C				
Universal Asynchronous Receiver Transmitter - UARTx					
URXDx	UART Receive Data	Input		VDDIO	Analog Mode for Optical Receiver
UTXDx	UART Transmit Data	Output			
Universal Synchronous Asynchronous Receiver Transmitter - USARTx					
SCKx	USARTx Serial Clock	I/O		VDDIO	
TXDx	USARTx Transmit Data	I/O			
RXDx	USARTx Receive Data	Input			
RTSx	USARTx Request To Send	Output			
CTSx	USARTx Clear To Send	Input			
Timer/Counter - TC					
TCLKx	TC Channel x External Clock Input	Input		VDDIO	
TIOAx	TC Channel x I/O Line A	I/O			
TIOBx	TC Channel x I/O Line B	I/O			
Serial Peripheral Interface - SPI					
SPI1_MISO	Master In Slave Out	I/O		VDDIO	
SPI1_MOSI	Master Out Slave In	I/O			
SPI1_SPCK	SPI Serial Clock	I/O			
SPI1_NPCS0	SPI Peripheral Chip Select 0	I/O	Low		NPCS0 is also NSS for slave mode
SPI1_NPCS1 - SPI1_NPCS3	SPI Peripheral Chip Select	Output	Low		

Table 3-1. Signal Description List (Continued)

Signal Name	Function	Type	Active Level	Voltage reference	Comments
Pulse Width Modulation Controller - PWMC					
PWMx	PWM Waveform Output for channel x	Output		VDDIO	
Segmented LCD Controller - SLCDC					
COM[4:0]	Common Terminals	Output		VDDIO	
SEG49 SEG[47:3]	Segment Terminals	Output			
Two-Wire Interface - TWI					
TWDx	TWIx Two-wire Serial Data	I/O		VDDIO	
TWCKx	TWIx Two-wire Serial Clock	I/O			
Analog					
ADVREF	External Voltage Reference for ADC	Analog Input			
10-bit Analog-to-Digital Converter - ADC					
AD0 - AD1 AD3 - AD5	Analog Inputs	Analog Digital		VDDIO	ADC input range limited to [0 - ADVREF]
ADTRG	ADC Trigger	Input			
Fast Flash Programming Interface - FFPI					
PGMEN0- PGMEN1	Programming Enabling	Input		VDDIO	
PGMM0-PGMM3	Programming Mode				
PGMD0-PGMD15	Programming Data	I/O			
PGMRDY	Programming Ready	Output	High		
PGMNVALID	Data Direction		Low		
PGMNOE	Programming Read	Input	Low		
PGMNCMD	Programming Command		Low		

Notes:
1. VDDLCD must be inferior or equals to (VDDIO/VDDIN - 100mv) if VDDLCD is powered externally.
2. See “Typical Powering Schematics” Section for restrictions on voltage range of Analog Cells.
3. See Table 45-5 on page 927.
4. See Table 45-11 on page 936.
5. Different configurations allowed depending on external topology and net behavior.
6. Depending on whether an isolated or a non-isolated power supply is being used, isolation of this pin should be taken into account in the circuitry design. Please refer to the reference design of the evaluation board for further information.

4. Package and Pinout

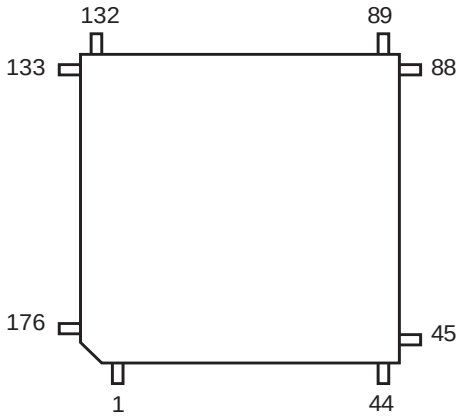
4.1 SAM4CP16C Package and Pinout

4.1.1 176-Lead LQFP Package Outline

The 176-lead LQFP package has a 0.5 mm pitch and respects Green standards.

Figure 4-1 shows the orientation of the 176-lead LQFP package. Refer to the section "Mechanical Characteristics" for the 176-lead LQFP package mechanical drawing.

Figure 4-1. Orientation of the 176-lead LQFP Package



4.1.2 176-lead LQFP Pinout

Table 4-1. SAM4CP16C 176-lead LQFP Pinout

1	VDDIO	45	INTEST6	89	PA30/XOUT	133	PA15
2	PA2	46	TDI/PB0	90	VDDIO	134	PA16
3	PB6	47	NC	91	NC	135	PA17
4	INTEST4	48	TCK/SWCLK/PB3	92	PA31/XIN	136	VDDIO
5	PB7	49	TMS/SWDIO/PB2	93	CLKOUT	137	ADVREF
6	PB18	50	ERASE/PC9	94	INTEST0	138	GND
7	GND	51	TDO/TRACESWO/ PB1/RTCOUT0	95	INTEST1	139	VDDIO
8	PB19	52	PC1	96	GND	140	PB31/AD5
9	PB8	53	NC	97	INTEST2	141	PB23/AD4
10	AGND	54	NC	98	GND	142	PB13/AD3
11	VDDOUT AN	55	NC	99	INTEST3	143	GND
12	VIMA	56	ARST	100	VDDPLL	144	INTEST5
13	VIPA	57	PLL INIT	101	PC8	145	PA4/AD1
14	VDDOUT AN	58	PC6	102	PC5	146	EMIT8
15	AGND	59	VDDIO	103	PC4	147	PA12/AD0
16	VRP	60	GND	104	PC3	148	VDDIN
17	VRM	61	CLKEA	105	VDDIO	149	EMIT9
18	VRC	62	VDDIO	106	PC2	150	VDDIN
19	PB22	63	CLKEB	107	PA29	151	EMIT10
20	INTEST9	64	VDDIO	108	PA28	152	VDDOUT
21	PB25	65	VDDBU	109	GND	153	EMIT11
22	VDDIN AN	66	FWUP	110	PA27	154	PB21
23	PB24	67	JTAGSEL	111	INTEST8	155	PB20
24	VDDCORE	68	SHDN	112	VDDCORE	156	VDDIO
25	AGND	69	TST	113	EMIT0	157	VDDCORE
26	PB29	70	VDDPLL PLC	114	PA3	158	PA0
27	PB9	71	TMP0/WKUP0	115	PA21	159	VDDOUT PLC
28	PB10	72	GND	116	PA22	160	TXRX0
29	PB11	73	XIN32	117	EMIT1	161	TXRX1
30	VDDIN AN	74	VDDIN PLC	118	EMIT2	162	AGC2
31	PB12	75	VDDIN PLC	119	EMIT3	163	PB27/TMP2
32	PB14	76	XOUT32	120	VDDIO	164	AGC5
33	PB15	77	GND	121	GND	165	VDDLCD
34	PA26	78	VDDOUT PLC	122	EMIT4	166	AGC1
35	GND	79	GND	123	EMIT5	167	AGC4
36	PA25	80	NC	124	PA23	168	AGC0
37	VDDIO	81	PB4	125	EMIT6	169	AGC3
38	PA24	82	VDDCORE	126	PA9	170	PB26
39	VZ CROSS	83	PB5	127	PA10	171	VDDIO
40	PA20	84	SRST	128	PA11	172	PB28/TMP3
41	NC	85	PC7	129	EMIT7	173	PB16/TMP1
42	PA19	86	PC0	130	PA13	174	PA1
43	PA18	87	NRST	131	PA14	175	PB17
44	NC	88	VDDIO	132	GND	176	INTEST7

5. Power Supply and Power Control

5.1 Power Supplies

The SAM4CP16C has several types of power supply pins. In most cases, a single supply scheme for all power supplies (except VDDBU) is possible. Figure 5.1.1 below shows power domains according to the different power supply pins.

Figure 5-1. Power Domains

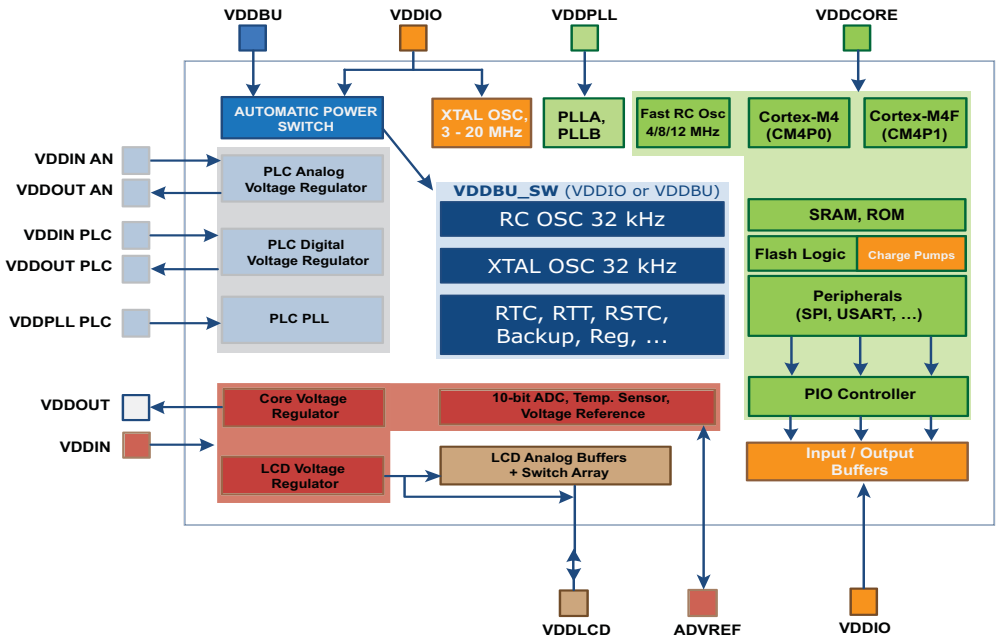


Table 5-1. Power Supply Voltage Ranges

Power Supplies	Ranges	Comments
VDDIO	3.0V to 3.6V	Flash memory charge pumps supply for erase and program operations, and read operation Input/Output buffers supply Oscillator pads supply Restrictions on range may apply. Refer to the section "Electrical Characteristics"
VDDBU	1.6V to 3.6V	Backup area power supply VDDBU is automatically disconnected when VDDIO is present (>1.9V)
VDDIN	2.5V to 3.6V	Core Voltage Regulator supply LCD Regulator supply ADC and Programmable Voltage Reference supply Restrictions on range may apply. Refer to the section "Electrical Characteristics"
VDDLCD	2.4V to 3.6V	LCD Voltage Regulator output External LCD power supply (LCD regulator not used) VDDIO/VDDIN must be supplied when the LCD Controller is used
VDDOUT	1.2V	Core Voltage Regulator Output. 120mA output current
VDDPLL	1.08V to 1.32V	PLLA and PLLB supply
VDDCORE	1.08V to 1.32V	Core logic, processors, memories and analog peripherals supply
VDDPLL PLC	1.2V	PLC PLL

Table 5-1. Power Supply Voltage Ranges (Continued)

Power Supplies	Ranges	Comments
VDDIN PLC	3.0V to 3.6V	PLC Digital LDO Regulator input
VDDOUT PLC	1.2V	PLC Digital LDO Regulator output
VDDIN AN	3.0V to 3.6V	PLC Analog LDO Regulator input
VDDOUT AN	1.2V	PLC Analog LDO Regulator output
GND	-	Digital Ground
AGND	-	Analog Ground

Separate pins are provided for GND and AGND grounds. Layout considerations should be taken into account to reduce interference. Ground pins should be connected as shortly as possible to the system ground plane.

5.1.1 Core Voltage Regulator

The core voltage regulator is managed by the Supply Controller.

It features two operating modes:

- In Normal mode, the quiescent current of the voltage regulator is less than 500 µA when sourcing maximum load current, i.e., 120 mA. 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 100 nA while its output (VDDOUT) is driven internally to GND.

The default output voltage is 1.20 V and the start-up time to reach Normal mode is less than 500 µs.

For further information, refer to “Core Voltage Regulator Characteristics” in the section “Electrical Characteristics”

5.1.2 LCD Voltage Regulator

The SAM4CP16C embeds an adjustable LCD voltage regulator that is managed by the Supply Controller.

This internal regulator is designed to supply the Segment LCD outputs. The LCD regulator output voltage is software selectable with 16 levels to adjust the display contrast.

If not used, its output (VDDLCD) can be bypassed (Hi-z mode) and an external power supply can be provided onto the VDDLCD pin. In this case, VDDIO still needs to be supplied.

The LCD voltage regulator can be used in all power modes (Backup, Wait, Sleep and Active).

For further information, refer to “LCD Voltage Regulator Characteristics” in the section “Electrical Characteristics”

5.1.3 PLC Voltage Regulators

The SAM4CP embeds two PLC-dedicated voltage regulators, PLC Analog Voltage Regulator (VDDIN AN) and PLC Digital Voltage Regulator (VDDIN PLC). These internal regulators are designed to supply the PLC peripheral block in an efficient way trying to minimize noise coupling in power supply.

5.1.4 Automatic Power Switch

The SAM4CP16C features an automatic power switch between VDDBU and VDDIO. When VDDIO is present, the backup zone power supply is powered by VDDIO and current consumption on VDDBU is close to zero (around 100 nA, typ.). When VDDIO is removed, the backup area of the device is supplied from VDDBU. Switching between VDDIO and VDDBU is transparent to the user.

5.1.5 Typical Powering Schematics

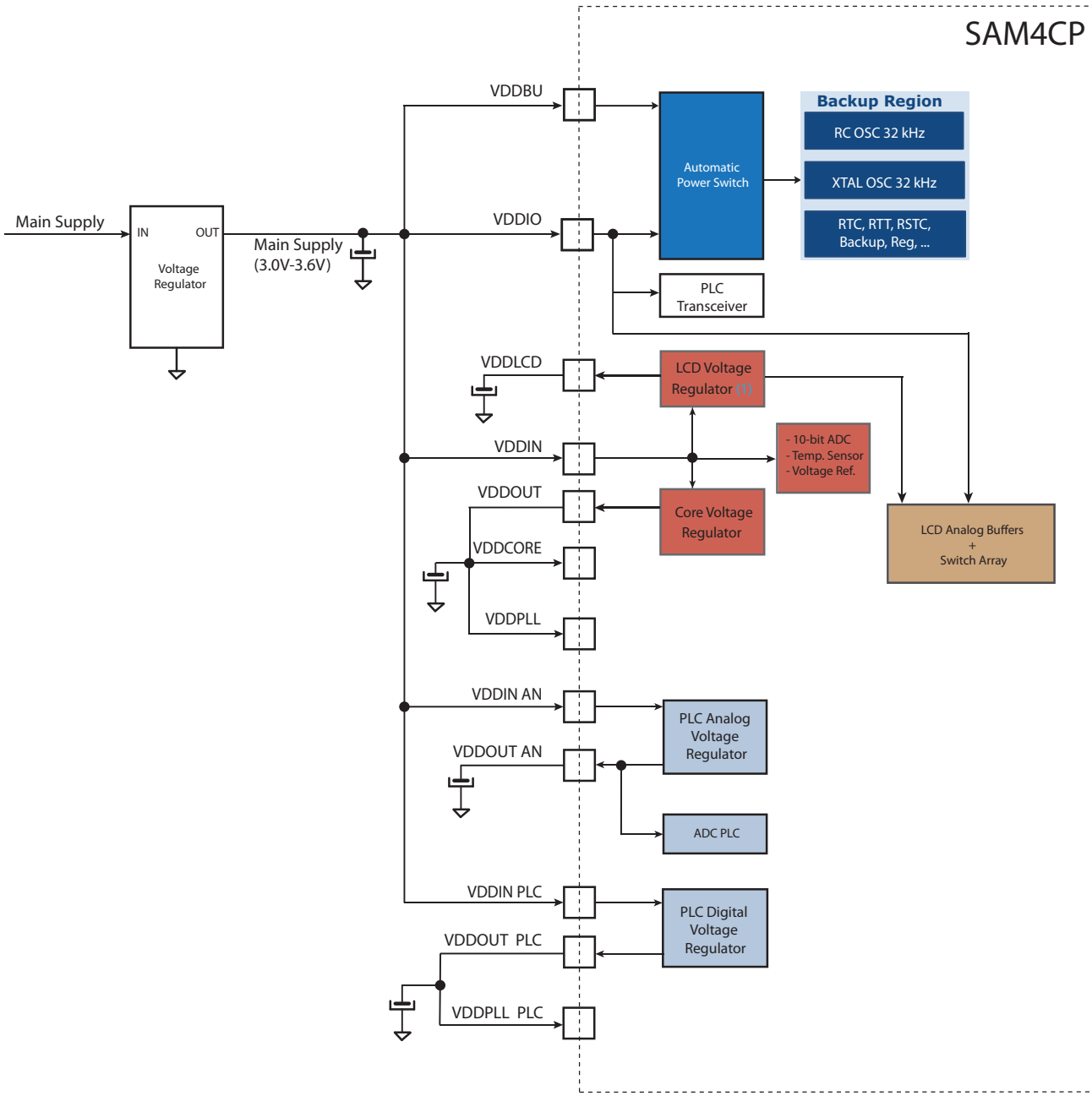
The SAM4CP supports 3.0V to 3.6V single-supply operation. Restrictions on this range may apply depending on enabled features. Refer to the section Electrical Characteristics.

Figure 5-2, Figure 5-3 and Figure 5-4 show simplified schematics of the power section.

5.1.5.1 Single Supply Operation

Figure 5-2 below shows a typical power supply scheme with a single power source. VDDIO, VDDIN, VDDBU, VDDIN AN and VDDIN PLC are derived from the main power source (typically a 3.3V regulator output) while VDDCORE, VDDPLL, VDDLCD and VDDPLL PLC are fed by the embedded regulator outputs.

Figure 5-2. Single Supply Operation



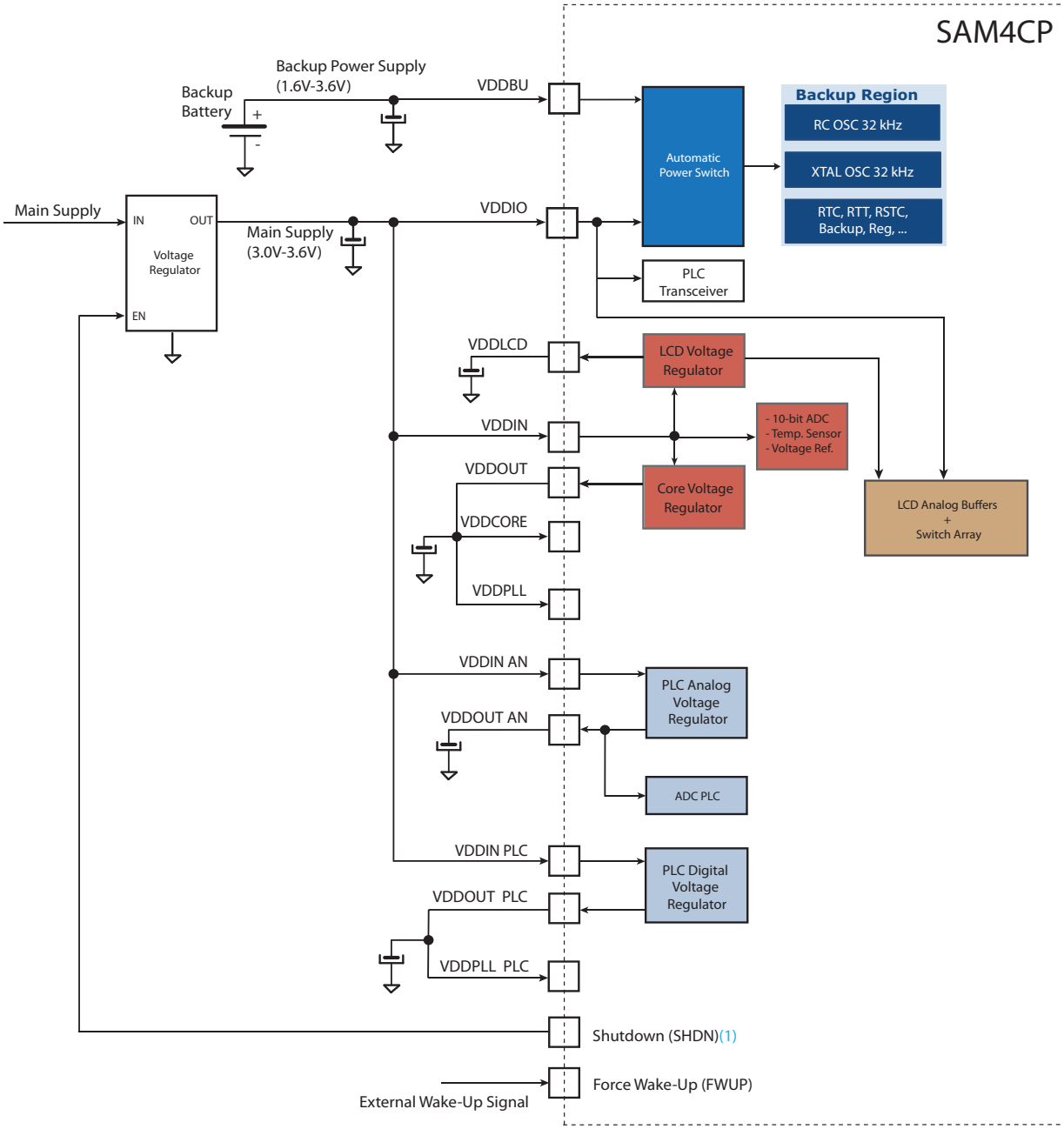
Notes: 1. Internal LCD Voltage Regulator can be disabled to save its operating current. VDDLCD must then be provided externally.

5.1.5.2 Single Supply Operation with Backup Battery

Figure 5-3 shows the single-supply operation schematic from Figure 5-2, improved by adding a backup capability. VDDBU is supplied with a separate backup battery while VDDIO, VDDIN, VDDIN AN and VDDIN PLC are still connected to the main power source. Note that the TMP1 to TMP3 and RTCOUT0 pins cannot be used in Backup mode as they are referred to VDDIO, which is not powered in this application case. To keep using these pins in Backup mode, VDDIO must be maintained.

Note that PLC transceiver is not functional when working in backup mode.

Figure 5-3. Single Supply Operation with Backup Battery



Notes: 1. Example with the SHDN pin used to control the main regulator enable pin. SHDN defaults to VDDBU at startup and when the device wakes up from a wake-up event (external pin, RTC alarm, etc). When the device is in Backup mode, SHDN defaults to 0.

5.1.5.3 Single Power Supply using One Main Battery and LCD Controller in Backup Mode

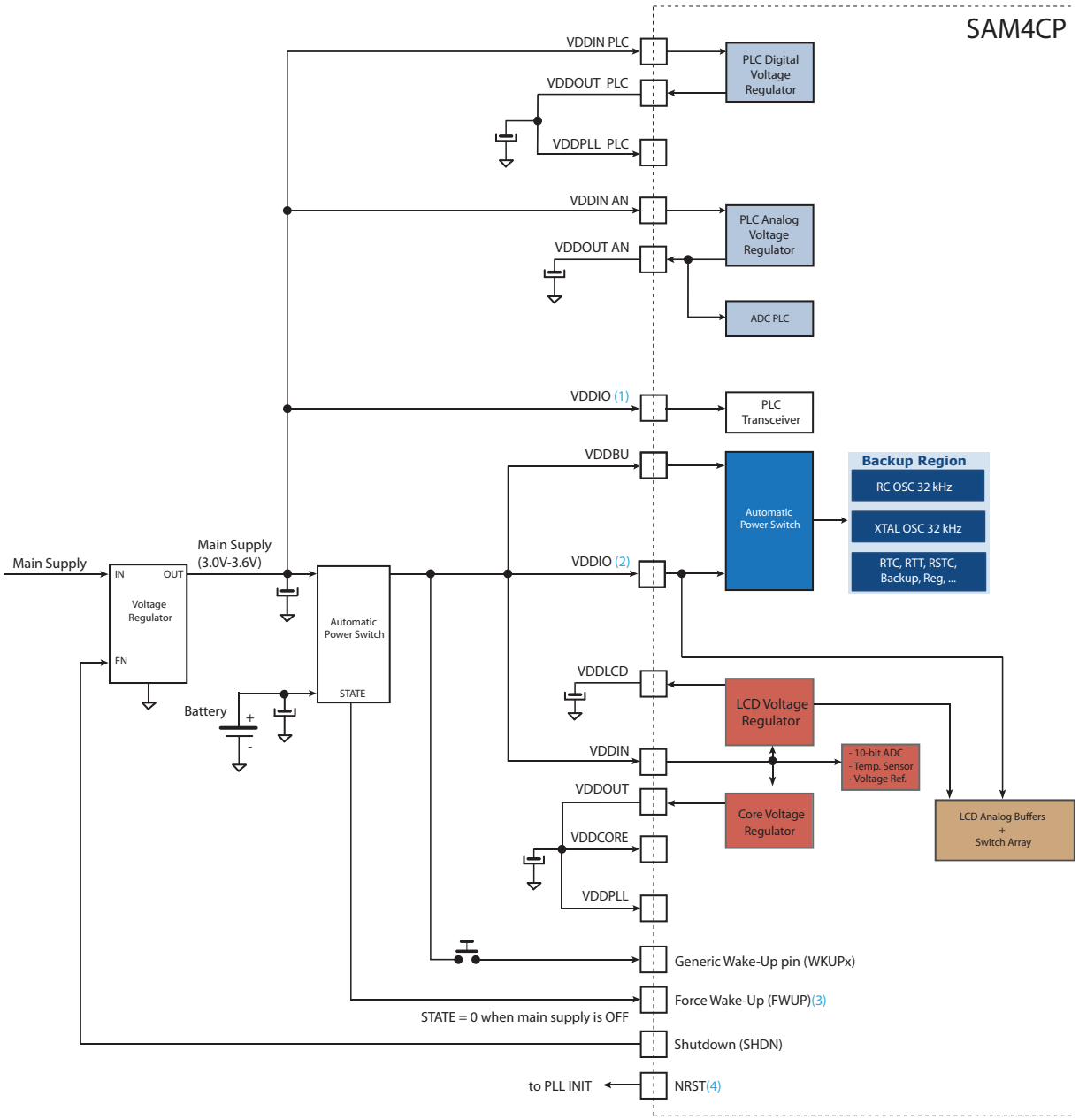
Figure 5-4 below shows a typical power supply scheme that maintains VDDBU, VDDIO, and VDDLCD when entering Backup mode. This is useful to enable the display and/or some supplementary wake-up sources in Backup mode when the main voltage is not present.

In this power supply scheme, the SAM4CP can wake up both from an internal wake-up source, such as RTT, RTC and VDDIO Supply Monitor, and from an external source, such as generic wake-up pins (WKUPx), anti-tamper inputs (TMPx) or force wake-up (FWUP).

Note that PLC transceiver is not functional when working in backup mode.

Note: The VDDIO supply monitor only wakes up the device from Backup mode on a negative-going VDDIO supply (as system alert). As a result, the supply monitor cannot be used to wake up the device when the VDDIO supply is rising at power cycle. See the section “Supply Controller (SUPC)” for more information on the VDDIO supply monitor.

Figure 5-4. Single Power Supply using Battery and LCD Controller in Backup Mode



Notes:
1. VDDIO corresponds to the following pins: 37, 64, 90, 105, 120, 139, 156 and 171.
2. VDDIO corresponds to the following pins: 1, 59, 62, 88 and 136.
3. The STATE output of the automatic power switch indicates to the device that the main power is back and forces its wake-up.
4. The NRST pin integrates a permanent pull-up resistor to VDDIO of typical 100 kΩ. When used to control PLL INIT as in the example, external design should take into account minimizing leakage currents.

5.1.5.4 Wake-up, Anti-tamper and RTCOUT0 Pins

In all power supply figures shown above, if generic wake-up pins other than WKUP0/TMP0 are used either as a wake-up or a fast startup input, or as anti-tamper inputs, VDDIO must be present. This also applies to the RTCOUT0 pin.

5.1.5.5 General-purpose IO (GPIO) State in Low-power Modes

In dual-power supply schemes shown in Figure 5-3 and Figure 5-4, where Backup or Wait mode must be used, configuration of the GPIO lines is maintained in the same state as before entering Backup or Wait mode. Thus, to avoid extra current consumption on the VDDIO power rail, the user must configure the GPIOs either as an input with pull-up or pull-down enabled, or as an output with low or high level to comply with external components.

5.1.5.6 Default General-purpose IOs (GPIO) State after Reset

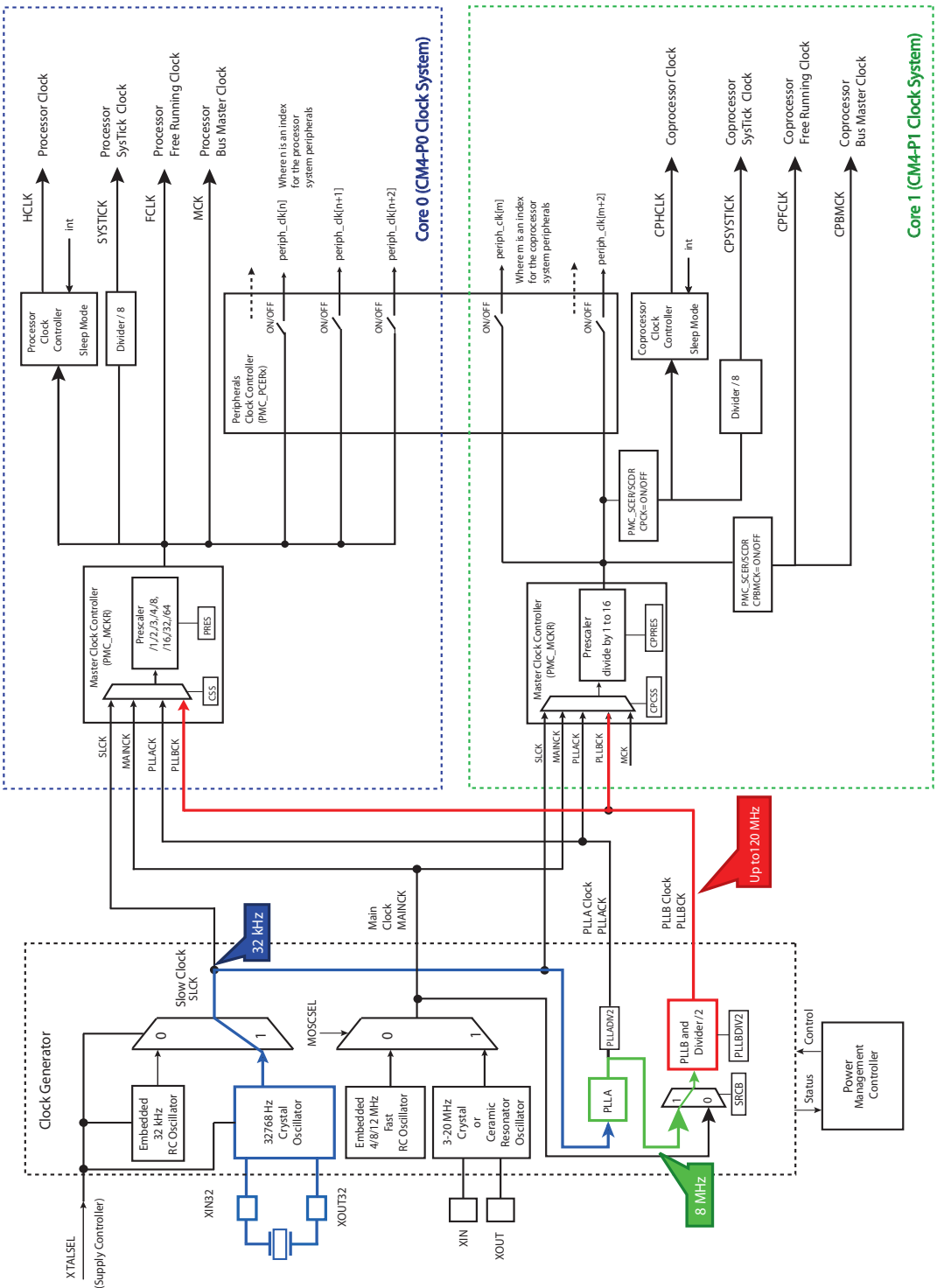
The reset state of the GPIO lines after reset is given in Table 11-5, “Multiplexing on PIO Controller A (PIOA)”, Table 11-6, “Multiplexing on PIO Controller B (PIOB)” and Table 11-7, “Multiplexing on PIO Controller C (PIOC)”. For further details about the GPIO and system lines, wake-up sources and wake-up time, and typical power consumption in different low-power modes, refer to Table 5-2, “Low-power Mode Configuration Summary”.

5.2 Clock System

Figure 5-5 illustrates the SAM4CP16C clock system with single crystal operation:

- The 32 kHz crystal oscillator can be the source clock of the 8 MHz digital PLL (PLLA).
- The 8 MHz clock can feed the high frequency PLL (PLLB) input.
- The output of the PLLB can be used as a main clock for both cores and the peripherals.

Figure 5-5. SAM4CP16C Global Clock System



5.3 System State at Power-up

5.3.1 Device Configuration after the First Power-up

At the first power-up, the SAM4CP16C boots from the ROM. The device configuration is defined by SAM-BA® boot program.

5.3.2 Device Configuration after a Power Cycle when Booting from Flash Memory

After a power cycle of all the power supply rails, the system peripherals, such as the Flash Controller, the Clock Generator, the Power Management Controller and the Supply Controller, are in the following states:

- Slow Clock (SLCK) source is the internal 32 kHz RC Oscillator (32 kHz crystal oscillator is disabled).
- Main Clock (MAINCK) source is set to the 4 MHz internal RC Oscillator.
- 3 - 20 MHz crystal oscillator and PLLs are disabled.
- Core Brownout Detector and Core Reset are enabled.
- Backup Power-on-reset is enabled.
- VDDIO Supply Monitor is disabled.
- Flash Wait State (FWS) bit in the EEFC Flash Mode Register is set to 0.
- Core 0 Cache Controller (CMCC0) is enabled (only used if the application link address for the Core 0 is 0x11000000).
- Sub-system 1 is in the reset state and not clocked.

5.3.3 Device Configuration after a Reset

After a reset or a wake-up from Backup mode, the following system peripherals default to the same state as after a power cycle:

- Main Clock (MAINCK) source is set to the 4 MHz internal RC oscillator.
- 3 - 20 MHz crystal oscillator and PLLs are disabled.
- Flash Wait State (FWS) bit in the EEFC Flash Mode Register is set to 0.
- Core 0 Cache Controller (CMCC0) is enabled (only used if the application link address for the Core 0 is 0x11000000).
- Sub-system 1 is in the reset state and not clocked.

The states of the other peripherals are saved in the backup area managed by the Supply Controller as long as VDDBU is maintained during device reset:

- Slow Clock (SLCK) source selection is written in SUPC_CR.XTALSEL.
- Core Brownout Detector enable/disable is written in SUPC_MR.BODDIS.
- Backup Power-on-reset enable/disable is written in the SUPC_MR.BUPPOREN.
- VDDIO Supply Monitor mode is written in the SUPC_SMMR.

5.4 Active Mode

Active mode is the normal running mode, with the single core or the dual cores executing code. The system clock can be the fast RC oscillator, the main crystal oscillator or the PLLs. The Power Management Controller (PMC) can be used to adapt the frequency and to disable the peripheral clocks when unused.

5.5 Low-power Modes

The various low-power modes (Backup, Wait and Sleep modes) of the SAM4CP16C are described below. Note that the Segmented LCD Controller can be used in all low-power modes.

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 to the design of application state machines. This is due to the fact that the WFE instruction is associated with an event flag of the Cortex core that 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. When an event occurs just before WFE execution, the processor takes it into account and does not enter Low-power mode. Atmel has made provision to avoid using the WFE instruction. The workarounds to ease application design, including the use of the WFE instruction, are given in the following description of the low-power mode sequences.

5.5.1 Backup Mode

The purpose of Backup mode is to achieve the lowest possible power consumption in a system that executes periodic wake-ups to perform tasks but which does not require fast start-up time. The total current consumption is 0.5 µA typical on VDDBU.

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

Wake-up from Backup mode can be done through the Force Wake-up (FWUP) pin, WKUP0, WKUP1 to WKUP15 pins, the VDDIO Supply Monitor (SM) if VDDIO is supplied, or through an RTT or RTC wake-up event. Wake-up pins multiplexed with anti-tampering functions are additional possible sources of a wake-up if an anti-tampering event is detected. The TMP0 pad is supplied by the backup power supply (VDDBU). Other anti-tamper input pads are supplied by VDDIO.

The LCD Controller can be used in Backup mode. The purpose is to maintain the displayed message on the LCD display after entering Backup mode. The current consumption on VDDIN to maintain the LCD is 10 µA typical.

Please refer to the section “Electrical Characteristics”.

In case the VDDIO power supply is maintained with VDDBU when entering Backup mode, it is up to the application to configure all PIO lines in a stable and known state to avoid extra power consumption or possible current path with the input/output lines of the external on-board devices.

5.5.1.1 Entering and Exiting Backup Mode

To enter Backup mode, follow the steps in the sequence below:

1. Depending on the application, set the PIO lines in the correct mode and configuration (input pull-up or pull-down, output low or high levels).
2. Disable the Main Crystal Oscillator (enabled by SAM-BA boot if the device is booting from ROM).
3. Configure PA30/PA31 (XIN/XOUT) into PIO mode depending on their use.
4. Disable the JTAG lines using the SFR1 register in Matrix 0 (by default, internal pull-up or pull-down is disabled on JTAG lines).
5. Enable the RTT in 1 Hz mode.
6. Disable Normal mode of the RTT (RTT will run in 1 Hz mode).
7. To reduce power consumption, disable the POR backup if not needed.

Note: If VDDBU is likely to go below the POR threshold but not completely down to GND, disabling the PORBU creates unexpected behavior of the VDDBU domain logic. PORBU can be disabled if VDDBU is known to be stable above the PORBU threshold.

8. Disable the Core brownout detector.
9. Select one of the following methods to complete the sequence:
    a. To enter Backup mode using the VROFF bit:
        - Write a 1 to the VROFF bit of SUPC_CR.

b. 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.

After this step, the core voltage regulator is shut down and the SHDN pin goes low. The digital internal logic (cores, peripherals and memories) is not powered. The LCD controller can be enabled if needed before entering Backup mode.

Whether the VROFF bit or the WFE instruction was used to enter Backup mode, the system exits Backup mode if one of the following enabled wake-up events occurs:

- WKUP[0-15] pins.
- Force Wake-up pin.
- VDDIO Supply Monitor (if VDDIO is present, and VDDIO supply falling).
- Anti-tamper event detection.
- RTC alarm.
- RTT alarm.

After exiting Backup mode, the device is in the reset state. Only the configuration of the backup area peripherals remains unchanged.

Note that the device does not automatically enter Backup mode if VDDIN is disconnected, or if it falls below minimum voltage. The Shutdown pin (SHDN) remains high in this case.

For current consumption in Backup mode, refer to the section “Electrical Characteristics”.

5.5.2 Wait Mode

The purpose of Wait mode is to achieve very low power consumption while maintaining the whole device in a powered state for a start-up time of less than 10 µs. For current consumption in Wait mode, refer to the section “Electrical Characteristics”.

In Wait mode, the bus and peripheral clocks of Sub-system 0 and Sub-system 1 (MCK/CPBMCK), the clocks of Core 0 and Core 1 (HCLK/CPHCLK) are stopped when Wait mode is entered (see Section 5.5.2.1). However, the power supply of core, peripherals and memories are maintained using Standby mode of the core voltage regulator.

The SAM4CP16C is able to handle external and internal events in order to perform a wake-up. This is done by configuring the external WKUPx lines as fast startup wake-up pins (refer to Section 5.7 “Fast Start-up”). RTC alarm, RTT alarm and anti-tamper events can also wake up the device.

Wait mode can be used together with Flash in Read-Idle mode, Standby mode or Deep Power-down mode to further reduce the current consumption. Flash in Read-Idle mode provides a faster start-up; Standby mode offers lower power consumption. For power consumption details, see subsection “Power Supply Current Consumption” in “Electrical Characteristics” section.

5.5.2.1 Entering and Exiting Wait Mode

1. Stop Sub-system 1.
2. Select the 4/8/12 MHz fast RC Oscillator as Main Clock(1).
3. Depending on the application, set the PIO lines in the correct mode and configuration (input pull-up or pull-down, output low or high level).
4. Disable the Main Crystal Oscillator (enabled by SAM-BA boot if device is booting from ROM).
5. Configure PA30/PA31 (XIN/XOUT) into PIO mode according to their use.
6. Disable the JTAG lines using the SFR1 register in Matrix 0 (by default, internal pull-up or pull-down is disabled on JTAG lines).
7. Set the FLPM field in the PMC Fast Startup Mode Register (PMC_FSMR)(2).
8. Set the Flash Wait State (FWS) bit in the EEFC Flash Mode Register to 0.
9. Select one of the following methods to complete the sequence:

a. To enter Wait mode using the WAITMODE bit:
   - Set the WAITMODE bit to 1 in the PMC Main Oscillator Register (CKGR_MOR).
   - Wait for Master Clock Ready MCKRDY = 1 in the PMC Status Register (PMC_SR).

b. To enter Wait mode using the WFE instruction:
   - Select the 4/8/12 MHz fast RC Oscillator as Main Clock.
   - Set the FLPM field in the PMC Fast Startup Mode Register (PMC_FSMR).
   - Set Flash Wait State at 0.
   - Set the LPM bit in the PMC Fast Startup Mode Register (PMC_FSMR).
   - Write a 0 to the SLEEPDEEP bit of the Cortex-M4 processor.
   - Execute the Wait-For-Event (WFE) instruction of the processor.

Notes:
1. Any frequency can be chosen. The 12 MHz frequency will provide a faster start-up compared to the 4 MHz, but with the increased current consumption (in the µA range). See electrical characteristics of the product.
2. Depending on the Flash Low-power Mode (FLPM) value, the Flash enters three different modes:
   • If FLPM = 0, the Flash enters Stand-by mode (Low consumption).
   • If FLPM = 1, the Flash enters Deep Power-down mode (Extra low consumption).
   • If FLPM = 2, the Flash enters Idle mode. Memory is ready for Read access.

Whether the WAITMODE bit or the WFE instruction was used to enter Wait mode, the system exits Wait mode if one of the following enabled wake-up events occurs:

- WKUP[0-15] pins in Fast wake-up mode.
- Anti-tamper event detection.
- RTC alarm.
- RTT alarm.

After exiting Wait mode, the PIO controller has the same configuration state as before entering Wait mode. The SAM4CP16C is clocked back to the RC oscillator frequency which was used before entering Wait mode. The core will start fetching from Flash at this frequency. Depending on the configuration of the Flash Low-power Mode (FLPM) bits used to enter Wait mode, the application has to reconfigure it back to Read-idle mode.

5.5.3 Sleep Mode

The purpose of Sleep mode is to optimize power consumption of the device versus response time. In this mode, only the core clocks of CM4P0 and/or CM4P1 are stopped. Some of the peripheral clocks can be enabled depending on the application needs. The current consumption in this mode is application dependent. This mode is entered using Wait for Interrupt (WFI) or Wait for Event (WFE) instructions of the Cortex-M4.

The processor can be awakened from an interrupt if the WFI instruction of the Cortex-M4 is used to enter Sleep mode, or from a wake-up event if the WFE instruction is used. The WFI instruction can also be used to enter Sleep mode with the SLEEPONEXIT bit set to 1 in the System Control Register (SCB_SCR) of the Cortex-M. If the SLEEPONEXIT bit of the SCB_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. This mechanism can be used in applications that require the processor to run only when an exception occurs. Setting the SLEEPONEXIT bit to 1 enables an interrupt-driven application in order to avoid returning to an empty main application.

5.5.4 Low-power Mode Summary Table

The modes detailed above are the main low-power modes. Table 5-2 below provides a configuration summary of the low-power modes. For more information on power consumption, refer to the section “Electrical Characteristics”.

Table 5-2. Low-power Mode Configuration Summary

Mode	SUPC, 32 kHz Oscillator, RTC, RTT Backup Registers, POR (Backup Region)	Core Regulator / LCD Regulator	Core 0/1 Memory Peripherals	Mode Entry(1)	Potential Wake-up Sources	Core at Wake-up	PIO State in Low-power Mode	PIO State at Wake-up	Wake-up Time(2)
Backup Mode	ON	OFF/OFF	OFF / OFF (Not powered)	VROFF bit = 1 or SLEEPDEEP = 1 + WFE	- FWUP pin - WKUP0-15 pins(3) - Supply Monitor - Anti-tamper inputs(3) - RTC or RTT alarm	Reset	Previous state saved	Reset state(4)	< 1,5 ms
Backup Mode with LCD	ON	OFF/ON	OFF / OFF (Not powered)	VROFF bit = 1 or SLEEPDEEP = 1 + WFE	- FWUP pin - WKUP0-15 pins(3) - Supply Monitor - Anti-Tamper inputs(3) - RTC or RTT alarm	Reset	Previous state saved	Unchanged (LCD Pins) / Inputs with pull ups	< 1,5 ms
Wait Mode Flash in Standby Mode(5)	ON	ON/(6)	Core 0 and 1, memories and peripherals: Powered, but Not clocked	WAITMODE = 1 + FLPM = 0 or SLEEPDEEP = 0 + LPM = 1 + FLPM = 0 + WFE	Any Event from: - Fast start-up through WKUP0-15 pins - Anti-Tamper inputs(3) - RTC or RTT alarm	Clocked back	Previous state saved	Unchanged	< 10 µs
Wait Mode Flash in Deep Power - down Mode(5)	ON	ON/(6)	Core 0 and 1, memories and peripherals: Powered, but Not clocked	WAITMODE = 1 + FLPM = 1 or SLEEPDEEP = 0 + LPM = 1 + FLPM = 1 + WFE	Any Event from: - Fast start-up through WKUP0-15 pins - Anti-Tamper inputs(3) - RTC or RTT alarm	Clocked back	Previous state saved	Unchanged	< 75 µs
Sleep Mode	ON	ON/(6)	Core 0 and/or Core 1: Powered (Not clocked)(7)	SLEEPDEEP = 0 + LPM = 0 + WFE or WFI	Entry mode = WFI Any enabled Interrupts; Entry mode = WFE Any enabled events: - Fast start-up through WKUP0-15 pins - Anti-Tamper inputs(3) - RTC or RTT alarm	Clocked back	Previous state saved	Unchanged	(8)

Notes:
1. Refer to the note in Section 5.5 “Low-power Modes”.
2. When considering wake-up time, the time required to start the PLL is not taken into account. Once started, the device works either from the 4, 8 or 12 MHz fast RC oscillator. The user has to add the PLL start-up time if it is needed in the system. The wake-up time is defined as the time taken for wake-up until the first instruction is fetched.
3. Refer to Table 3-1, “Signal Description List”. Some anti-tamper pin pads are VDDIO powered.
4. See PIO Controller Multiplexing tables in Section 11.4 “Peripheral Signal Multiplexing on I/O Lines”.
5. Fast RC Oscillator set to 4 MHz Frequency.
6. LCD voltage regulator can be OFF if VDDLCD is supplied externally thus saving current consumption of the LCD voltage regulator.
7. In this mode, the core is supplied and not clocked but some peripherals can be clocked.
8. Depends on MCK frequency.

5.6 Wake-up Sources

Wake-up events allow the device to exit Backup mode. When a wake-up event is detected, the Supply Controller performs a sequence which automatically re-enables the core power supply and all digital logic.

5.7 Fast Start-up

The SAM4CP16C allows the processor to restart in a few microseconds while the processor is in Wait mode or in Sleep mode. A fast start-up occurs upon detection of one of the wake-up inputs.

The fast restart 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 and re-enables the processor clock.

6. Input/Output Lines

The SAM4CP16C has two types of input/output (I/O) lines: general-purpose I/Os (GPIO) and system I/Os. GPIOs 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.

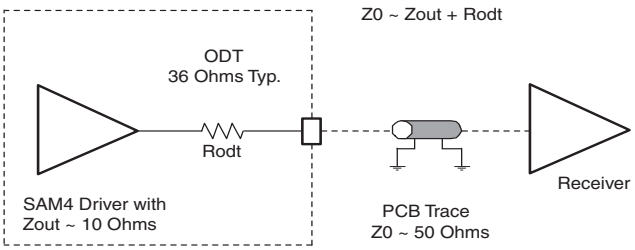
6.1 General-Purpose I/O Lines

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

The input/output buffers of the PIO lines are supplied through VDDIO power supply rail when used as GPIOs. When used as extra functions such as LCD or Analog modes, GPIO lines have either VDDLCD or VDDIN voltage range.

Each I/O line embeds an ODT (On-die Termination), shown in Figure 6-1 below. ODT consists of an internal series resistor termination scheme for impedance matching between the driver output (SAM4CP16C) and the PCB trace impedance preventing signal reflection. The series resistor helps to reduce IOs switching current (di/dt) thereby reducing EMI. It also decreases overshoot and undershoot (ringing) due to inductance of interconnect between devices or between boards. Finally, ODT helps diminish signal integrity issues.

Figure 6-1. On-die Termination



6.2 System I/O Lines

System I/O lines are pins used by oscillators, test mode, reset, JTAG and other features. Table 6-1 describes the SAM4CP16C system I/O lines shared with PIO lines.

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

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

SYSTEM_IO Bit Number	Default Function after Reset	Other Function	Constraints for Normal Start	Configuration
0	TDI	PB0	-	In Matrix User Interface Registers (Refer to the System I/O Configuration Register in the “Bus Matrix” section of this datasheet)
1	TDO/TRACESWO	PB1	-	
2	TMS/SWDIO	PB2	-	
3	TCK/SWCLK	PB3	-	
4	ERASE	PC9	Low level at Start-up(1)	
-	PA31	XIN	-	(2)
-	PA30	XOUT	-	

Notes: 1. If PC9 is used as PIO input in user applications, a low level must be ensured at start-up to prevent Flash erase before the user application sets PC9 into PIO mode.

2. Refer to “3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator” in the section “Clock Generator”

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

The SWJ-DP pins are TCK/SWCLK, TMS/SWDIO, TDO/TRACESWO, TDI and commonly provided on a standard 20-pin JTAG connector defined by ARM. For more details about voltage reference and reset state, refer to Table 11-6, “Multiplexing on PIO Controller B (PIOB)”.

At start-up, SWJ-DP pins are configured in SWJ-DP mode to allow connection with debugging probe. Refer to the section “Debug and Test” of this datasheet.

SWJ-DP pins can be used as standard I/Os to provide users with more general input/output pins when the debug port is not needed in the end application. Mode selection between SWJ-DP mode (System IO mode) and general IO mode is performed through the AHB Matrix Special Function Registers (MATRIX_SFR). Configuration of the pad for pull-up, triggers, debouncing and glitch filters is possible regardless of the mode.

The JTAGSEL pin is used to select the JTAG boundary scan when asserted at a high level. It integrates a permanent pull-down resistor of about 15 kΩ to GND, so that it can be left unconnected for normal operations.

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

The asynchronous TRACE output (TRACESWO) is multiplexed with TDO. So the asynchronous trace can only be used with SW-DP, not JTAG-DP. For more information about SW-DP and JTAG-DP switching, refer to the “Debug and Test” section of this datasheet. The SW-DP/SWJ-DP pins are used for debug access to both cores.

6.3 TST Pin

The TST pin is used for JTAG Boundary Scan Manufacturing Test or Fast Flash programming mode of the SAM4CP16C series. For details on entering Fast Programming mode, see the “Fast Flash Programming Interface (FFPI)” section of this datasheet. For more information on the manufacturing and test modes, refer to the section “Debug and Test” of this datasheet.

6.4 NRST Pin

The NRST pin is bidirectional. It is handled by the on-chip reset controller and can be driven low to provide a reset signal to the external components, or asserted low externally to reset the microcontroller. It resets the core and the peripherals, with the exception of the GPLC peripheral and 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.

6.5 TMPx Pins: Anti-tamper Pins

Anti-tamper pins detect intrusion, for example, into a smart meter case. Upon detection through a tamper switch, automatic, asynchronous and immediate clear of registers in the backup area, and time stamping in the RTC are performed. Anti-tamper pins can be used in all modes. Date and number of tampering events are stored automatically. Anti-tampering events can be programmed so that half of the General-purpose Backup Registers (GPBR) are erased automatically. TMP1 to TMP3 signals are shared with a PIO pin which requires that VDDIO is supplied, whereas TMP0 is in the VDDBU domain.

6.6 RTCOUT0 Pin

The RTCOUT0 pin shared in the PIO (supplied by VDDIO) can be used to generate waveforms from the RTC in order to take advantage of the RTC inherent prescalers while the RTC is the only powered circuitry (Low-power mode, Backup mode) or in any Active mode. Entering Backup or low-power operating modes does not affect the waveform generation outputs (VDDIO still must be supplied). Anti-tampering pin detection can be synchronized with this signal.

Note: To use the RTCOUT0 signal during application development using JTAG-ICE interface, the programmer must use Serial Wire Debug (SWD) mode. In this case, the TDO pin is not used as a JTAG signal by the ICE interface.

6.7 Shutdown (SHDN) Pin

The SHDN pin designates the Backup mode of operation. When the device is in Backup mode, SHDN = 0. In any other mode, SHDN = 1 (VDDBU). This pin is designed to control the enable pin of the main external voltage regulator. When the device enters Backup mode, the SHDN pin disables the external voltage regulator and, upon the wake-up event, it re-enables the voltage regulator.

The SHDN pin is asserted low when the VROFF bit in the Supply Controller Control Register (SUPC_CR) is set to 1.

6.8 Force Wake-up (FWUP) Pin

The FWUP pin can be used as a wake-up source in all low-power modes as it is supplied by VDDBU.

6.9 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. The ERASE pin integrates a pull-down resistor of about 100 kΩ into GND, so that it can be left unconnected for normal operations.

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

The ERASE pin is a system I/O pin and can be used as a standard I/O. At start-up, the ERASE pin is not configured as a PIO pin. If the ERASE pin is used as a standard I/O, the start-up level of this pin must be low to prevent unwanted erasing. Refer to Section 11.3 “APB/AHB Bridge”. If the ERASE pin is used as a standard I/O output, asserting the pin to low does not erase the Flash.

To avoid unexpected erase at power-up, a minimum ERASE pin assertion time is required. This time is defined in the AC Flash Characteristics in the section “Electrical 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.

With the following sequence, in any case, the erase operation is performed:

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. Product Mapping and Peripheral Access

Figure 7-1 shows the default memory mapping of the ARM Cortex-M core.

Figure 7-1. Cortex-M Memory Mapping

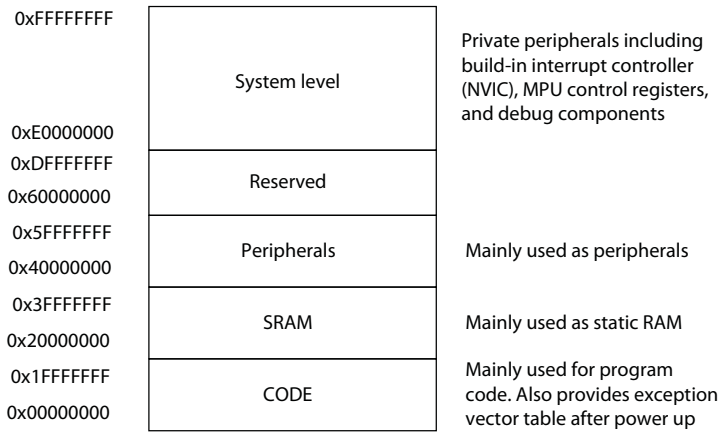
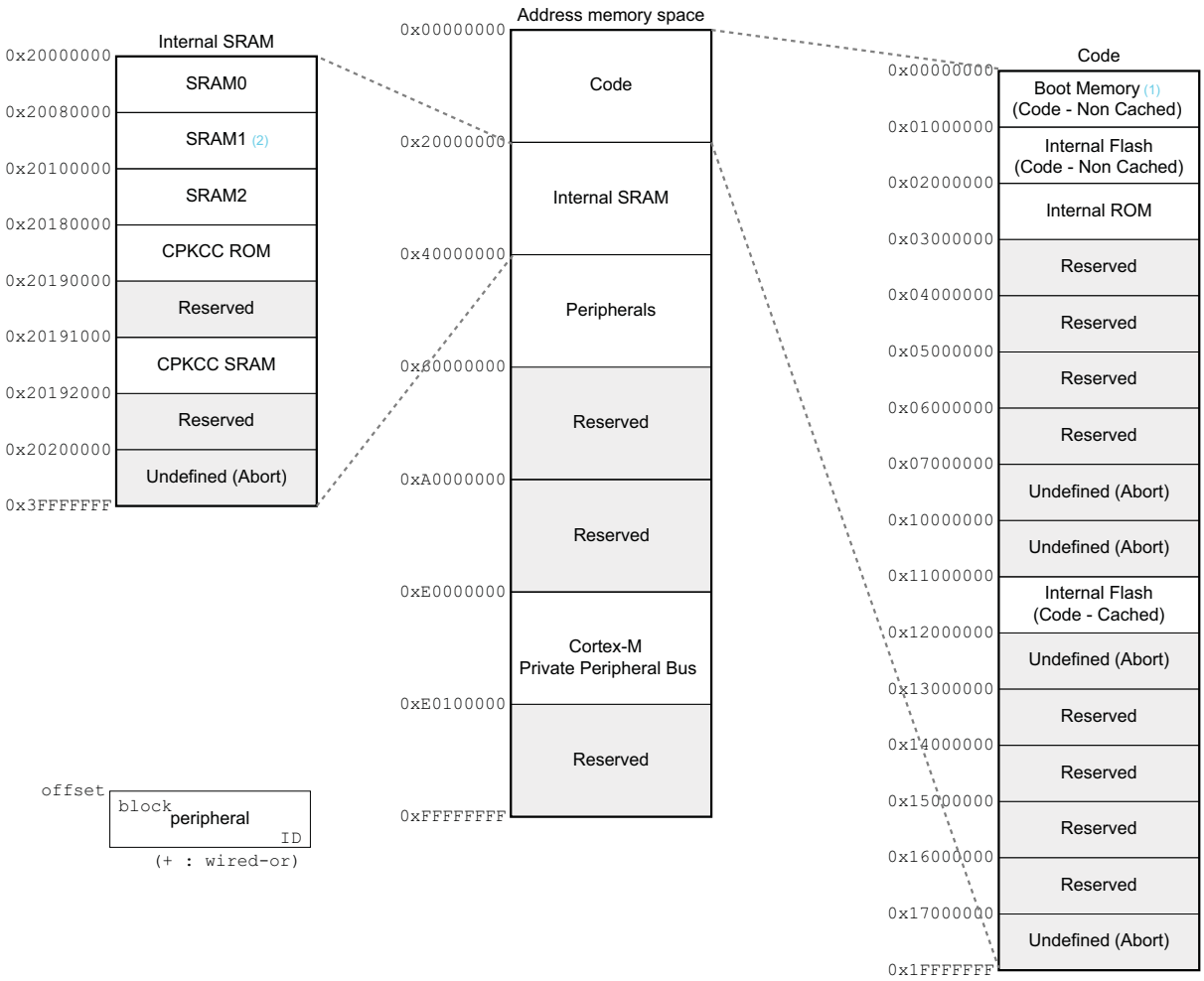
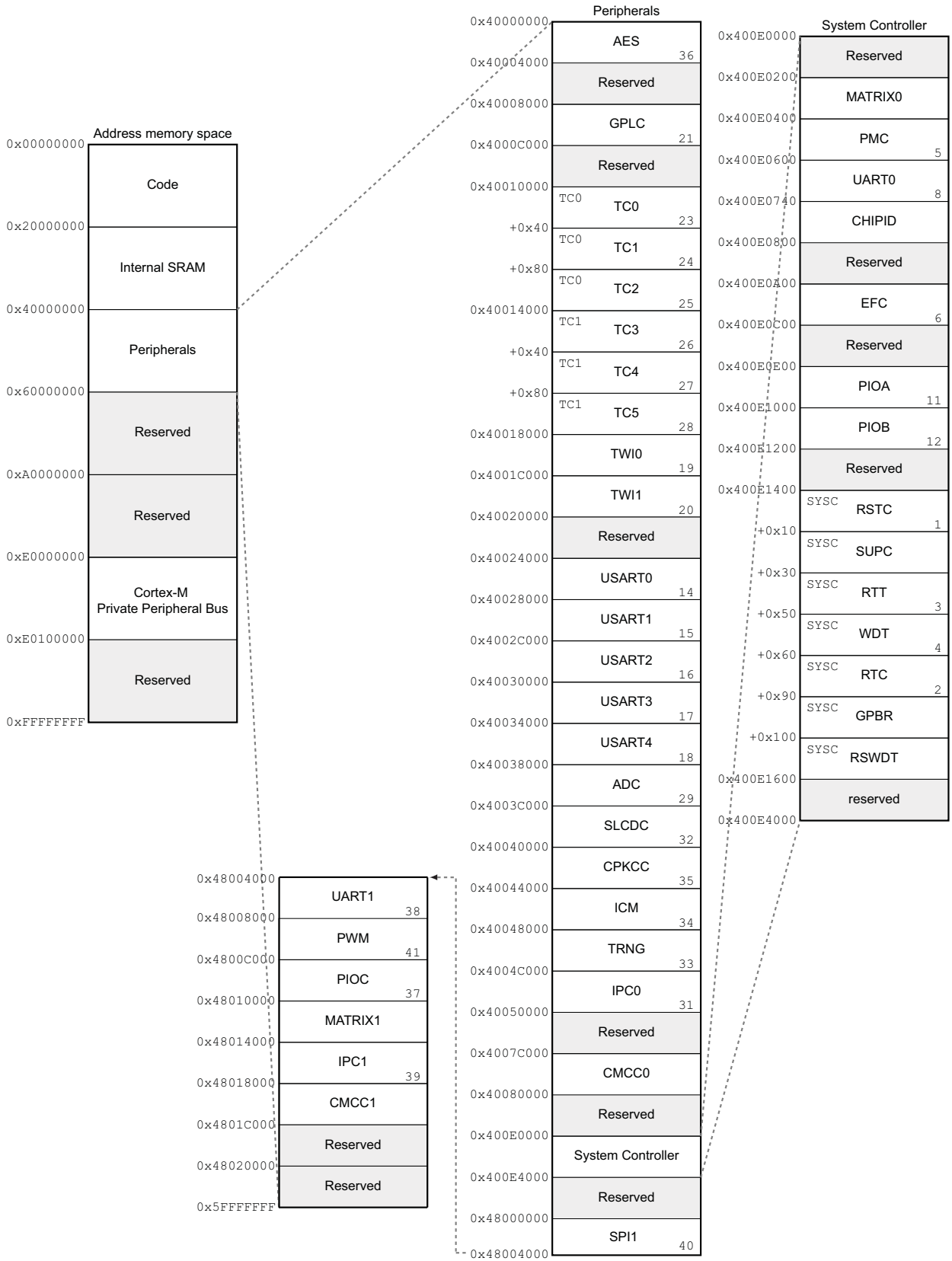


Figure 7-2. SAM4CP16C Memory Mapping of CODE and SRAM Area



Notes: 1. Boot Memory for Core 0.
2. Boot Memory for Core 1 at 0x00000000.

Figure 7-3. SAM4CP16C Memory Mapping of the Peripherals Area



In Figure 7-2, ‘Code’ means ‘Program Code over I-Code bus’ and ‘Program Data over D-Code bus’.

SRAM1 is at the address 0x20080000 (through S-bus) and the address 0x00000000 (through I/D Bus) for Core 1. Instruction fetch from Core 1 to the SRAM address range is possible but leads to reduced performance due to the fact that instructions and read/write data go through the System Bus (S-Bus). Maximum performance for Core 1 is obtained by mapping the instruction code to the address 0x00000000 (SRAM1 through I/D-Code) and read/write data from the address 0x20100000 (SRAM2 through S-Bus).

For Core 0 (Application Core), maximum performance is achieved when the instruction code is mapped to the Flash address and read/write data is mapped into SRAM0.

Each core can access the following memories and peripherals:

- Core 0 (Application Core):
  - All internal memories
  - All internal peripherals
- Core 1 (Coprocessor Core):
  - All internal memories
  - All internal peripherals

Note that Peripheral DMA 0 on Matrix 0 cannot access SRAM1 or SRAM2, Peripheral DMA 1 on Matrix 1 cannot access SRAM0, SRAM2 or SRAM0 can be the Data RAM for Inter-core Communication.

If Core 1 is not to be used (clock stopped and reset active), all the peripherals, SRAM1 and SRAM2 of the Sub-system 1 can be used by the Application Core (Core 0) as long as the peripheral bus clock and reset are configured.

Detailed memory mapping and memory access versus Matrix masters/slaves are given in the “Bus Matrix (MATRIX)” section of this datasheet.

8. Memories

The memory map shown in Figure 7-2, “SAM4CP16C Memory Mapping of CODE and SRAM Area” is common to both Cortex-M4 processors with the exception of the “Boot Memory” block. For more information on Boot Memory, refer to Section 8.1.5 “Boot Strategy” on page 37.

Each processor uses its own ARM Private Peripheral Bus (PPB) for the NVIC and other system functions.

8.1 Embedded Memories

8.1.1 Internal SRAM

The SAM4CP16C embeds a total of 152 Kbytes high-speed SRAM with zero wait state access time.

SRAM0 on Matrix0 is 128 Kbytes. It is dedicated to the application processor (CM4P0) or other peripherals on Matrix0 but can be identified and used by masters on Matrix1.

SRAM1 on Matrix1 is 16 Kbytes. It is mainly dedicated to be the code region of the CM4P1 processor but can be identified and used by Matrix0.

SRAM2 on Matrix1 is 8 Kbytes. It is mainly dedicated to be the data region of the CM4P1 processor or other peripherals on Matrix1 but can be identified and used by masters on Matrix0.

Refer to the section “Bus Matrix (MATRIX)” of this datasheet for more details.

If the CM4P1 processor is in the reset state and not used, the application core may use it.

The SRAM is located in the bit band region. The bit band alias region is from 0x2200 0000 to 0x23FF_FFFF.

8.1.2 System ROM

The SAM4CP16C embeds an Internal ROM for the master processor (CM4P0), which contains the SAM Boot Assistant (SAM-BA), In Application Programming routines (IAP), and Fast Flash Programming Interface (FFPI).

The ROM is always mapped at the address 0x02000000.

8.1.3 CPKCC ROM

The ROM contains a cryptographic library using the CPKCC cryptographic accelerator peripheral (CPKCC) to provide support for Rivest Shamir Adleman (RSA), Elliptic Curve Cryptography (ECC), Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA).

8.1.4 Embedded Flash

8.1.4.1 Flash Overview

The embedded Flash is the boot memory for the Cortex-M4 Core 0 (CM4P0).

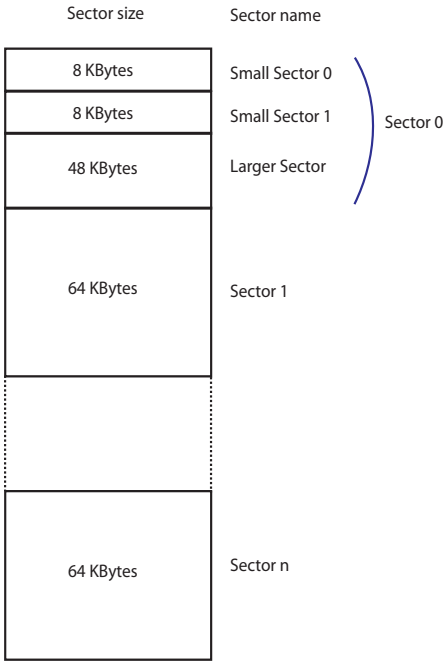
The flash memory can be accessed through the Cache Memory Controller (CMCC0) of the CM4P0 and can also be identified by the Cortex-M4F Core 1 (CM4P1) through its Cache Memory Controller (CMCC1).

The memory plane is organized in sectors. Each sector has a size of 64 Kbytes. The first sector of 64 Kbytes is divided into 3 smaller sectors.

The three smaller sectors are organized in 2 sectors of 8 Kbytes and 1 sector of 48 Kbytes. Refer to Figure 8-1 below.

The Flash memory has built-in error code correction providing 2-bit error detection and 1-bit correction per 128 bits.

Figure 8-1. Memory Plane Organization



Each sector is organized in pages of 512 bytes.

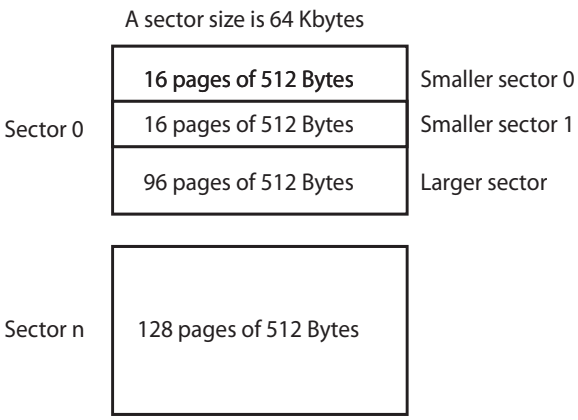
For sector 0:

- The small sector 0 has 16 pages of 512 bytes, 8 Kbytes in total.
- The small sector 1 has 16 pages of 512 bytes, 8 Kbytes in total.
- The larger sector has 96 pages of 512 bytes, 48 Kbytes in total.

From sector 1 to n:

The rest of the array is composed of 64-Kbyte sectors where each sector comprises 128 pages of 512 bytes. Refer to Figure 8-2, “Flash Sector Organization” below.

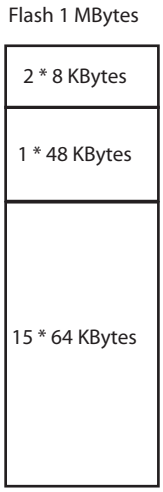
Figure 8-2. Flash Sector Organization



In SAM4CP16C the flash size is 1024 Kbytes.

Figure 8-3 illustrates the organization of the Flash depending on its size.

Figure 8-3. 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).

8.1.4.2 Enhanced Embedded Flash Controller

The Enhanced Embedded Flash Controller manages accesses performed by 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 the full set of commands.

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

8.1.4.3 Flash Speed

The user must set the number of wait states depending on the frequency used on the SAM4CP16C.

For more details, refer to “Embedded Flash” in the section “Electrical Characteristics”.

8.1.4.4 Lock Regions

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

Table 8-1. Lock bit number

Product	Number of Lock Bits	Lock Region Size
SAM4CP16C	128	8 Kbytes

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

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

8.1.4.5 Security Bit

The SAM4CP16C 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 SW-DP/JTAG-DP 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.

8.1.4.6 Unique Identifier

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

8.1.4.7 User Signature

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

8.1.4.8 Fast Flash Programming Interface

The Fast Flash Programming Interface allows programming the device through either a serial JTAG interface or 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.

8.1.4.9 SAM-BA Boot

The SAM-BA Boot is a default Boot Program for the master processor (CM4P0) which provides an easy way to program in-situ the on-chip Flash memory.

The SAM-BA Boot Assistant supports serial communication via the UART0.

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

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

8.1.4.10 GPNVM Bits

The SAM4CP16C 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 (see the “EEFC Flash Command Register” section of this datasheet).

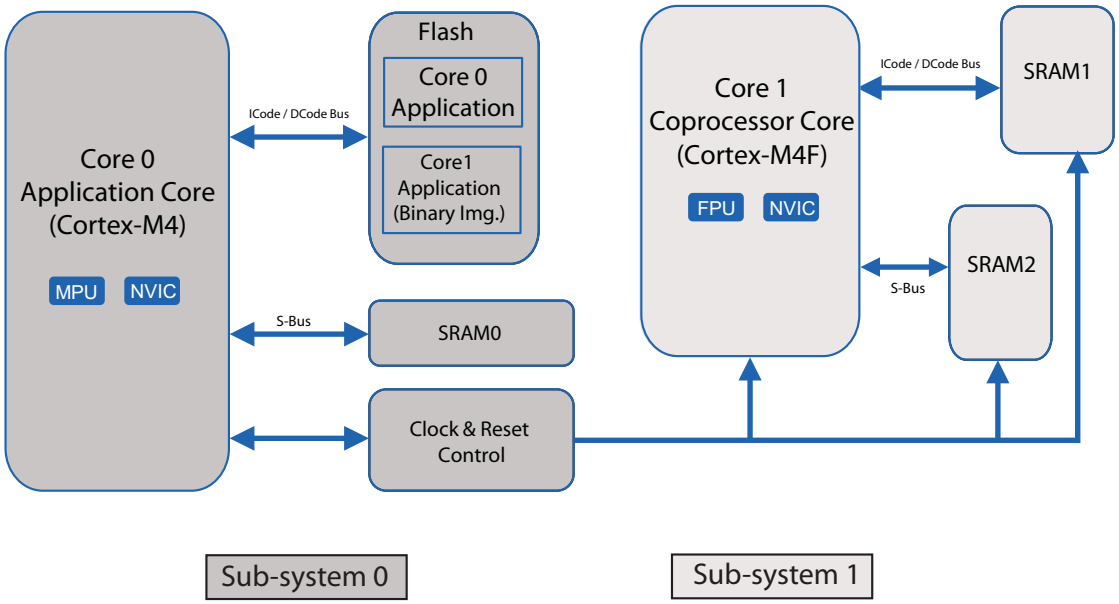
Table 8-2. General-purpose Nonvolatile Memory Bits

GPNVMBit	Function
0	Security bit
1	Boot mode selection

8.1.5 Boot Strategy

Figure 8-4 below shows a load view of the memory at boot time.

Figure 8-4. Simplified Load View at Boot Time



Note: Matrices, AHB and APB Bridges are not represented.

8.1.5.1 Application Core (Core 0) Boot Process

The application processor (CM4P0) always boots at the address 0x0. To ensure maximum boot possibilities, the memory layout can be changed using a General-purpose NVM (GPNVM) bit. A GPNVM bit is used to boot either on the ROM (default) or from the Flash. The GPNVM bit can be cleared or set 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 Flash, whereas clearing this bit selects the boot from ROM. Asserting ERASE clears the GPNVM Bit 1 and thus selects the boot from the ROM by default.

8.1.5.2 Coprocessor Core (Core 1) Boot Process

After reset, the Sub-system 1 is hold in reset and with no clock. It is up to the Master Application (Core 0 Application) running on the Core 0 to enable the Sub-system 1. Then the application code can be downloaded into the CM4P1 Boot memory (SRAM1), and CM4P0 can afterwards de-assert the CM4P1 reset line. The secondary processor (CM4P1) always identifies SRAM1 as “Boot memory”.

8.1.5.3 Sub-system 1 Startup Sequence

After the Core 0 is booted from Flash, the Core 0 application must perform the following steps:

1. Enable Core 1 System Clock (Bus and peripherals).
2. Enable Core 1 Clock.
3. Release Core 1 System Reset (Bus and peripherals).
4. Enable SRAM1 and SRAM2 Clock.
5. Copy Core 1 Application from Flash into SRAM1.
6. Release Core 1 Reset.

After Step 6, the Core 1 boots from SRAM1 memory.

Pseudo-code

```
1- // Enable Coprocessor Bus Master Clock (PMC_SCER.CPBMCK).

2- // Enable Coprocessor Clocks (PMC_SCER.CPCK).
      // Set Coprocessor Clock Prescaler and Source (PMC_MCKR.CPPRES).
      // Choose coprocessor main clock source (PMC_MCKR.CPCSS).

3- // Release coprocessor peripheral reset (RSTC_CPMR.CPEREN).

4- // Enable Core 1 SRAM1 and SRAM2 Memories (PMC_PCER.PID42).

5- // Copy Core 1 application code from Flash into SRAM1.

6- // Release coprocessor reset (RSTC_CPMR.CPROCEN).
```

8.1.5.4 Sub-system 1 Start-up Time

Table 8-3 provides the start-up time of sub-system 1 in terms of the number of clock cycles for different CPU speeds. The figures in this table take into account the time to copy 16 Kbytes of code from Flash into SRAM1 using the ‘memcopy’ function from the standard C library and to release Core 1 reset signal. The start-up time of the device from power-up is not taken into account.

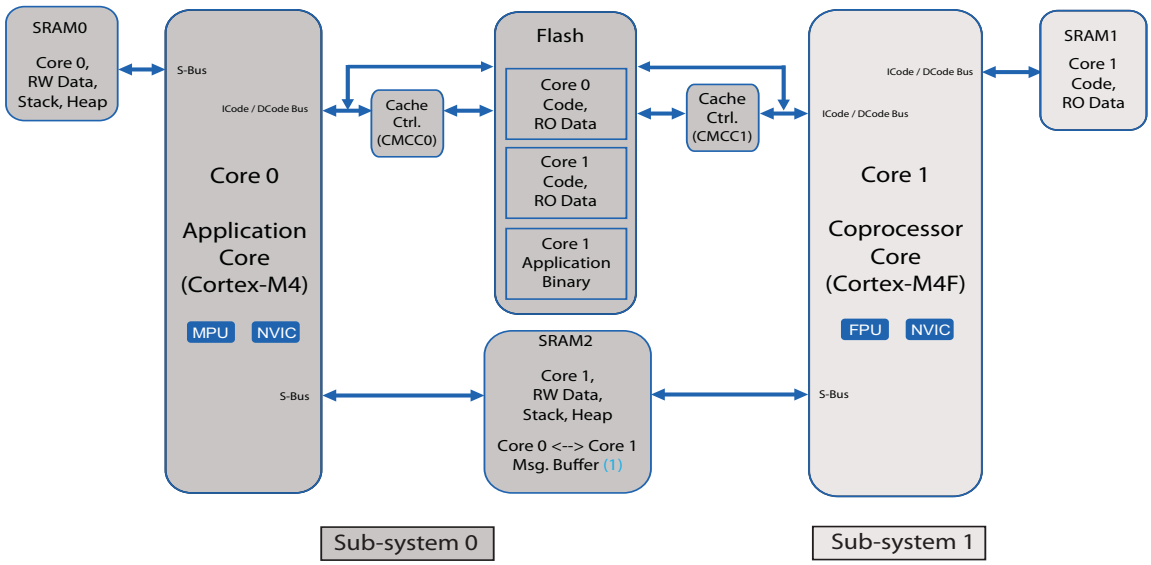
Table 8-3. Sub-system 1 Start-up Time

Core Clock (MHz)	Flash Wait State	Core Clock Cycles	Time
21	0	44122	2.1 ms
42	1	45158	1.07 ms
63	2	46203	735 µs
85	3	47242	55 µs
106	4	48284	455 µs
120	5	49329	411 µs

8.1.5.5 Typical Execution View

Figure 8-5 provides the code execution view for both Cortex-M4 cores. AHB to APB, AHB to AHB and Matrices are not represented in this view.

Figure 8-5. Execution View



Note: 1. SRAM0 can also be used as Message Buffer Exchange.
Note: Matrices, AHB and APB Bridges are not represented.

9. Real-time Event Management

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

9.1 Embedded Characteristics

- Timers generate event triggers which are directly routed to event managers, such as ADC, to start measurement/conversion without processor intervention.
- UART, USART, SPI, TWI, and PIO generate event triggers directly connected to Peripheral DMA controller (PDC) for data transfer without processor intervention.
- PMC Security Event (Clock Failure Detection) can be programmed to switch the MCK on reliable main RC internal clock.

9.2 Real-time Event Mapping List

Table 9-1. Real-time Event Mapping List

Function	Application	Description	Event Source	Event Destination
Safety	General-purpose	Automatic switch to reliable main RC oscillator in case of main crystal clock failure(1)	Power Management Controller (PMC)	PMC
Security	General-purpose	Immediate (asynchronous) clear of first half of GPBR on tamper detection through pins(2)	Anti-tamper Inputs (TMPx)	GPBR
Measurement trigger	General-purpose	Trigger source selection in ADC(3)	IO (ADTRG)	ADC
			TC Output 0	
			TC Output 1	
			TC Output 2	
			TC Output 3	
			TC Output 4	
			TC Output 5	

Notes:
1. Refer to Section 30.13 “Main Clock Failure Detector”.
2. Refer to Section 20.4.9.3 “Low-power Debouncer Inputs (Tamper Detection Pins)” and Section 21.3.1 “General Purpose Backup Register x”.
3. Refer to Section 40.7.2 “ADC Mode Register”.

10. System Controller

The System Controller comprises a set of peripherals. It handles key elements of the system, such as power, resets, clocks, time, interrupts, watchdog, reinforced safety watchdog, etc.

10.1 System Controller and Peripheral Mapping

Refer to “SAM4CP16C Memory Mapping of CODE and SRAM Area” .

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

10.2 Power Supply Monitoring

The SAM4CP16C embeds Supply Monitor, Power-on-Reset and Brownout detectors for power supplies monitoring allowing to warn and/or reset the chip.

10.2.1 Power-on-Reset on VDDCORE

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

10.2.2 Brownout Detector on VDDCORE

The Brownout Detector monitors VDDCORE. It is active by default. It can be deactivated by software through the Supply Controller (SUPC_MR).

If VDDCORE goes below the threshold voltage, the reset of the core is asserted.

10.2.3 Power-on-Reset on VDDIO

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 IOs are reset but the core continues to run. Voltage detection is fixed.

10.2.4 Supply Monitor on VDDIO

The supply monitor on VDDIO is fully programmable with 4 steps for the threshold (between 3.0V to 3.4V). It provides the user the flexibility to set a voltage level detection higher then the power-on-reset on VDDIO. Either a reset or an interrupt can be generated upon detection. It can be activated by software and it is controlled by the Supply Controller (SUPC). A sample mode is possible. It divides the supply monitor power consumption by a factor of up to 2048.

The supply monitor is used as “system alert” in case VDDIO supply is falling. It can be used while the device is in Backup mode to wake up the device if VDDIO is falling.

10.2.5 Power-on-Reset and Brownout Detector on VDDBU

The Power-on-Reset monitors VDDBU. It is active by default and monitors voltage at start-up but also during power-down. It can be deactivated by software through the Supply Controller (SUPC_MR). If VDDBU goes below the threshold voltage, the entire chip is reset.

10.3 Reset Controller

The Reset Controller uses the Power-on-Reset, supply monitor and brownout detector cells.

The Reset Controller returns the source of the last reset to the software. Refer to the description of the field RSTTYP in the section “Reset Controller (RSTC)”.

The Reset Controller controls the internal resets of the system (or independent reset of CM4P1 processor) and the NRST pin input/output. It shapes a reset signal for the external devices, simplifying to a minimum connection of a push-button on the NRST pin to implement a manual reset.

The configuration of the Reset Controller is saved during Backup mode as it is supplied by VDDBU.

10.4 Supply Controller (SUPC)

The Supply Controller controls the power supplies of each section of the processor.

The Supply Controller starts up the device by sequentially enabling the internal power switches and the Voltage Regulator, then it generates the proper reset signals to the core power supply.

It also sets the system in different low-power modes, wakes it up from a wide range of events.

11. Peripherals

11.1 Peripheral Identifiers

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

The two ARM Cortex-M4 processors share the same interrupt mapping, and thus, they share all the interrupts of the peripherals.

Note: Some peripherals are on the Bus Matrix 0/AHB to APB Bridge 0 and other peripherals are on the Bus Matrix 1/AHB to APB Bridge 1. If Core 0 needs to access a peripheral on the Bus Matrix 1/AHB to APB Bridge 1, the Core 0 application must enable the Core 1 System Clock (Bus and peripherals) and release Core 1 System Reset (Bus and peripherals). Peripherals on Sub-system 0 or Sub-system 1 are mentioned in the Instance description table that follows.

Table 11-1. Peripheral Identifiers

Instance ID	Instance Name	NVIC Interrupt	PMC Clock Control	Instance Description
0	SUPC	X	-	Supply Controller
1	RSTC	X	-	Reset Controller
2	RTC	X	-	Real-time Clock
3	RTT	X	-	Real-time Timer
4	WDT	X	-	Watchdog Timer
5	PMC	X	-	Power Management Controller
6	EFC	X	-	Enhanced Embedded Flash Controller 0
7	-	-	-	Reserved
8	UART0	X	X	UART 0 (Sub-system 0 Clock)
9	-	-	-	Reserved
10	-	-	-	Reserved
11	PIOA	X	X	Parallel I/O Controller A (Sub-system 0 Clock)
12	PIOB	X	X	Parallel I/O Controller B (Sub-system 0 Clock)
13	-	-	-	Reserved
14	USART0	X	X	USART 0 (Sub-system 0 Clock)
15	USART1	X	X	USART 1 (Sub-system 0 Clock)
16	USART2	X	X	USART 2 (Sub-system 0 Clock)
17	USART3	X	X	USART 3 (Sub-system 0 Clock)
18	USART4	X	X	USART 4 (Sub-system 0 Clock)
19	TWI0	X	X	Two Wire Interface 0 (Sub-system 0 Clock)
20	TWI1	X	X	Two Wire Interface 1 (Sub-system 0 Clock)
21	GPLC	X	X	Power Line Communication (Sub-system 0 Clock)
22	-	-	-	Reserved
23	TC0	X	X	Timer/Counter 0 (Sub-system 0 Clock)
24	TC1	X	X	Timer/Counter 1 (Sub-system 0 Clock)
25	TC2	X	X	Timer/Counter 2 (Sub-system 0 Clock)

Table 11-1. Peripheral Identifiers (Continued)

Instance ID	Instance Name	NVIC Interrupt	PMC Clock Control	Instance Description
26	TC3	X	X	Timer/Counter 3 (Sub-system 0 Clock)
27	TC4	X	X	Timer/Counter 4 (Sub-system 0 Clock)
28	TC5	X	X	Timer/Counter 5 (Sub-system 0 Clock)
29	ADC	X	X	Analog To Digital Converter (Sub-system 0 Clock)
30	ARM	X	-	FPU signals (only on CM4P1 core): FPIXC, FPOFC, FPUFC, FPIOC, FPDZC, FPIDC, FPIXC
31	IPC0	X	X	Interprocessor communication 0 (Sub-system 0 Clock)
32	SLCDC	X	X	Segment LCD Controller (Sub-system 0 Clock)
33	TRNG	X	X	True Random Generator (Sub-system 0 Clock)
34	ICM	X	X	Integrity Check Module (Sub-system 0 Clock)
35	CPKCC	X	X	Classical Public Key Cryptography Controller (Sub-system 0 Clock)
36	AES	X	X	Advanced Enhanced Standard (Sub-system 0 Clock)
37	PIOC	X	X	Parallel I/O Controller C (Sub-system 1 Clock)
38	UART1	X	X	UART 1 (Sub-system 1 Clock)
39	IPC1	X	X	Interprocessor communication 1 (Sub-system 1 Clock)
40	SPI1	X	X	Serial Peripheral Interface 1 (Sub-system 1 Clock)
41	PWM	X	X	Pulse Width Modulation (Sub-system 1 Clock)
42	SRAM	-	X	SRAM1 (I/D Code bus of CM4P1), SRAM2 (System bus of CM4P1) (Sub-system 1 Clock)
43	-	-	-	Reserved

11.2 Peripheral DMA Controller (PDC)

Two Peripheral DMA Controllers (PDC) are available:

- PDC0: dedicated to peripherals on APB0
- PDC1: dedicated to peripherals on APB1

Features of the PDC include:

- Data transfer handling between peripherals and memories
- Low bus arbitration overhead
  - One master clock cycle needed for a transfer from memory to peripheral
  - Two master clock cycles needed for a transfer from peripheral to memory
- Next Pointer management to reduce interrupt latency requirement

Note that Peripheral DMA 0 on Matrix 0 cannot access SRAM1 or SRAM2. Peripheral DMA 1 on Matrix 1 cannot access SRAM0.

The PDC handles transfer requests from the channel according to the following priorities (Low to High priorities):

Table 11-2. Peripheral DMA Controller (PDC0)

Instance name	Channel T/R
AES	Transmit
TWI0	Transmit
UART0	Transmit
USART1	Transmit
USART0	Transmit
USART2	Transmit
USART3	Transmit
USART4	Transmit
GPLC	Transmit
AES	Receive
TWI0	Receive
UART0	Receive
USART4	Receive
USART3	Receive
USART2	Receive
USART1	Receive
USART0	Receive
ADC	Receive
GPLC	Receive

Table 11-3. Peripheral DMA Controller (PDC1)

Instance name	Channel T/R
UART1	Transmit
SPI1	Transmit
UART1	Receive
SPI1	Receive

11.3 APB/AHB Bridge

The SAM4CP16C embeds two peripheral bridges: one on each Matrix, with Matrix 0 for CM4P0 and Matrix 1 for CM4P1.

The peripherals of the bridge corresponding to CM4P0 (APB0) are clocked by MCK, and the peripherals of the bridge corresponding to CM4P1 (APB1) are clocked by CPBMCK.

11.4 Peripheral Signal Multiplexing on I/O Lines

The SAM4CP16C can multiplex the I/O lines of the peripheral set.

The SAM4CP16C PIO Controllers control up to 32 lines. Each line can be assigned to one of two peripheral functions: A or B. The multiplexing tables that follow 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 may be duplicated within the tables.

11.4.1 Pad Features

In Table 11-5 to Table 11-7, the column “Feature” indicates whether the corresponding I/O line has programmable Pull-up, Pull-down and/or Schmitt Trigger. Table 11-4 provides the key to the abbreviations used.

Table 11-4. I/O Line Features Abbreviations

Abbreviation	Definition
PUP (P)	Programmable Pull-up
PUP (NP)	Non-programmable Pull-up
PDN (P)	Programmable Pull-down
PDN (NP)	Non-programmable Pull-down
ST (P)	Programmable Schmitt Trigger
ST (NP)	Non-programmable Schmitt Trigger
LDRV (P)	Programmable Low Drive
LDRV (NP)	Non-programmable Low Drive
HDRV (P)	Programmable High Drive
HDRV (NP)	Non-programmable High Drive
MaxDRV (NP)	Non-programmable Maximum Drive

11.4.2 Reset State

In Table 11-5 to Table 11-7, the column “Reset State” indicates the reset state of the line.

- PIO or signal name: Indicates whether the PIO line resets in I/O mode or in peripheral mode.
  If “PIO” is mentioned, the PIO line is in general-purpose I/O (GPIO). If a signal name is mentioned in the “Reset State” column, the PIO line is assigned to this function.
- I or O: Indicates whether the signal is input or output state.
- PU or PD: Indicates whether Pull-up, Pull-down or nothing is enabled.
- ST: Indicates that Schmitt Trigger is enabled.

11.4.3 PIO Controller A Multiplexing

Table 11-5. Multiplexing on PIO Controller A (PIOA)

I/O Line	Peripheral A	Peripheral B	Peripheral C	Extra Function	System Function	Feature	Reset State	Comments
PA0	RTS3	PCK2	–	COM0	WKUP5	- PUP(P) / PDN(P) - ST(P) - MaxDRV(NP)	PIO, I, PU	
PA1	CTS3	–	–	COM1	–	- PUP(P) / PDN(P) - ST(P) - LDRV(P) / HDRV(P)		
PA2	SCK3	–	–	COM2	–			
PA3	RXD3	–	–	COM3	WKUP6			
PA4	TXD3	–	–	COM4/AD1	–			
PA9	RXD2	–	–	SEG3	WKUP2	- PUP(P) / PDN(P) - ST(P) - LDRV(P) / HDRV(P)		
PA10	TXD2	–	–	SEG4	–			
PA11	RXD1	–	–	SEG5	WKUP9			
PA12	TXD1	–	–	SEG6/AD0	–			
PA13	SCK2	TIOA0	–	SEG7	–			
PA14	RTS2	TIOB0	–	SEG8	WKUP3			
PA15	CTS2	TIOA4	–	SEG9	–			
PA16	SCK1	TIOB4	–	SEG10	–			
PA17	RTS1	TCLK4	–	SEG11	WKUP7			
PA18	CTS1	TIOA5	–	SEG12	–			
PA19	RTS0	TCLK5	–	SEG13	WKUP4			
PA20	CTS0	TIOB5	–	SEG14	–			
PA21	–	–	–	SEG15	–			
PA22	–	–	–	SEG16	–			
PA23	–	–	–	SEG17	–			
PA24	TWD0	–	–	SEG18	WKUP1			
PA25	TWCK0	–	–	SEG19	–			
PA26	CTS4	–	–	SEG20	–			
PA27	–	–	–	SEG21	–			
PA28	–	–	–	SEG22	–			
PA29	PCK1	–	–	SEG23	–	- PUP(P) / PDN(P) - ST(P) - MaxDRV(NP)		
PA30	PCK1	–	–	–	XOUT	- PUP(P) / PDN(P) - ST(P) - LDRV(P) / HDRV(P)	XOUT	
PA31	PCK0	–	–	–	XIN	- PUP(P) / PDN(P) - ST(P) - LDRV(P) / HDRV(P)	XIN	

11.4.4 PIO Controller B Multiplexing

Table 11-6. Multiplexing on PIO Controller B (PIOB)

I/O Line	Peripheral A	Peripheral B	Peripheral C	Extra Function	System Function	Feature	Reset State	Comments
PB0	TWD1	–	–	–	TDI	- PUP(P) / PDN(P) - ST(P) - LDRV(P) / HDRV(P)	JTAG, I	
PB1	TWCK1	–	–	RTCOUT0	TDO/ TRACESWO	- PUP(P) / PDN(P) - LDRV(NP)	JTAG, O	
PB2	–	–	–	–	TMS/SWDIO	- PUP(P) / PDN(P) - ST(P) - LDRV(P) / HDRV(P)	JTAG, I	
PB3	–	–	–	–	TCK/SWCLK			
PB4	URXD0	TCLK0	–	–	WKUP8		PIO, I, PU	
PB5	UTXD0	–	–	–	–			
PB6	–	–	–	SEG24	–			
PB7	TIOA1	–	–	SEG25	–			
PB8	TIOB1	–	–	SEG26	–			
PB9	TCLK1	–	–	SEG27	–			
PB10	TIOA2	–	–	SEG28	–			
PB11	TIOB2	–	–	SEG29	–			
PB12	TCLK2	–	–	SEG30	–			
PB13	PCK0	–	–	SEG31/AD3	–	- PUP(P) / PDN(P) - ST(P) - MaxDRV(NP)		
PB14	–	–	–	SEG32	–	- PUP(P) / PDN(P) - ST(P) - LDRV(P) / HDRV(P)		
PB15	–	–	–	SEG33	–			
PB16	RXD0	–	–	SEG34	WKUP10/ TMP1		PIO, I, PD	
PB17	TXD0	–	–	SEG35	–			
PB18	SCK0	PCK2	–	SEG36	–			
PB19	RXD4	–	–	SEG37	–			
PB20	TXD4	–	–	SEG38	–			
PB21	SCK4	–	–	SEG39	WKUP11			
PB22	RTS4	–	–	SEG40	–		PIO, I, PU	
PB23	ADTRG	–	–	SEG41/AD4	–			
PB24	TIOA3	–	–	SEG42	–			
PB25	TIOB3	–	–	SEG43	–			
PB26	TCLK3	–	–	SEG44	WKUP13			
PB27	–	–	–	SEG45	WKUP14/ TMP2			
PB28	–	–	–	SEG46	WKUP15/ TMP3			
PB29	–	–	–	SEG47	–			
PB31	–	–	–	SEG49/AD5	–			

11.4.5 PIO Controller C Multiplexing

Table 11-7. Multiplexing on PIO Controller C (PIOC)

I/O Line	Peripheral A	Peripheral B	Peripheral C	Extra Function	System Function	Feature	Reset State	Comments
PC0	UTXD1	PWM0	–	–	–	- PUP(P) - MaxDRV(NP)	PIO, I, PU	
PC1	URXD1	PWM1	–	–	WKUP12	- PUP(P) / PDN(P) - ST(P) - LDRV(P) / HDRV(P)		
PC2	SPI1_NPCS0	PWM2	–	–	–			
PC3	SPI1_MISO	PWM3	–	–	–			
PC4	SPI1_MOSI	–	–	–	–			
PC5	SPI1_SPCK	–	–	–	–	- PUP(P) - MaxDRV(NP)		
PC6	PWM0	SPI1_NPCS1	–	–	–	- PUP(P) / PDN(P) - ST(P) - LDRV(P) / HDRV(P)		
PC7	PWM1	SPI1_NPCS2	–	–	–			
PC8	PWM2	SPI1_NPCS3	–	–	–			
PC9	PWM3	–	–	–	ERASE		ERASE, PD	

12. ARM Cortex-M4 Processor

12.1 Description

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

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

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

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

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

12.1.1 System Level Interface

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

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

12.1.2 Integrated Configurable Debug

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

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

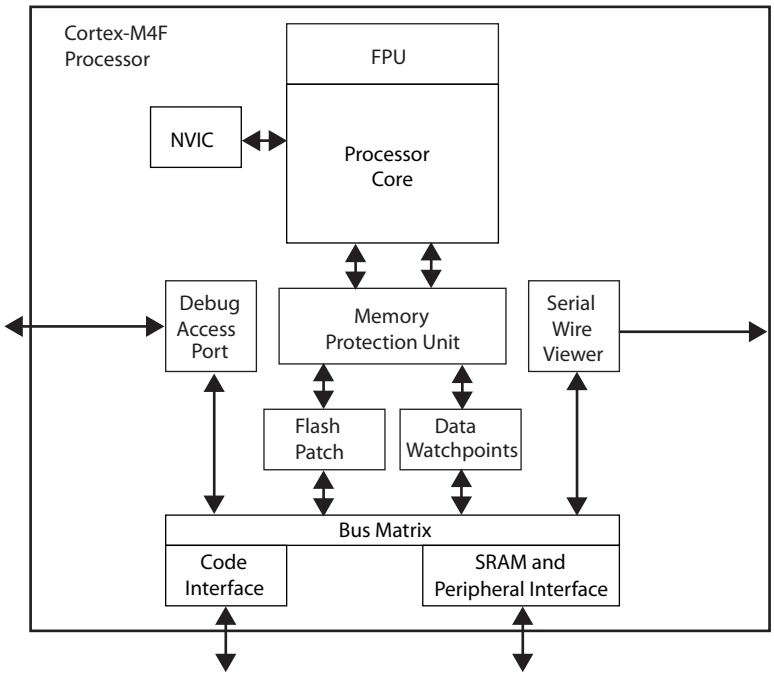
The Flash Patch and Breakpoint Unit (FPB) provides up to 8 hardware breakpoint comparators that debuggers can use. The comparators in the FPB also provide remap functions of up to 8 words in the program code in the CODE memory region. This enables applications stored on a non-erasable, ROM-based microcontroller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

12.2 Embedded Characteristics

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

12.3 Block Diagram

Figure 12-1. Typical Cortex-M4F Implementation



12.4 Cortex-M4 Models

12.4.1 Programmers Model

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

12.4.1.1 Processor Modes and Privilege Levels for Software Execution

The processor *modes* are:

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

The *privilege levels* for software execution are:

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

Unprivileged software executes at the unprivileged level.

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

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

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

12.4.1.2 Stacks

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

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

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

The options for processor operations are:

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

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

Note: 1. See “Control Register”.

12.4.1.3 Core Registers

Figure 12-2. Processor Core Registers

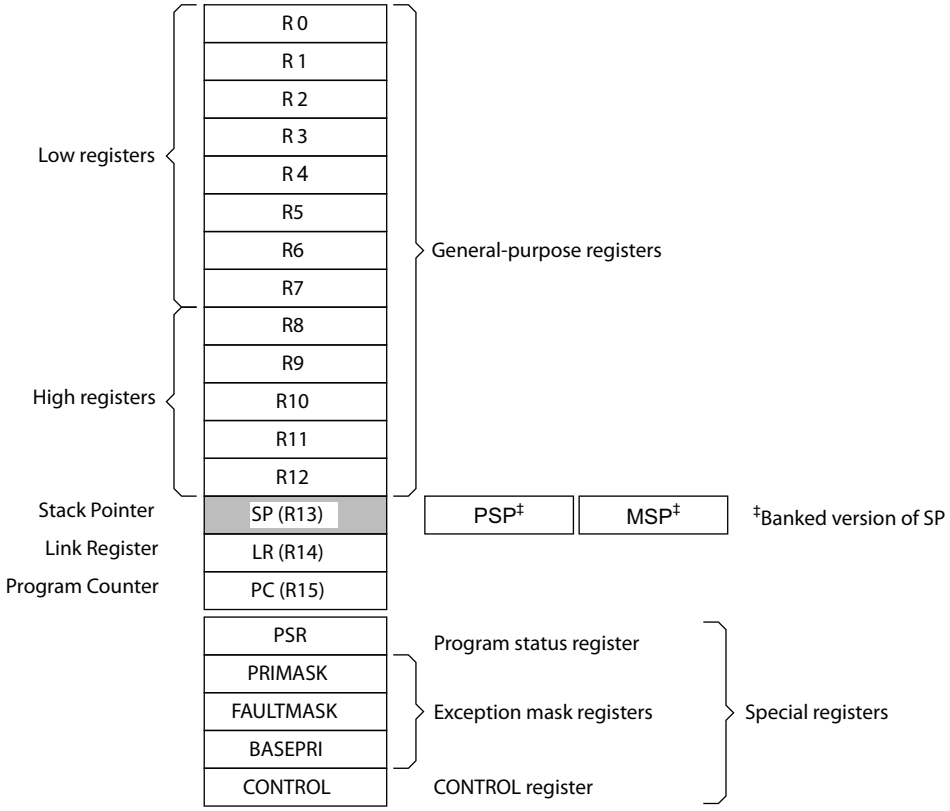


Table 12-2. Core Processor Registers

Register	Name	Access[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.

12.4.1.4 General-purpose Registers

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

12.4.1.5 Stack Pointer

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

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

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

12.4.1.6 Link Register

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

12.4.1.7 Program Counter

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

12.4.1.8 Program Status Register

Name: PSR

Access: Read/Write

Reset: 0x00000000

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.

12.4.1.9 Application Program Status Register

Name: APSR

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
N	Z	C	V	Q	–		

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

15	14	13	12	11	10	9	8
–							

7	6	5	4	3	2	1	0
–							

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

- **N: Negative Flag**

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

1: Operation result was negative or less than.

- **Z: Zero Flag**

0: Operation result was not zero.

1: Operation result was zero.

- **C: Carry or Borrow Flag**

Carry or borrow flag:

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

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

- **V: Overflow Flag**

0: Operation did not result in an overflow.

1: Operation resulted in an overflow.

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

Sticky saturation flag:

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

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

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

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

See “SEL” for more information.

12.4.1.10 Interrupt Program Status Register

Name: IPSR

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–							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

56 = IRQ40

See “Exception Types” for more information.

12.4.1.11 Execution Program Status Register

Name: EPSR

Access: Read/Write

Reset: 0x00000000

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

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
ICI/IT						–	

7	6	5	4	3	2	1	0
–							

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

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

- **ICI: Interruptible-continuable Instruction**

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

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

After servicing the interrupt, the processor:

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

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

- **IT: If-Then Instruction**

Indicates the execution state bits of the IT instruction.

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

- **T: Thumb State**

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

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

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

12.4.1.12 Exception Mask Registers

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

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

12.4.1.13 Priority Mask Register

Name: PRIMASK

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–							

7	6	5	4	3	2	1	0
–							PRIMASK

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

- **PRIMASK**

0: No effect.

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

12.4.1.14 Fault Mask Register

Name: FAULTMASK

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–							

7	6	5	4	3	2	1	0
–							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.

12.4.1.15 Base Priority Mask Register

Name: BASEPRI

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–							

7	6	5	4	3	2	1	0
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.

12.4.1.16 Control Register

Name: CONTROL

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–							

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

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

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

Indicates whether the floating-point context is currently active:

0: No floating-point context active.

1: Floating-point context active.

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

- **SPSEL: Active Stack Pointer**

Defines the current stack:

0: MSP is the current stack pointer.

1: PSP is the current stack pointer.

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

- **nPRIV: Thread Mode Privilege Level**

Defines the Thread mode privilege level:

0: Privileged.

1: Unprivileged.

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

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

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

- Use the MSR instruction to set the Active stack pointer bit to 1, see "MSR", or
- Perform an exception return to Thread mode with the appropriate EXC_RETURN value, see Table 12-10.

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

12.4.1.17 Exceptions and Interrupts

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

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

12.4.1.18 Data Types

The processor supports the following data types:

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

12.4.1.19 Cortex Microcontroller Software Interface Standard (CMSIS)

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

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

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

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

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

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

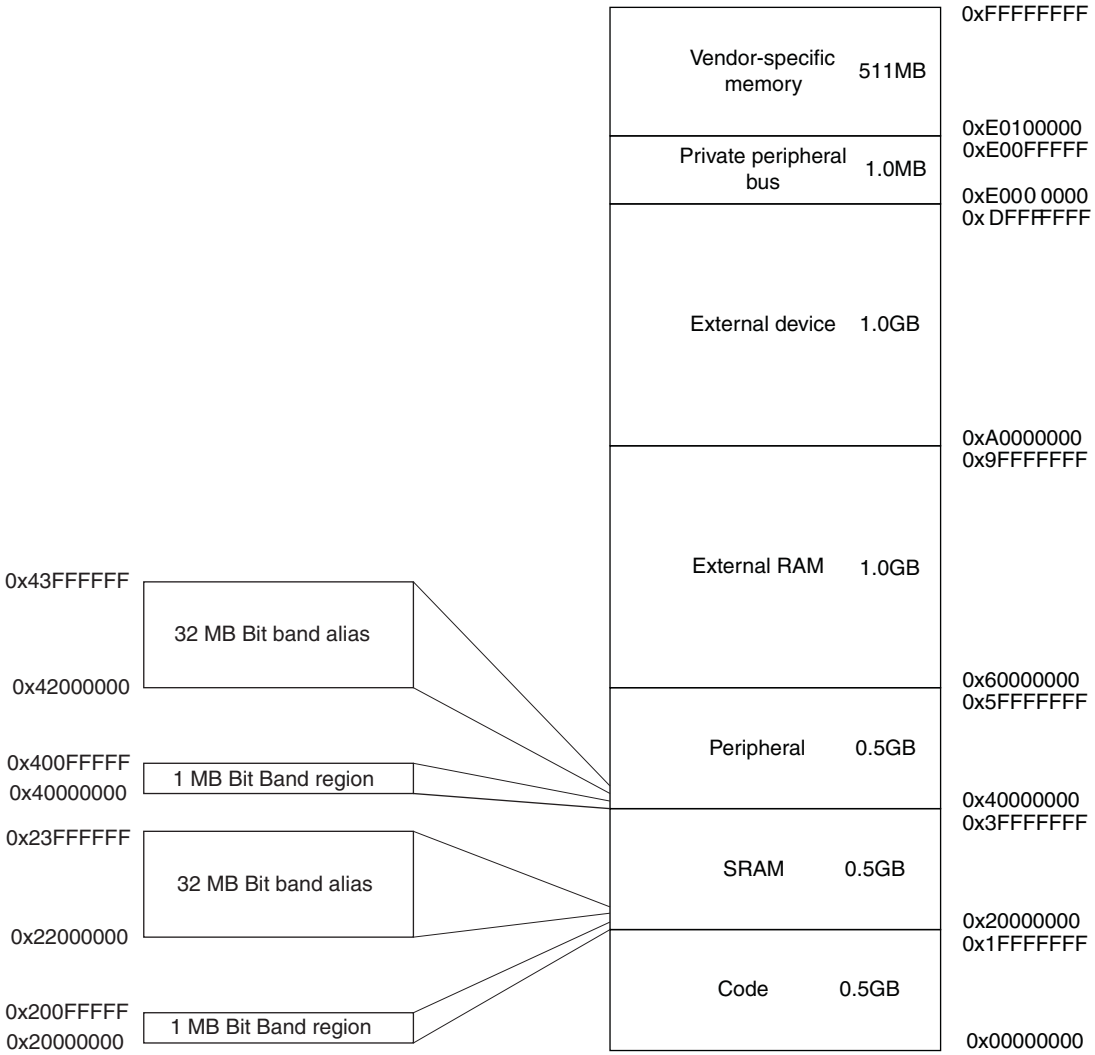
The following sections give more information about the CMSIS:

- Section 12.5.3 "Power Management Programming Hints"
- Section 12.6.2 "CMSIS Functions"
- Section 12.8.2.1 "NVIC Programming Hints"

12.4.2 Memory Model

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

Figure 12-3. Memory Map



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

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

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

12.4.2.1 Memory Regions, Types and Attributes

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

Memory Types

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

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

Additional Memory Attributes

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

12.4.2.2 Memory System Ordering of Memory Accesses

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

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

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

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.

12.4.2.3 Behavior of Memory Accesses

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

Table 12-4. 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 12-6.
0x40000000 - 0x5FFFFFFF	Peripheral	Device[1]	XN	This region includes bit band and bit band alias areas, see Table 12-6.
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 Shared Memory

When a system includes shared memory, some memory regions have additional access constraints, and some regions are subdivided, as Table 12-5 shows:

Table 12-5. Memory Region Shareability Policies

Address Range	Memory Region	Memory Type	Shareability
0x00000000 - 0x1FFFFFFF	Code	Normal[1]	-
0x20000000 - 0x3FFFFFFF	SRAM	Normal[1]	-
0x40000000 - 0x5FFFFFFF	Peripheral	Device[1]	-
0x60000000 - 0x7FFFFFFF	External RAM	Normal[1]	-
0x80000000 - 0x9FFFFFFF			
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.

Instruction Prefetch and Branch Prediction

The Cortex-M4 processor:

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

12.4.2.4 Software Ordering of Memory Accesses

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

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

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

DMB

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

DSB

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

ISB

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

MPU Programming

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

12.4.2.5 Bit-banding

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

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

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

Table 12-6. SRAM Memory Bit-banding Regions

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

Table 12-7. Peripheral Memory Bit-banding Regions

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

Notes: 1. A word access to the SRAM or peripheral bit-band alias regions map to a single bit in the SRAM or peripheral bit-band region.

2. Bit-band accesses can use byte, halfword, or word transfers. The bit-band transfer size matches the transfer size of the instruction making the bit-band access.

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

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

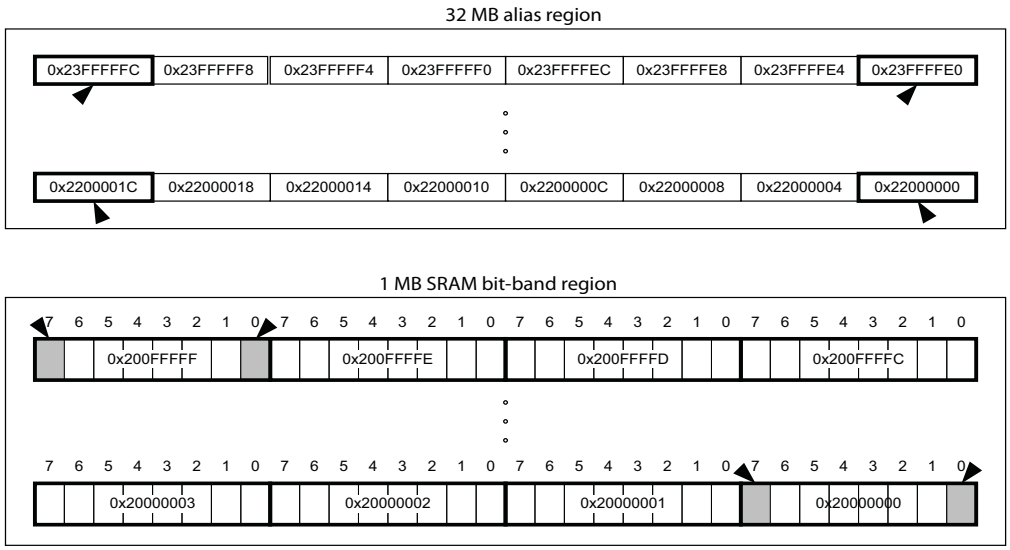
where:

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

Figure 12-4 shows examples of bit-band mapping between the SRAM bit-band alias region and the SRAM bit-band region:

- The alias word at 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 12-4. Bit-band Mapping



Directly Accessing an Alias Region

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

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

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

Reading a word in the alias region:

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

Directly Accessing a Bit-band Region

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

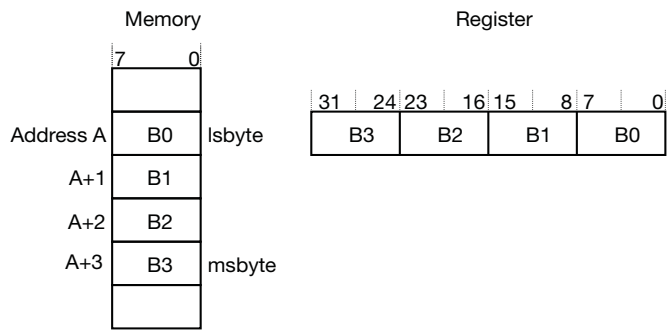
12.4.2.6 Memory Endianness

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

Little-endian Format

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

Figure 12-5. *Little-endian Format*



12.4.2.7 Synchronization Primitives

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

A pair of synchronization primitives comprises:

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

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

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

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

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

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

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

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

   0: The read-modify-write completed successfully.

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

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

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

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

The processor removes its exclusive access tag if:

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

In a multiprocessor implementation:

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

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

12.4.2.8 Programming Hints for the Synchronization Primitives

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

Table 12-8. CMSIS Functions for Exclusive Access Instructions

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

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

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

12.4.3 Exception Model

This section describes the exception model.

12.4.3.1 Exception States

Each exception is in one of the following states:

Inactive

The exception is not active and not pending.

Pending

The exception is waiting to be serviced by the processor.

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

Active

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

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

Active and Pending

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

12.4.3.2 Exception Types

The exception types are:

Reset

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

Non Maskable Interrupt (NMI)

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

NMIs cannot be:

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

Hard Fault

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

Memory Management Fault (MemManage)

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

Bus Fault

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

Usage Fault

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

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

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

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

SVCall

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

PendSV

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

SysTick

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

Interrupt (IRQ)

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

Table 12-9. Properties of the Different Exception Types

Exception Number[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 12-9 shows as having configurable priority, see:

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

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

12.4.3.3 Exception Handlers

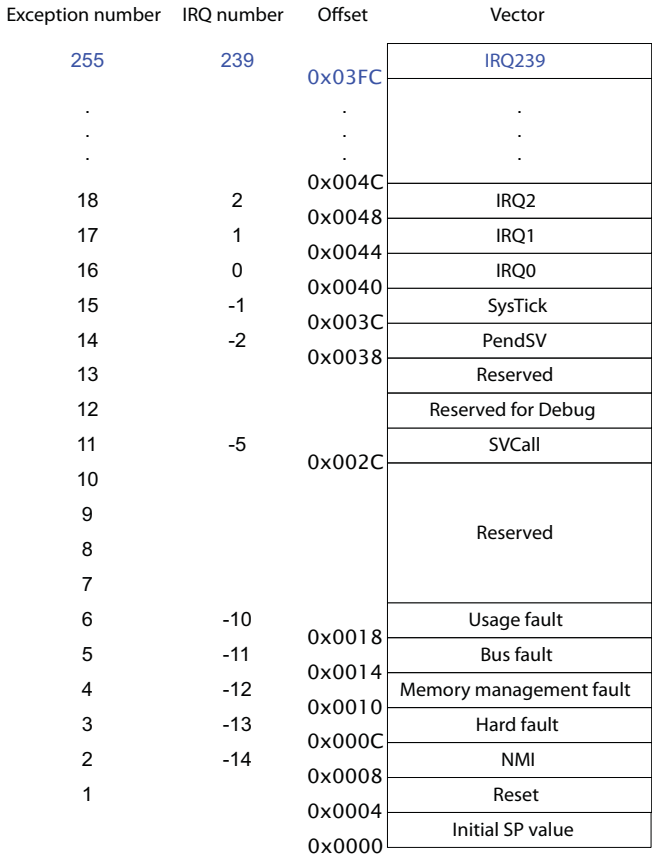
The processor handles exceptions using:

- Interrupt Service Routines (ISRs)
  Interrupts IRQ0 to IRQ40 are the exceptions handled by ISRs.
- Fault Handlers
  Hard fault, memory management fault, usage fault, bus fault are fault exceptions handled by the fault handlers.
- System Handlers
  NMI, PendSV, SVCall SysTick, and the fault exceptions are all system exceptions that are handled by system handlers.

12.4.3.4 Vector Table

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

Figure 12-6. Vector Table



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".

12.4.3.5 Exception Priorities

As Table 12-9 shows, all exceptions have an associated priority, with:

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

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

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

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

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

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

12.4.3.6 Interrupt Priority Grouping

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

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

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

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

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

12.4.3.7 Exception Entry and Return

Descriptions of exception handling use the following terms:

Preemption

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

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

Return

This occurs when the exception handler is completed, and:

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

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

Tail-chaining

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

Late-arriving

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

Exception Entry

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

When one exception preempts another, the exceptions are nested.

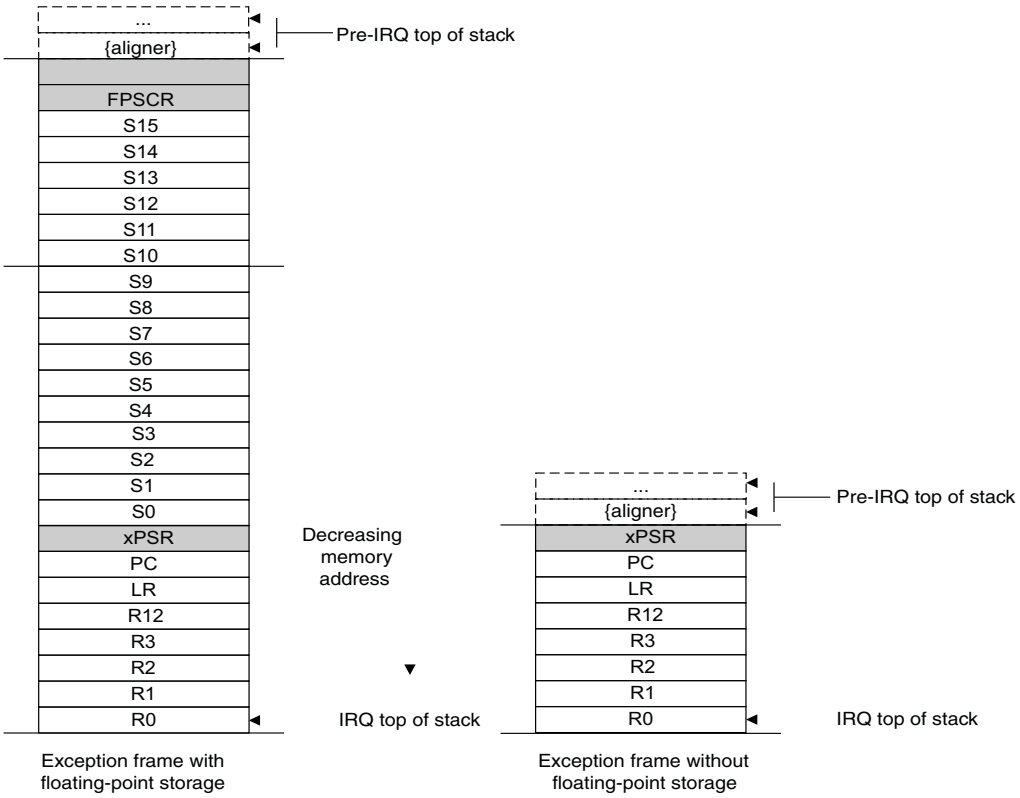
Sufficient priority means that the exception has more priority than any limits set by the mask registers, see "Exception Mask Registers". An exception with less priority than this is pending but is not handled by the processor.

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

When using floating-point routines, the Cortex-M4 processor automatically stacks the architected floating-point state on exception entry. Figure 12-7 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 12-7 shows this stack frame also.

Figure 12-7. Exception Stack Frame



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

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

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

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

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

Exception Return

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

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

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

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

Table 12-10. Exception Return Behavior

EXC_RETURN[31:0]	Description
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.

12.4.3.8 Fault Handling

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

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

Fault Types

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

Table 12-11. Faults

Fault	Handler	Bit Name	Fault Status Register
Bus error on a vector read	Hard fault	VECTTBL	"Hard Fault Status Register"
Fault escalated to a hard fault		FORCED	

Table 12-11. Faults (Continued)

Fault	Handler	Bit Name	Fault Status Register
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)	
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 12-12.

Table 12-12. 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.

12.5 Power Management

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

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

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

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

12.5.1 Entering Sleep Mode

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

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

12.5.1.1 Wait for Interrupt

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

12.5.1.2 Wait for Event

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

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

See "WFE" for more information.

12.5.1.3 Sleep-on-exit

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

12.5.2 Wakeup from Sleep Mode

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

12.5.2.1 Wakeup from WFI or Sleep-on-exit

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

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

12.5.2.2 Wakeup from WFE

The processor wakes up if:

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

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

12.5.2.3 External Event Input

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

12.5.3 Power Management Programming Hints

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

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

12.6 Cortex-M4 Instruction Set

12.6.1 Instruction Set Summary

The processor implements a version of the Thumb instruction set. Table 12-13 lists the supported instructions.

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

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

Table 12-13. Cortex-M4 Instructions

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

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

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

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

Mnemonic	Operands	Description	Flags
QSUB16	{Rd,} Rn, Rm	Saturating Subtract 16	-
QSUB8	{Rd,} Rn, Rm	Saturating Subtract 8	-
RBIT	Rd, Rn	Reverse Bits	-
REV	Rd, Rn	Reverse byte order in a word	-
REV16	Rd, Rn	Reverse byte order in each halfword	-
REVSH	Rd, Rn	Reverse byte order in bottom halfword and sign extend	-
ROR, RORS	Rd, Rm, <Rs|#n>	Rotate Right	N,Z,C
RRX, RRXS	Rd, Rm	Rotate Right with Extend	N,Z,C
RSB, RSBS	{Rd,} Rn, Op2	Reverse Subtract	N,Z,C,V
SADD16	{Rd,} Rn, Rm	Signed Add 16	GE
SADD8	{Rd,} Rn, Rm	Signed Add 8 and Subtract with Exchange	GE
SASX	{Rd,} Rn, Rm	Signed Add	GE
SBC, SBCS	{Rd,} Rn, Op2	Subtract with Carry	N,Z,C,V
SBFX	Rd, Rn, #lsb, #width	Signed Bit Field Extract	-
SDIV	{Rd,} Rn, Rm	Signed Divide	-
SEL	{Rd,} Rn, Rm	Select bytes	-
SEV	-	Send Event	-
SHADD16	{Rd,} Rn, Rm	Signed Halving Add 16	-
SHADD8	{Rd,} Rn, Rm	Signed Halving Add 8	-
SHASX	{Rd,} Rn, Rm	Signed Halving Add and Subtract with Exchange	-
SHSAX	{Rd,} Rn, Rm	Signed Halving Subtract and Add with Exchange	-
SHSUB16	{Rd,} Rn, Rm	Signed Halving Subtract 16	-
SHSUB8	{Rd,} Rn, Rm	Signed Halving Subtract 8	-
SMLABB, SMLABT, SMLATB, SMLATT	Rd, Rn, Rm, Ra	Signed Multiply Accumulate Long (halfwords)	Q
SMLAD, SMLADX	Rd, Rn, Rm, Ra	Signed Multiply Accumulate Dual	Q
SMLAL	RdLo, RdHi, Rn, Rm	Signed Multiply with Accumulate (32 x 32 + 64), 64-bit result	-
SMLALBB, SMLALBT, SMLALTB, SMLALTT	RdLo, RdHi, Rn, Rm	Signed Multiply Accumulate Long, halfwords	-
SMLALD, SMLALDX	RdLo, RdHi, Rn, Rm	Signed Multiply Accumulate Long Dual	-
SMLAWB, SMLAWT	Rd, Rn, Rm, Ra	Signed Multiply Accumulate, word by halfword	Q
SMLSD	Rd, Rn, Rm, Ra	Signed Multiply Subtract Dual	Q
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 12-13. Cortex-M4 Instructions (Continued)

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

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

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

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

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

12.6.2 CMSIS Functions

ISO/IEC cannot directly access some Cortex-M4 instructions. This section describes intrinsic functions that can generate these instructions, provided by the 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 12-14. CMSIS Functions to Generate some Cortex-M4 Instructions

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

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

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

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

12.6.3 Instruction Descriptions

12.6.3.1 Operands

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

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

12.6.3.2 Restrictions when Using PC or SP

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

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

12.6.3.3 Flexible Second Operand

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

Operand2 can be a:

- “Constant”.
- “Register with Optional Shift”.

Constant

Specify an Operand2 constant in the form:

```
#constant
```

where *constant* can be:

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

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

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

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

Instruction Substitution

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

12.6.3.4 Shift Operations

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

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

The permitted shift lengths depend on the shift type and the instruction. If the shift length is 0, no shift occurs. Register shift operations update the carry flag except when the specified shift length is 0. The following 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

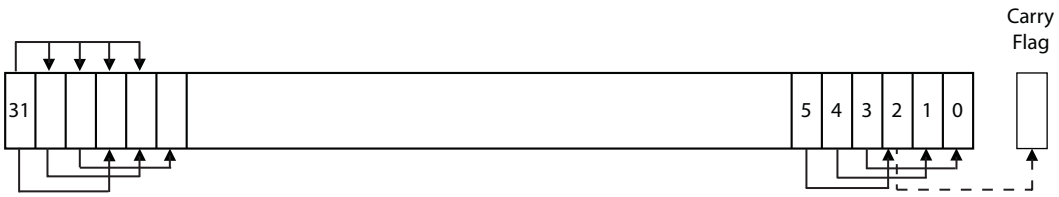
Arithmetic shift right by *n* bits moves the left-hand 32-n bits of the register, *Rm*, to the right by *n* places, into the right-hand 32-n bits of the result. And it copies the original bit[31] of the register into the left-hand *n* bits of the result. See Figure 12-8.

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

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

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

Figure 12-8. ASR #3



LSR

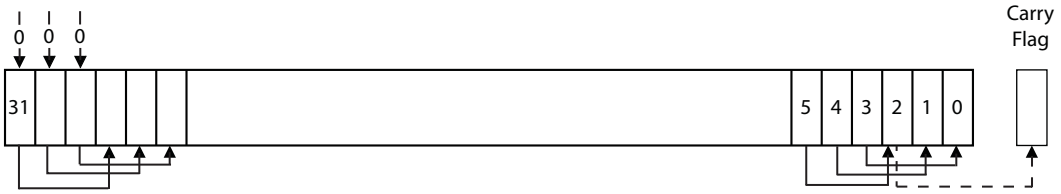
Logical shift right by n bits moves the left-hand 32-n bits of the register Rm, to the right by *n* places, into the right-hand 32-n bits of the result. And it sets the left-hand n bits of the result to 0. See Figure 12-9.

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

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

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

Figure 12-9. LSR #3



LSL

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

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

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

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

Figure 12-10. LSL #3



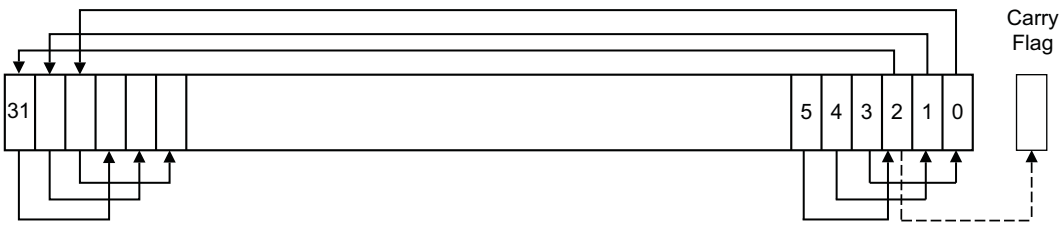
ROR

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 12-11.

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

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

Figure 12-11. ROR #3

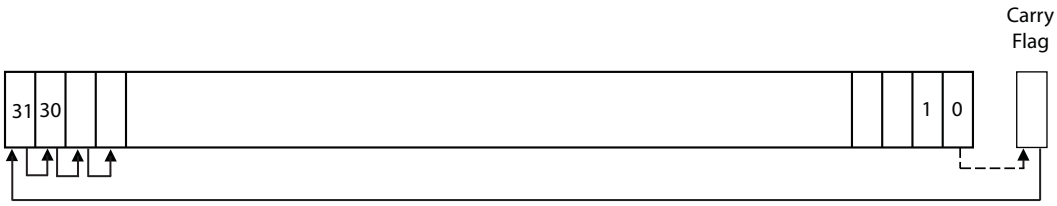


RRX

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

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

Figure 12-12. RRX



12.6.3.5 Address Alignment

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

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

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

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

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

12.6.3.6 PC-relative Expressions

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

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

12.6.3.7 Conditional Execution

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

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

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

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

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

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

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

This section describes:

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

Condition Flags

The APSR contains the following condition flags:

N Set to 1 when the result of the operation was negative, cleared to 0 otherwise.

Z Set to 1 when the result of the operation was zero, cleared to 0 otherwise.

C Set to 1 when the operation resulted in a carry, cleared to 0 otherwise.

V Set to 1 when the operation caused overflow, cleared to 0 otherwise.

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

A carry occurs:

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

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

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

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

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

Condition Code Suffixes

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

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

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

Table 12-16. Condition Code Suffixes

Suffix	Flags	Meaning
EQ	Z = 1	Equal
NE	Z = 0	Not equal
CS or HS	C = 1	Higher or same, unsigned ≥
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
```

12.6.3.8 Instruction Width Selection

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

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

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

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

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

12.6.4 Memory Access Instructions

The table below shows the memory access instructions:

Table 12-17. Memory Access Instructions

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

12.6.4.1 ADR

Load PC-relative address.

Syntax

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

where:

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

Rd is the destination register.

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

Operation

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

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

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

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

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

Restrictions

Rd must not be SP and must not be PC.

Condition Flags

This instruction does not change the flags.

Examples

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

12.6.4.2 LDR and STR, Immediate Offset

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

Syntax

```
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 12-18. Offset Ranges

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

Restrictions

For load instructions:

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

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

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

For store instructions:

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

Condition Flags

These instructions do not change the flags.

Examples

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

12.6.4.3 LDR and STR, Register Offset

Load and Store with register offset.

Syntax

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

where:

op is one of:

- LDR Load Register.
- STR Store Register.

type is one of:

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

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

Rt is the register to load or store.

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

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

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

Operation

LDR instructions load a register with a value from memory.

STR instructions store a register value into memory.

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

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

Restrictions

In these instructions:

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

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

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

Condition Flags

These instructions do not change the flags.

Examples

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

12.6.4.4 LDR and STR, Unprivileged

Load and Store with unprivileged access.

Syntax

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

where:

op is one of:

- LDR Load Register.
- STR Store Register.

type is one of:

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

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

Rt is the register to load or store.

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

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

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

Operation

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

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

Restrictions

In these instructions:

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

Condition Flags

These instructions do not change the flags.

Examples

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

12.6.4.5 LDR, PC-relative

Load register from memory.

Syntax

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

where:

type is one of:

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

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

Rt is the register to load or store.

Rt2 is the second register to load or store.

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

Operation

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

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

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

Table 12-19. Offset Ranges

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

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

Restrictions

In these instructions:

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

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

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

Condition Flags

These instructions do not change the flags.

Examples

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

12.6.4.6 LDM and STM

Load and Store Multiple registers.

Syntax

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

where:

op — is one of:

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

addr_mode — is any one of the following:

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

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

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

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

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

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

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

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

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
```

12.6.4.7 PUSH and POP

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

Syntax

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

where:

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

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

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

Operation

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

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

See "LDM and STM" for more information.

Restrictions

In these instructions:

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

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

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

Condition Flags

These instructions do not change the flags.

Examples

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

12.6.4.8 LDREX and STREX

Load and Store Register Exclusive.

Syntax

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

where:

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

Operation

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

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

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

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

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

Restrictions

In these instructions:

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

Condition Flags

These instructions do not change the flags.

Examples

```
MOV     R1, #0x1            ; Initialize the 'lock taken' value try
LDREX   R0, [LockAddr]      ; Load the lock value
CMP     R0, #0              ; Is the lock free?
ITT     EQ                  ; IT instruction for STREXEQ and CMPEQ
STREXEQ R0, R1, [LockAddr]  ; Try and claim the lock
CMPEQ   R0, #0              ; Did this succeed?
BNE     try                 ; No – try again
....                        ; Yes – we have the lock
```

12.6.4.9 CLREX

Clear Exclusive.

Syntax

`CLREX{`*cond*`}`

where:

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

Operation

Use CLREX to make the next STREX, STREXB, or STREXH instruction write 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

12.6.5 General Data Processing Instructions

The table below shows the data processing instructions:

Table 12-20. Data Processing Instructions

Mnemonic	Description
ADC	Add with Carry
ADD	Add
ADDW	Add
AND	Logical AND
ASR	Arithmetic Shift Right
BIC	Bit Clear
CLZ	Count leading zeros
CMN	Compare Negative
CMP	Compare
EOR	Exclusive OR
LSL	Logical Shift Left
LSR	Logical Shift Right
MOV	Move
MOVT	Move Top
MOVW	Move 16-bit constant
MVN	Move NOT
ORN	Logical OR NOT
ORR	Logical OR
RBIT	Reverse Bits
REV	Reverse byte order in a word
REV16	Reverse byte order in each halfword
REVSH	Reverse byte order in bottom halfword and sign extend
ROR	Rotate Right
RRX	Rotate Right with Extend
RSB	Reverse Subtract
SADD16	Signed Add 16
SADD8	Signed Add 8
SASX	Signed Add and Subtract with Exchange
SSAX	Signed Subtract and Add with Exchange
SBC	Subtract with Carry
SHADD16	Signed Halving Add 16
SHADD8	Signed Halving Add 8
SHASX	Signed Halving Add and Subtract with Exchange
SHSAX	Signed Halving Subtract and Add with Exchange
SHSUB16	Signed Halving Subtract 16

Table 12-20. Data Processing Instructions (Continued)

Mnemonic	Description
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

12.6.5.1 ADD, ADC, SUB, SBC, and RSB

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

Syntax

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

where:

op — is one of:

ADD Add.

ADC Add with Carry.

SUB Subtract.

SBC Subtract with Carry.

RSB Reverse Subtract.

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

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

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

Rn — is the register holding the first operand.

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

imm12 is any value in the range 0 - 4095.

Operation

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

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

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

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

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

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

See also “ADR”.

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

Restrictions

In these instructions:

- *Operand2* must not be SP and must not be PC.
- *Rd* can be SP only in ADD and SUB, and only with the additional restrictions:
  - *Rn* must also be SP.
  - Any shift in *Operand2* must be limited to a maximum of 3 bits using LSL.
- *Rn* can be SP only in ADD and SUB.
- *Rd* can be PC only in the ADD{*cond*} PC, PC, Rm instruction where:
  - The user must not specify the S suffix.
  - *Rm* must not be PC and must not be SP.
  - If the instruction is conditional, it must be the last instruction in the IT block.
- With the exception of the ADD{*cond*} PC, PC, Rm instruction, *Rn* can be PC only in ADD and SUB, and only with the additional restrictions:
  - The user must not specify the S suffix.
  - The second operand must be a constant in the range 0 to 4095.
  - Note: When using the PC for an addition or a subtraction, bits[1:0] of the PC are rounded to 0b00 before performing the calculation, making the base address for the calculation word-aligned.
  - Note: To generate the address of an instruction, the constant based on the value of the PC must be adjusted. ARM recommends to use the ADR instruction instead of ADD or SUB with *Rn* equal to the PC, because the assembler automatically calculates the correct constant for the ADR instruction.

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

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

Condition Flags

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

Examples

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

Multiword Arithmetic Examples

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

64-bit Addition Example

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

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

96-bit Subtraction Example

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

12.6.5.2 AND, ORR, EOR, BIC, and ORN

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

Syntax

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

where:

op — is one of:

AND logical AND.

ORR logical OR, or bit set.

EOR logical Exclusive OR.

BIC logical AND NOT, or bit clear.

ORN logical OR NOT.

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

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

Rd — is the destination register.

Rn — is the register holding the first operand.

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

Operation

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

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

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

If S is specified, these instructions:

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

Examples

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

12.6.5.3 ASR, LSL, LSR, ROR, and RRX

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

Syntax

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

where:

op — is one of:

ASR Arithmetic Shift Right.

LSL Logical Shift Left.

LSR Logical Shift Right.

ROR Rotate Right.

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

Rd — is the destination register.

Rm — is the register holding the value to be shifted.

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

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

ASR shift length from 1 to 32

LSL shift length from 0 to 31

LSR shift length from 1 to 32

ROR shift length from 0 to 31

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

Operation

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

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

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

If S is specified:

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

Examples

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

12.6.5.4 CLZ

Count Leading Zeros.

Syntax

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

where:

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

Rd — is the destination register.

Rm — is the operand register.

Operation

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

This instruction does not change the flags.

Examples

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

12.6.5.5 CMP and CMN

Compare and Compare Negative.

Syntax

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

where:

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

Rn — is the register holding the first operand.

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

Operation

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

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

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

Restrictions

In these instructions:

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

Condition Flags

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

Examples

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

12.6.5.6 MOV and MVN

Move and Move NOT.

Syntax

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

where:

S	is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see "Conditional Execution".
cond	is an optional condition code, see "Conditional Execution".
Rd	is the destination register.
Operand2	is a flexible second operand. See "Flexible Second Operand" for details of the options.
imm16	is any value in the range 0-65535.

Operation

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

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

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

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

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

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

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

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

Restrictions

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

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

When *Rd* is PC in a MOV instruction:

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

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

Condition Flags

If S is specified, these instructions:

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

Examples

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

12.6.5.7 MOVT

Move Top.

Syntax

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

where:

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

Rd is the destination register.

imm16 is a 16-bit immediate constant.

Operation

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

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

Restrictions

Rd must not be SP and must not be PC.

Condition Flags

This instruction does not change the flags.

Examples

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

12.6.5.8 REV, REV16, REVSH, and RBIT

Reverse bytes and Reverse bits.

Syntax

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

where:

op is any of:

REV Reverse byte order in a word.

REV16 Reverse byte order in each halfword independently.

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

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

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

Rd is the destination register.

Rn is the register holding the operand.

Operation

Use these instructions to change endianness of data:

REV converts either:

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

REV16 converts either:

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

REVSH converts either:

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

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

12.6.5.9 SADD16 and SADD8

Signed Add 16 and Signed Add 8

Syntax

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

where:

op is any of:

SADD16 Performs two 16-bit signed integer additions.

SADD8 Performs four 8-bit signed integer additions.

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

Rd is the destination register.

Rn is the first register holding the operand.

Rm is the second register holding the operand.

Operation

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

The SADD16 instruction:

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

The SADD8 instruction:

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

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

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

12.6.5.10 SHADD16 and SHADD8

Signed Halving Add 16 and Signed Halving Add 8

Syntax

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

where:

op is any of:

SHADD16 Signed Halving Add 16.

SHADD8 Signed Halving Add 8.

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

Rd is the destination register.

Rn is the first operand register.

Rm is the second operand register.

Operation

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

The SHADD16 instruction:

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

The SHADDB8 instruction:

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

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

12.6.5.11 SHASX and SHSAX

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

Syntax

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

where:

op	is any of:
	SHASX Add and Subtract with Exchange and Halving.
	SHSAX Subtract and Add with Exchange and Halving.
cond	is an optional condition code, see “Conditional Execution”.
Rd	is the destination register.
Rn, Rm	are registers holding the first and second operands.

Operation

The SHASX instruction:

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

The SHSAX instruction:

1. Subtracts the bottom halfword of the second operand from the top highword of the first operand.
2. Writes the halfword result of the 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.
```

12.6.5.12 SHSUB16 and SHSUB8

Signed Halving Subtract 16 and Signed Halving Subtract 8

Syntax

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

where:

op — is any of:

- SHSUB16 Signed Halving Subtract 16.
- SHSUB8 Signed Halving Subtract 8.

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

Rd — is the destination register.

Rn — is the first operand register.

Rm — is the second operand register.

Operation

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

The SHSUB16 instruction:

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

The SHSUB8 instruction:

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

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

12.6.5.13 SSUB16 and SSUB8

Signed Subtract 16 and Signed Subtract 8

Syntax

op{cond}{Rd,} Rn, Rm

where:

op is any of:

SSUB16 Performs two 16-bit signed integer subtractions.

SSUB8 Performs four 8-bit signed integer subtractions.

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

Rd is the destination register.

Rn is the first operand register.

Rm is the second operand register.

Operation

Use these instructions to change endianness of data:

The SSUB16 instruction:

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

The SSUB8 instruction:

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

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

12.6.5.14 SASX and SSAX

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

Syntax

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

where:

op is any of:

SASX Signed Add and Subtract with Exchange.

SSAX Signed Subtract and Add with Exchange.

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

Rd is the destination register.

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

Operation

The SASX instruction:

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

The SSAX instruction:

1. Subtracts the signed bottom halfword of the second operand from the top signed highword of the first operand.
2. Writes the signed result of the 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.
```

12.6.5.15 TST and TEQ

Test bits and Test Equivalence.

Syntax

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

where

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

Rn is the register holding the first operand.

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

Operation

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

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

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

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

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

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions:

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

Examples

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

12.6.5.16 UADD16 and UADD8

Unsigned Add 16 and Unsigned Add 8

Syntax

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

where:

op — is any of:

UADD16 Performs two 16-bit unsigned integer additions.

UADD8 Performs four 8-bit unsigned integer additions.

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

Rd — is the destination register.

Rn — is the first register holding the operand.

Rm — is the second register holding the operand.

Operation

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

The UADD16 instruction:

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

The UADD8 instruction:

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

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

12.6.5.17 UASX and USAX

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

Syntax

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

where:

op — is one of:

UASX Add and Subtract with Exchange.

USAX Subtract and Add with Exchange.

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

Rd — is the destination register.

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

Operation

The UASX instruction:

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

The USAX instruction:

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

Examples

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

12.6.5.18 UHADD16 and UHADD8

Unsigned Halving Add 16 and Unsigned Halving Add 8

Syntax

op{cond}{Rd,} Rn, Rm

where:

op is any of:

UHADD16 Unsigned Halving Add 16.

UHADD8 Unsigned Halving Add 8.

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

Rd is the destination register.

Rn is the register holding the first operand.

Rm is the register holding the second operand.

Operation

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

The UHADD16 instruction:

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

The UHADD8 instruction:

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

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

12.6.5.19 UHASX and UHSAX

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

Syntax

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

where:

op is one of:

UHASX Add and Subtract with Exchange and Halving.

UHSAX Subtract and Add with Exchange and Halving.

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

Rd is the destination register.

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

Operation

The UHASX instruction:

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

The UHSAX instruction:

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

Examples

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

12.6.5.20 UHSUB16 and UHSUB8

Unsigned Halving Subtract 16 and Unsigned Halving Subtract 8

Syntax

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

where:

op is any of:

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

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

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

Rd is the destination register.

Rn is the first register holding the operand.

Rm is the second register holding the operand.

Operation

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

The UHSUB16 instruction:

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

The UHSUB8 instruction:

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

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

12.6.5.21 SEL

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

Syntax

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

where:

c, q — are standard assembler syntax fields.

Rd — is the destination register.

Rn — is the first register holding the operand.

Rm — is the second register holding the operand.

Operation

The SEL instruction:

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

Restrictions

None.

Condition Flags

These instructions do not change the flags.

Examples

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

12.6.5.22 USAD8

Unsigned Sum of Absolute Differences

Syntax

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

where:

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

Rd — is the destination register.

Rn — is the first operand register.

Rm — is the second operand register.

Operation

The USAD8 instruction:

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

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

12.6.5.23 USADA8

Unsigned Sum of Absolute Differences and Accumulate

Syntax

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

where:

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

Rd is the destination register.

Rn is the first operand register.

Rm is the second operand register.

Ra is the register that contains the accumulation value.

Operation

The USADA8 instruction:

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

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

12.6.5.24 USUB16 and USUB8

Unsigned Subtract 16 and Unsigned Subtract 8

Syntax

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

where

op is any of:

USUB16 Unsigned Subtract 16.

USUB8 Unsigned Subtract 8.

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

Rd is the destination register.

Rn is the first operand register.

Rm is the second operand register.

Operation

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

The USUB16 instruction:

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

The USUB8 instruction:

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

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

12.6.6 Multiply and Divide Instructions

The table below shows the multiply and divide instructions:

Table 12-21. Multiply and Divide Instructions

Mnemonic	Description
MLA	Multiply with Accumulate, 32-bit result
MLS	Multiply and Subtract, 32-bit result
MUL	Multiply, 32-bit result
SDIV	Signed Divide
SMLA[B,T]	Signed Multiply Accumulate (halfwords)
SMLAD, SMLADX	Signed Multiply Accumulate Dual
SMLAL	Signed Multiply with Accumulate (32 x 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 (32 x 32), 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 x 32 + 32 + 32), 64-bit result
UMLAL	Unsigned Multiply with Accumulate (32 x 32 + 64), 64-bit result
UMULL	Unsigned Multiply (32 x 32), 64-bit result

12.6.6.1 MUL, MLA, and MLS

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

Syntax

```
MUL{S}{cond} {Rd,} Rn, Rm ; Multiply
MLA{cond} Rd, Rn, Rm, Ra  ; Multiply with accumulate
MLS{cond} Rd, Rn, Rm, Ra  ; Multiply with subtract
```

where:

cond	is an optional condition code, see "Conditional Execution".
S	is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see "Conditional Execution".
Rd	is the destination register. If *Rd* is omitted, the destination register is *Rn*.
Rn, Rm	are registers holding the values to be multiplied.
Ra	is a register holding the value to be added or subtracted from.

Operation

The MUL instruction multiplies the values from *Rn* and *Rm*, and places the least significant 32 bits of the result in *Rd*.

The MLA instruction multiplies the values from *Rn* and *Rm*, adds the value from *Ra*, and places the least significant 32 bits of the result in *Rd*.

The MLS instruction multiplies the values from *Rn* and *Rm*, subtracts the product from the value from *Ra*, and places the least significant 32 bits of the result in *Rd*.

The results of these instructions do not depend on whether the operands are signed or unsigned.

Restrictions

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

If the S suffix is used with the MUL instruction:

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

Condition Flags

If S is specified, the MUL instruction:

- Updates the N and Z flags according to the result.
- Does not affect the C and V flags.

Examples

```
MUL    R10, R2, R5     ; Multiply, R10 = R2 x R5
MLA    R10, R2, R1, R5 ; Multiply with accumulate, R10 = (R2 x R1) + R5
MULS   R0, R2, R2      ; Multiply with flag update, R0 = R2 x R2
MULLT  R2, R3, R2      ; Conditionally multiply, R2 = R3 x R2
MLS    R4, R5, R6, R7  ; Multiply with subtract, R4 = R7 - (R5 x R6)
```

12.6.6.2 UMULL, UMAAL, UMLAL

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

Syntax

```
op{cond} RdLo, RdHi, Rn, Rm
```

where:

op is one of:

UMULL Unsigned Long Multiply.

UMAAL Unsigned Long Multiply with Accumulate Accumulate.

UMLAL Unsigned Long Multiply, with Accumulate.

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

RdHi, RdLo are the destination registers. For UMAAL, UMLAL and UMLAL they also hold the accumulating value.

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

Operation

These instructions interpret the values from *Rn* and *Rm* as unsigned 32-bit integers.
The UMULL instruction:

- Multiplies the two unsigned integers in the first and second operands.
- Writes the least significant 32 bits of the result in *RdLo.*
- Writes the most significant 32 bits of the result in *RdHi*.

The UMAAL instruction:

- Multiplies the two unsigned 32-bit integers in the first and second operands.
- Adds the unsigned 32-bit integer in *RdHi* to the 64-bit result of the multiplication.
- Adds the unsigned 32-bit integer in *RdLo* to the 64-bit result of the addition.
- Writes the top 32-bits of the result to *RdHi.*
- Writes the lower 32-bits of the result to *RdLo*.

The UMLAL instruction:

- Multiplies the two unsigned integers in the first and second operands.
- Adds the 64-bit result to the 64-bit unsigned integer contained in *RdHi* and *RdLo*.
- Writes the result back to *RdHi* and *RdLo*.

Restrictions

In these instructions:

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

Condition Flags

These instructions do not affect the condition code flags.

Examples

```
UMULL   R0, R4, R5, R6   ; Multiplies R5 and R6, writes the top 32 bits to R4
                         ; and the bottom 32 bits to R0
UMAAL   R3, R6, R2, R7   ; Multiplies R2 and R7, adds R6, adds R3, writes the
                         ; top 32 bits to R6, and the bottom 32 bits to R3
UMLAL   R2, R1, R3, R5   ; Multiplies R5 and R3, adds R1:R2, writes to R1:R2.
```

12.6.6.3 SMLA and SMLAW

Signed Multiply Accumulate (halfwords).

Syntax

```
op{XY}{cond} Rd, Rn, Rm
op{Y}{cond} Rd, Rn, Rm, Ra
```

where:

op — is one of:

SMLA Signed Multiply Accumulate Long (halfwords).

X and *Y* specifies which half of the source registers *Rn* and *Rm* are used as the first and second multiply operand.

If *X* is *B*, then the bottom halfword, bits [15:0], of *Rn* is used.
If *X* is *T*, then the top halfword, bits [31:16], of *Rn* is used.

If *Y* is *B*, then the bottom halfword, bits [15:0], of *Rm* is used.
If *Y* is *T*, then the top halfword, bits [31:16], of *Rm* is used.

SMLAW Signed Multiply Accumulate (word by halfword).

Y specifies which half of the source register *Rm* is used as the second multiply operand.

If *Y* is T, then the top halfword, bits [31:16] of *Rm* is used.

If *Y* is B, then the bottom halfword, bits [15:0] of *Rm* is used.

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

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

Rn, Rm — are registers holding the values to be multiplied.

Ra — is a register holding the value to be added or subtracted from.

Operation

The SMALBB, SMLABT, SMLATB, SMLATT instructions:

- Multiplies the specified signed halfword, top or bottom, values from *Rn* and *Rm*.
- Adds the value in *Ra* to the resulting 32-bit product.
- Writes the result of the multiplication and addition in *Rd*.

The non-specified halfwords of the source registers are ignored.

The SMLAWB and SMLAWT instructions:

- Multiply the 32-bit signed values in *Rn* with:
  - The top signed halfword of *Rm*, *T* instruction suffix.
  - The bottom signed halfword of *Rm*, *B* instruction suffix.
- Add the 32-bit signed value in *Ra* to the top 32 bits of the 48-bit product.
- Writes the result of the multiplication and addition in *Rd*.

The bottom 16 bits of the 48-bit product are ignored.

If overflow occurs during the addition of the accumulate value, the instruction sets the Q flag in the APSR. No overflow can occur during the multiplication.

Restrictions

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

Condition Flags

If an overflow is detected, the Q flag is set.

Examples

```
SMLABB  R5, R6, R4, R1  ; Multiplies bottom halfwords of R6 and R4, adds
                        ; R1 and writes to R5
SMLATB  R5, R6, R4, R1  ; Multiplies top halfword of R6 with bottom halfword
                        ; of R4, adds R1 and writes to R5
SMLATT  R5, R6, R4, R1  ; Multiplies top halfwords of R6 and R4, adds
                        ; R1 and writes the sum to R5
SMLABT  R5, R6, R4, R1  ; Multiplies bottom halfword of R6 with top halfword
                        ; of R4, adds R1 and writes to R5
SMLABT  R4, R3, R2      ; Multiplies bottom halfword of R4 with top halfword of
                        ; R3, adds R2 and writes to R4
SMLAWB  R10, R2, R5, R3 ; Multiplies R2 with bottom halfword of R5, adds
                        ; R3 to the result and writes top 32-bits to R10
SMLAWT  R10, R2, R1, R5 ; Multiplies R2 with top halfword of R1, adds R5
                        ; and writes top 32-bits to R10.
```

12.6.6.4 SMLAD

Signed Multiply Accumulate Long Dual

Syntax

```
op{X}{cond} Rd, Rn, Rm, Ra  ;
```

where:

op — is one of:

SMLAD Signed Multiply Accumulate Dual.

SMLADX Signed Multiply Accumulate Dual Reverse.

X specifies which halfword of the source register *Rn* is used as the multiply operand.
If *X* is omitted, the multiplications are bottom × bottom and top × top.
If *X* is present, the multiplications are bottom × top and top × bottom.

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

Rd — is the destination register.

Rn — is the first operand register holding the values to be multiplied.

Rm — the second operand register.

Ra — is the accumulate value.

Operation

The SMLAD and SMLADX instructions regard the two operands as four halfword 16-bit values. The SMLAD and SMLADX instructions:

- If *X* is not present, multiply the top signed halfword value in *Rn* with the top signed halfword of *Rm* and the bottom signed halfword values in *Rn* with the bottom signed halfword of *Rm*.
- Or if *X* is present, multiply the top signed halfword value in *Rn* with the bottom signed halfword of *Rm* and the bottom signed halfword values in *Rn* with the top signed halfword of *Rm*.
- Add both multiplication results to the signed 32-bit value in *Ra*.
- Writes the 32-bit signed result of the multiplication and addition to *Rd*.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

```
SMLAD   R10, R2, R1, R5 ; Multiplies two halfword values in R2 with
                        ; corresponding halfwords in R1, adds R5 and
                        ; writes to R10
SMLALDX R0, R2, R4, R6  ; Multiplies top halfword of R2 with bottom
                        ; halfword of R4, multiplies bottom halfword of R2
                        ; with top halfword of R4, adds R6 and writes to
                        ; R0.
```

12.6.6.5 SMLAL and SMLALD

Signed Multiply Accumulate Long, Signed Multiply Accumulate Long (halfwords) and Signed Multiply Accumulate Long Dual.

Syntax

```
op{cond} RdLo, RdHi, Rn, Rm
op{XY}{cond} RdLo, RdHi, Rn, Rm
op{X}{cond} RdLo, RdHi, Rn, Rm
```

where:

op — is one of:

MLAL Signed Multiply Accumulate Long.

SMLAL Signed Multiply Accumulate Long (halfwords, X and Y).

X and Y specify which halfword of the source registers *Rn* and *Rm* are used as the first and second multiply operand:

If *X* is B, then the bottom halfword, bits [15:0], of *Rn* is used.
If *X* is T, then the top halfword, bits [31:16], of *Rn* is used.

If *Y* is B, then the bottom halfword, bits [15:0], of *Rm* is used.
If *Y* is T, then the top halfword, bits [31:16], of *Rm* is used.

SMLALD Signed Multiply Accumulate Long Dual.

SMLALDX Signed Multiply Accumulate Long Dual Reversed.

If the *X* is omitted, the multiplications are bottom × bottom and top × top.

If *X* is present, the multiplications are bottom × top and top × bottom.

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

RdHi, RdLo — are the destination registers.
RdLo is the lower 32 bits and *RdHi* is the upper 32 bits of the 64-bit integer.
For SMLAL, SMLALBB, SMLALBT, SMLALTB, SMLALTT, SMLALD and SMLALDX, they also hold the accumulating value.

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

Operation

The SMLAL instruction:

- Multiplies the two’s complement signed word values from *Rn* and *Rm*.
- Adds the 64-bit value in *RdLo* and *RdHi* to the resulting 64-bit product.
- Writes the 64-bit result of the multiplication and addition in *RdLo* and *RdHi*.

The SMLALBB, SMLALBT, SMLALTB and SMLALTT instructions:

- Multiplies the specified signed halfword, Top or Bottom, values from *Rn* and *Rm*.
- Adds the resulting sign-extended 32-bit product to the 64-bit value in *RdLo* and *RdHi*.
- Writes the 64-bit result of the multiplication and addition in *RdLo* and *RdHi*.

The non-specified halfwords of the source registers are ignored.

The SMLALD and SMLALDX instructions interpret the values from *Rn* and *Rm* as four halfword two's complement signed 16-bit integers. These instructions:

- If *X* is not present, multiply the top signed halfword value of *Rn* with the top signed halfword of *Rm* and the bottom signed halfword values of *Rn* with the bottom signed halfword of *Rm*.
- Or if *X* is present, multiply the top signed halfword value of *Rn* with the bottom signed halfword of *Rm* and the bottom signed halfword values of *Rn* with the top signed halfword of *Rm*.
- Add the two multiplication results to the signed 64-bit value in *RdLo* and *RdHi* to create the resulting 64-bit product.
- Write the 64-bit product in *RdLo* and *RdHi*.

Restrictions

In these instructions:

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

Condition Flags

These instructions do not affect the condition code flags.

Examples

```
SMLAL    R4, R5, R3, R8  ; Multiplies R3 and R8, adds R5:R4 and writes to
                         ; R5:R4
SMLALBT  R2, R1, R6, R7  ; Multiplies bottom halfword of R6 with top
                         ; halfword of R7, sign extends to 32-bit, adds
                         ; R1:R2 and writes to R1:R2
SMLALTB  R2, R1, R6, R7  ; Multiplies top halfword of R6 with bottom
                         ; halfword of R7,sign extends to 32-bit, adds R1:R2
                         ; and writes to R1:R2
SMLALD   R6, R8, R5, R1  ; Multiplies top halfwords in R5 and R1 and bottom
                         ; halfwords of R5 and R1, adds R8:R6 and writes to
                         ; R8:R6
SMLALDX  R6, R8, R5, R1  ; Multiplies top halfword in R5 with bottom
                         ; halfword of R1, and bottom halfword of R5 with
                         ; top halfword of R1, adds R8:R6 and writes to
                         ; R8:R6.
```

12.6.6.6 SMLSD and SMLSLD

Signed Multiply Subtract Dual and Signed Multiply Subtract Long Dual

Syntax

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

where:

op — is one of:

- SMLSD Signed Multiply Subtract Dual.
- SMLSDX Signed Multiply Subtract Dual Reversed.
- SMLSLD Signed Multiply Subtract Long Dual.
- SMLSLDX Signed Multiply Subtract Long Dual Reversed.
- SMLAW Signed Multiply Accumulate (word by halfword).

If *X* is present, the multiplications are bottom × 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.
```

12.6.6.7 SMMLA and SMMLS

Signed Most Significant Word Multiply Accumulate and Signed Most Significant Word Multiply Subtract

Syntax

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

where:

op — is one of:

SMMLA Signed Most Significant Word Multiply Accumulate.

SMMLS Signed Most Significant Word Multiply Subtract.

If the *X* is omitted, the multiplications are bottom × bottom and top × top.

R — is a rounding error flag. If *R* is specified, the result is rounded instead of being truncated. In this case the constant 0x80000000 is added to the product before the high word is extracted.

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

Rd — is the destination register.

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

Ra — is the register holding the accumulate value.

Operation

The SMMLA instruction interprets the values from *Rn* and *Rm* as signed 32-bit words.

The SMMLA instruction:

- Multiplies the values in *Rn* and *Rm*.
- Optionally rounds the result by adding 0x80000000.
- Extracts the most significant 32 bits of the result.
- Adds the value of *Ra* to the signed extracted value.
- Writes the result of the addition in *Rd*.

The SMMLS instruction interprets the values from *Rn* and *Rm* as signed 32-bit words.

The SMMLS instruction:

- Multiplies the values in *Rn* and *Rm*.
- Optionally rounds the result by adding 0x80000000.
- Extracts the most significant 32 bits of the result.
- Subtracts the extracted value of the result from the value in *Ra*.
- Writes the result of the subtraction in *Rd*.

Restrictions

In these instructions:

- Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

Examples

```
SMMLA  R0, R4, R5, R6  ; Multiplies R4 and R5, extracts top 32 bits, adds
                       ; R6, truncates and writes to R0
SMMLAR R6, R2, R1, R4  ; Multiplies R2 and R1, extracts top 32 bits, adds
                       ; R4, rounds and writes to R6
SMMLSR R3, R6, R2, R7  ; Multiplies R6 and R2, extracts top 32 bits,
                       ; subtracts R7, rounds and writes to R3
SMMLS  R4, R5, R3, R8  ; Multiplies R5 and R3, extracts top 32 bits,
                       ; subtracts R8, truncates and writes to R4.
```

12.6.6.8 SMMUL

Signed Most Significant Word Multiply

Syntax

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

where:

op — is one of:

SMMUL Signed Most Significant Word Multiply.

R — is a rounding error flag. If *R* is specified, the result is rounded instead of being truncated. In this case the constant 0x80000000 is added to the product before the high word is extracted.

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

Rd — is the destination register.

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

Operation

The SMMUL instruction interprets the values from *Rn* and *Rm* as two’s complement 32-bit signed integers. The SMMUL instruction:

- Multiplies the values from *Rn* and *Rm*.
- Optionally rounds the result, otherwise truncates the result.
- Writes the most significant signed 32 bits of the result in *Rd*.

Restrictions

In this instruction:

- Do not use SP and do not use PC.

Condition Flags

This instruction does not affect the condition code flags.

Examples

```
SMULL   R0, R4, R5  ; Multiplies R4 and R5, truncates top 32 bits
                    ; and writes to R0
SMULLR  R6, R2      ; Multiplies R6 and R2, rounds the top 32 bits
                    ; and writes to R6.
```

12.6.6.9 SMUAD and SMUSD

Signed Dual Multiply Add and Signed Dual Multiply Subtract

Syntax

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

where:

op — is one of:

SMUAD Signed Dual Multiply Add.

SMUADX Signed Dual Multiply Add Reversed.

SMUSD Signed Dual Multiply Subtract.

SMUSDX Signed Dual Multiply Subtract Reversed.

If *X* is present, the multiplications are bottom × 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.
```

12.6.6.10 SMUL and SMULW

Signed Multiply (halfwords) and Signed Multiply (word by halfword).

Syntax

```
op{XY}{cond} Rd, Rn, Rm
op{Y}{cond} Rd. Rn, Rm
```

For *SMULXY* only:

op is one of:

SMUL{*XY*} Signed Multiply (halfwords).

X and *Y* specify which halfword of the source registers *Rn* and *Rm* is used as the first and second multiply operand.
If *X* is B, then the bottom halfword, bits [15:0] of *Rn* is used.
If *X* is T, then the top halfword, bits [31:16] of *Rn* is used. If *Y* is B, then the bottom halfword, bits [15:0], of *Rm* is used.
If *Y* is T, then the top halfword, bits [31:16], of *Rm* is used.

SMULW{Y} Signed Multiply (word by halfword).

Y specifies which halfword of the source register *Rm* is used as the second multiply operand.
If *Y* is B, then the bottom halfword (bits [15:0]) of *Rm* is used.
If *Y* is T, then the top halfword (bits [31:16]) of *Rm* is used.

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

Rd is the destination register.

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

Operation

The SMULBB, SMULTB, SMULBT and SMULTT instructions interprets the values from *Rn* and *Rm* as four signed 16-bit integers. These instructions:

- Multiplies the specified signed halfword, Top or Bottom, values from *Rn* and *Rm*.
- Writes the 32-bit result of the multiplication in *Rd.*

The SMULWT and SMULWB instructions interprets the values from *Rn* as a 32-bit signed integer and *Rm* as two halfword 16-bit signed integers. These instructions:

- Multiplies the first operand and the top, T suffix, or the bottom, B suffix, halfword of the second operand.
- Writes the signed most significant 32 bits of the 48-bit result in the destination register.

Restrictions

In these instructions:

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

Examples

```
SMULBT       R0, R4, R5  ; Multiplies the bottom halfword of R4 with the
                         ; top halfword of R5, multiplies results and
                         ; writes to R0
SMULBB       R0, R4, R5  ; Multiplies the bottom halfword of R4 with the
                         ; bottom halfword of R5, multiplies results and
                         ; writes to R0
SMULTT       R0, R4, R5  ; Multiplies the top halfword of R4 with the top
                         ; halfword of R5, multiplies results and writes
                         ; to R0
SMULTB       R0, R4, R5  ; Multiplies the top halfword of R4 with the
                         ; bottom halfword of R5, multiplies results and
                         ; and writes to R0
SMULWT       R4, R5, R3  ; Multiplies R5 with the top halfword of R3,
                         ; extracts top 32 bits and writes to R4
SMULWB       R4, R5, R3  ; Multiplies R5 with the bottom halfword of R3,
                         ; extracts top 32 bits and writes to R4.
```

12.6.6.11 UMULL, UMLAL, SMULL, and SMLAL

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

Syntax

```
op{cond} RdLo, RdHi, Rn, Rm
```

where:

op is one of:

UMULL Unsigned Long Multiply.
UMLAL Unsigned Long Multiply, with Accumulate.
SMULL Signed Long Multiply.
SMLAL Signed Long Multiply, with Accumulate.

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

RdHi, RdLo are the destination registers. For UMLAL and SMLAL they also hold the accumulating value.

Rn, Rm are registers holding the operands.

Operation

The UMULL instruction interprets the values from *Rn* and *Rm* as unsigned integers. It multiplies these integers and places the least significant 32 bits of the result in *RdLo*, and the most significant 32 bits of the result in *RdHi*.

The UMLAL instruction interprets the values from *Rn* and *Rm* as unsigned integers. It multiplies these integers, adds the 64-bit result to the 64-bit unsigned integer contained in *RdHi* and *RdLo*, and writes the result back to *RdHi* and *RdLo*.

The SMULL instruction interprets the values from *Rn* and *Rm* as two's complement signed integers. It multiplies these integers and places the least significant 32 bits of the result in *RdLo*, and the most significant 32 bits of the result in *RdHi*.

The SMLAL instruction interprets the values from *Rn* and *Rm* as two's complement signed integers. It multiplies these integers, adds the 64-bit result to the 64-bit signed integer contained in *RdHi* and *RdLo*, and writes the result back to *RdHi* and *RdLo*.

Restrictions

In these instructions:

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

Condition Flags

These instructions do not affect the condition code flags.

Examples

```
UMULL       R0, R4, R5, R6   ; Unsigned (R4,R0) = R5 x R6
SMLAL       R4, R5, R3, R8   ; Signed (R5,R4) = (R5,R4) + R3 x R8
```

12.6.6.12 SDIV and UDIV

Signed Divide and Unsigned Divide.

Syntax

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

where:

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

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

Rn is the register holding the value to be divided.

Rm is a register holding the divisor.

Operation

SDIV performs a signed integer division of the value in *Rn* by the value in *Rm*.

UDIV performs an unsigned integer division of the value in *Rn* by the value in *Rm*.

For both instructions, if the value in *Rn* is not divisible by the value in *Rm*, the result is rounded towards zero.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

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

12.6.7 Saturating Instructions

The table below shows the saturating instructions:

Table 12-22. Saturating Instructions

Mnemonic	Description
SSAT	Signed Saturate
SSAT16	Signed Saturate Halfword
USAT	Unsigned Saturate
USAT16	Unsigned Saturate Halfword
QADD	Saturating Add
QSUB	Saturating Subtract
QSUB16	Saturating Subtract 16
QASX	Saturating Add and Subtract with Exchange
QSAX	Saturating Subtract and Add with Exchange
QDADD	Saturating Double and Add
QDSUB	Saturating Double and Subtract
UQADD16	Unsigned Saturating Add 16
UQADD8	Unsigned Saturating Add 8
UQASX	Unsigned Saturating Add and Subtract with Exchange
UQSAX	Unsigned Saturating Subtract and Add with Exchange
UQSUB16	Unsigned Saturating Subtract 16
UQSUB8	Unsigned Saturating Subtract 8

For signed *n*-bit saturation, this means that:

- If the value to be saturated is less than -2^{n-1}, the result returned is -2^{n-1}.
- If the value to be saturated is greater than $2^{n-1}-1$, the result returned is $2^{n-1}-1$.
- Otherwise, the result returned is the same as the value to be saturated.

For unsigned *n*-bit saturation, this means that:

- If the value to be saturated is less than 0, the result returned is 0.
- If the value to be saturated is greater than $2^{n}-1$, the result returned is $2^{n}-1$.
- Otherwise, the result returned is the same as the value to be saturated.

If the returned result is different from the value to be saturated, it is called *saturation*. If saturation occurs, the instruction sets the Q flag to 1 in the APSR. Otherwise, it leaves the Q flag unchanged. To clear the Q flag to 0, the MSR instruction must be used; see “MSR”.

To read the state of the Q flag, the MRS instruction must be used; see “MRS”.

12.6.7.1 SSAT and USAT

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

Syntax

```
op{cond} Rd, #n, Rm {, shift #s}
```

where:

op	is one of:
	SSAT Saturates a signed value to a signed range.
	USAT Saturates a signed value to an unsigned range.
cond	is an optional condition code, see “Conditional Execution”.
Rd	is the destination register.
n	specifies the bit position to saturate to:
n ranges from 1 to 32 for SSAT	n ranges from 0 to 31 for USAT.
Rm	is the register containing the value to saturate.
shift #s	is an optional shift applied to Rm before saturating. It must be one of the following:
ASR #s	where s is in the range 1 to 31.
LSL #s	where *s* is in the range 0 to 31.

Operation

These instructions saturate to a signed or unsigned *n*-bit value.

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

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.
```

12.6.7.2 SSAT16 and USAT16

Signed Saturate and Unsigned Saturate to any bit position for two halfwords.

Syntax

```
op{cond} Rd, #n, Rm
```

where:

op	is one of:
	SSAT16 Saturates a signed halfword value to a signed range.
	USAT16 Saturates a signed halfword value to an unsigned range.
cond	is an optional condition code, see “Conditional Execution”.
Rd	is the destination register.
n	specifies the bit position to saturate to:
n ranges from 1 to 16 for SSAT	n ranges from 0 to 15 for USAT.
Rm	is the register containing the value to saturate.

Operation

The SSAT16 instruction:

Saturates two signed 16-bit halfword values of the register with the value to saturate from selected by the bit position in *n*.

Writes the results as two signed 16-bit halfwords to the destination register.

The USAT16 instruction:

Saturates two unsigned 16-bit halfword values of the register with the value to saturate from selected by the bit position in *n*.

Writes the results as two unsigned halfwords in the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

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

Examples

```
SSAT16    R7, #9, R2   ; Saturates the top and bottom highwords of R2
                       ; as 9-bit values, writes to corresponding halfword
                       ; of R7
USAT16NE  R0, #13, R5  ; Conditionally saturates the top and bottom
                       ; halfwords of R5 as 13-bit values, writes to
                       ; corresponding halfword of R0.
```

12.6.7.3 QADD and QSUB

Saturating Add and Saturating Subtract, signed.

Syntax

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

where:

op is one of:

QADD Saturating 32-bit add.

QADD8 Saturating four 8-bit integer additions.

QADD16 Saturating two 16-bit integer additions.

QSUB Saturating 32-bit subtraction.

QSUB8 Saturating four 8-bit integer subtraction.

QSUB16 Saturating two 16-bit integer subtraction.

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

Rd is the destination register.

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

Operation

These instructions add or subtract two, four or eight values from the first and second operands and then writes a signed saturated value in the destination register.

The QADD and QSUB instructions apply the specified add or subtract, and then saturate the result to the signed range $-2^{n-1} \leq x \leq 2^{n-1}-1$, where x is given by the number of bits applied in the instruction, 32, 16 or 8.

If the returned result is different from the value to be saturated, it is called *saturation*. If saturation occurs, the QADD and QSUB instructions set the Q flag to 1 in the APSR. Otherwise, it leaves the Q flag unchanged. The 8-bit and 16-bit QADD and QSUB instructions always leave the Q flag unchanged.

To clear the Q flag to 0, the MSR instruction must be used; see “MSR”.

To read the state of the Q flag, the MRS instruction must be used; see “MRS”.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

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

Examples

```
QADD16   R7, R4, R2  ; Adds halfwords of R4 with corresponding halfword of
                     ; R2, saturates to 16 bits and writes to
                     ; corresponding halfword of R7
QADD8    R3, R1, R6  ; Adds bytes of R1 to the corresponding bytes of R6,
                     ; saturates to 8 bits and writes to corresponding
                     ; byte of R3
QSUB16   R4, R2, R3  ; Subtracts halfwords of R3 from corresponding
                     ; halfword of R2, saturates to 16 bits, writes to
                     ; corresponding halfword of R4
QSUB8    R4, R2, R5  ; Subtracts bytes of R5 from the corresponding byte
                     ; in R2, saturates to 8 bits, writes to corresponding
                     ; byte of R4.
```

12.6.7.4 QASX and QSAX

Saturating Add and Subtract with Exchange and Saturating Subtract and Add with Exchange, signed.

Syntax

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

where:

op is one of:

QASX Add and Subtract with Exchange and Saturate.

QSAX Subtract and Add with Exchange and Saturate.

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

Rd is the destination register.

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

Operation

The QASX instruction:

1. Adds the top halfword of the source operand with the bottom halfword of the second operand.
2. Subtracts the top halfword of the second operand from the bottom highword of the first operand.
3. Saturates the result of the subtraction and writes a 16-bit signed integer in the range $-2^{15} \le x \le 2^{15} - 1$, 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 $-2^{15} \le x \le 2^{15} - 1$, 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 $-2^{15} \le x \le 2^{15} - 1$, 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 $-2^{15} \le x \le 2^{15} - 1$, where x equals 16, to the top halfword of the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

Examples

```
QASX   R7, R4, R2 ; Adds top halfword of R4 to bottom halfword of R2,
                  ; saturates to 16 bits, writes to top halfword of R7
                  ; Subtracts top highword of R2 from bottom halfword of
                  ; R4, saturates to 16 bits and writes to bottom halfword
                  ; of R7
QSAX   R0, R3, R5 ; Subtracts bottom halfword of R5 from top halfword of
                  ; R3, saturates to 16 bits, writes to top halfword of R0
                  ; Adds bottom halfword of R3 to top halfword of R5,
                  ; saturates to 16 bits, writes to bottom halfword of R0.
```

12.6.7.5 QDADD and QDSUB

Saturating Double and Add and Saturating Double and Subtract, signed.

Syntax

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

where:

op is one of:

QDADD Saturating Double and Add.

QDSUB Saturating Double and Subtract.

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

Rd is the destination register.

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

Operation

The QDADD instruction:

- Doubles the second operand value.
- Adds the result of the doubling to the signed saturated value in the first operand.
- Writes the result to the destination register.

The QDSUB instruction:

- Doubles the second operand value.
- Subtracts the doubled value from the signed saturated value in the first operand.
- Writes the result to the destination register.

Both the doubling and the addition or subtraction have their results saturated to the 32-bit signed integer range $-2^{31} \le x \le 2^{31} - 1$. If saturation occurs in either operation, it sets the Q flag in the APSR.

Restrictions

Do not use SP and do not use PC.

Condition Flags

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

Examples

```
QDADD     R7, R4, R2    ; Doubles and saturates R4 to 32 bits, adds R2,
                        ; saturates to 32 bits, writes to R7
QDSUB     R0, R3, R5    ; Subtracts R3 doubled and saturated to 32 bits
                        ; from R5, saturates to 32 bits, writes to R0.
```

12.6.7.6 UQASX and UQSAX

Saturating Add and Subtract with Exchange and Saturating Subtract and Add with Exchange, unsigned.

Syntax

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

where:

type is one of:

UQASX Add and Subtract with Exchange and Saturate.

UQSAX Subtract and Add with Exchange and Saturate.

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

Rd is the destination register.

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

Operation

The UQASX instruction:

1. Adds the bottom halfword of the source operand with the top halfword of the second operand.
2. Subtracts the bottom halfword of the second operand from the top highword of the first operand.
3. Saturates the results of the sum and writes a 16-bit unsigned integer in the range $0 \le x \le 2^{16} - 1$, 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.
```

12.6.7.7 UQADD and UQSUB

Saturating Add and Saturating Subtract Unsigned.

Syntax

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

where:

op is one of:

UQADD8 Saturating four unsigned 8-bit integer additions.

UQADD16 Saturating two unsigned 16-bit integer additions.

UDSUB8 Saturating four unsigned 8-bit integer subtractions.

UQSUB16 Saturating two unsigned 16-bit integer subtractions.

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

Rd is the destination register.

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

Operation

These instructions add or subtract two or four values and then writes an unsigned saturated value in the destination register.

The UQADD16 instruction:

- Adds the respective top and bottom halfwords of the first and second operands.
- Saturates the result of the additions for each halfword in the destination register to the unsigned range $0 \le x \le 2^{16}-1$, 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.
```

12.6.8 Packing and Unpacking Instructions

The table below shows the instructions that operate on packing and unpacking data.

Table 12-23. Packing and Unpacking Instructions

Mnemonic	Description
PKH	Pack Halfword
SXTAB	Extend 8 bits to 32 and add
SXTAB16	Dual extend 8 bits to 16 and add
SXTAH	Extend 16 bits to 32 and add
SXTB	Sign extend a byte
SXTB16	Dual extend 8 bits to 16 and add
SXTH	Sign extend a halfword
UXTAB	Extend 8 bits to 32 and add
UXTAB16	Dual extend 8 bits to 16 and add
UXTAH	Extend 16 bits to 32 and add
UXTB	Zero extend a byte
UXTB16	Dual zero extend 8 bits to 16 and add
UXTH	Zero extend a halfword

12.6.8.1 PKHBT and PKHTB

Pack Halfword

Syntax

```
op{cond} {Rd}, Rn, Rm {, LSL #imm}
op{cond} {Rd}, Rn, Rm {, ASR #imm}
```

where:

op — is one of:

PKHBT Pack Halfword, bottom and top with shift.

PKHTB Pack Halfword, top and bottom with shift.

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

Rd — is the destination register.

Rn — is the first operand register.

Rm — is the second operand register holding the value to be optionally shifted.

imm — is the shift length. The type of shift length depends on the instruction:
For PKHBT
LSL a left shift with a shift length from 1 to 31, 0 means no shift.
For PKHTB
ASR an arithmetic shift right with a shift length from 1 to 32, a shift of 32-bits is encoded as 0b00000.

Operation

The PKHBT instruction:

1. Writes the value of the bottom halfword of the first operand to the bottom halfword of the destination register.
2. If shifted, the shifted value of the second operand is written to the top halfword of the destination register.

The PKHTB instruction:

1. Writes the value of the top halfword of the first operand to the top halfword of the destination register.
2. If shifted, the shifted value of the second operand is written to the bottom halfword of the destination register.

Restrictions

Rd must not be SP and must not be PC.

Condition Flags

This instruction does not change the flags.

Examples

```
PKHBT  R3, R4, R5 LSL #0 ; Writes bottom halfword of R4 to bottom halfword of
                         ; R3, writes top halfword of R5, unshifted, to top
                         ; halfword of R3
PKHTB  R4, R0, R2 ASR #1 ; Writes R2 shifted right by 1 bit to bottom halfword
                         ; of R4, and writes top halfword of R0 to top
                         ; halfword of R4.
```

12.6.8.2 SXT and UXT

Sign extend and Zero extend.

Syntax

```
op{cond} {Rd,} Rm {, ROR #n}
op{cond} {Rd}, Rm {, ROR #n}
```

where:

op — is one of:

SXTB Sign extends an 8-bit value to a 32-bit value.

SXTH Sign extends a 16-bit value to a 32-bit value.

SXTB16 Sign extends two 8-bit values to two 16-bit values.

UXTB Zero extends an 8-bit value to a 32-bit value.

UXTH Zero extends a 16-bit value to a 32-bit value.

UXTB16 Zero extends two 8-bit values to two 16-bit values.

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

Rd — is the destination register.

Rm — is the register holding the value to extend.

ROR #*n* — is one of:

ROR #8 Value from *Rm* is rotated right 8 bits.

Operation

These instructions do the following:

1. Rotate the value from *Rm* right by 0, 8, 16 or 24 bits.
2. Extract bits from the resulting value:
   - SXTB extracts bits[7:0] and sign extends to 32 bits.
   - UXTB extracts bits[7:0] and zero extends to 32 bits.
   - SXTH extracts bits[15:0] and sign extends to 32 bits.
   - UXTH extracts bits[15:0] and zero extends to 32 bits.
   - SXTB16 extracts bits[7:0] and sign extends to 16 bits, and extracts bits [23:16] and sign extends to 16 bits.
   - UXTB16 extracts bits[7:0] and zero extends to 16 bits, and extracts bits [23:16] and zero extends to 16 bits.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the flags.

Examples

```
SXTH  R4, R6, ROR #16  ; Rotates R6 right by 16 bits, obtains bottom halfword of
                       ; of result, sign extends to 32 bits and writes to R4
UXTB  R3, R10          ; Extracts lowest byte of value in R10, zero extends, and
                       ; writes to R3.
```

12.6.8.3 SXTA and UXTA

Signed and Unsigned Extend and Add

Syntax

```
op{cond} {Rd,} Rn, Rm {, ROR #n}
op{cond} {Rd,} Rn, Rm {, ROR #n}
```

where:

op — is one of:

SXTAB Sign extends an 8-bit value to a 32-bit value and add.

SXTAH Sign extends a 16-bit value to a 32-bit value and add.

SXTAB16 Sign extends two 8-bit values to two 16-bit values and add.

UXTAB Zero extends an 8-bit value to a 32-bit value and add.

UXTAH Zero extends a 16-bit value to a 32-bit value and add.

UXTAB16 Zero extends two 8-bit values to two 16-bit values and add.

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

Rd — is the destination register.

Rn — is the first operand register.

Rm — is the register holding the value to rotate and extend.

ROR #*n* — is one of:

ROR #8 Value from *Rm* is rotated right 8 bits.

ROR #16 Value from *Rm* is rotated right 16 bits.

ROR #24 Value from *Rm* is rotated right 24 bits.

If ROR #*n* is omitted, no rotation is performed.

Operation

These instructions do the following:

1. Rotate the value from *Rm* right by 0, 8, 16 or 24 bits.
2. Extract bits from the resulting value:
   - SXTAB extracts bits[7:0] from *Rm* and sign extends to 32 bits.
   - UXTAB extracts bits[7:0] from *Rm* and zero extends to 32 bits.
   - SXTAH extracts bits[15:0] from *Rm* and sign extends to 32 bits.
   - UXTAH extracts bits[15:0] from *Rm* and zero extends to 32 bits.
   - SXTAB16 extracts bits[7:0] from *Rm* and sign extends to 16 bits, and extracts bits [23:16] from *Rm* and sign extends to 16 bits.
   - UXTAB16 extracts bits[7:0] from *Rm* and zero extends to 16 bits, and extracts bits [23:16] from *Rm* and zero extends to 16 bits.
3. Adds the signed or zero extended value to the word or corresponding halfword of *Rn* and writes the result in *Rd*.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the flags.

Examples

```
SXTAH  R4, R8, R6, ROR #16 ; Rotates R6 right by 16 bits, obtains bottom
                           ; halfword, sign extends to 32 bits, adds
                           ; R8,and writes to R4
UXTAB  R3, R4, R10         ; Extracts bottom byte of R10 and zero extends
                           ; to 32 bits, adds R4, and writes to R3.
```

12.6.9 Bitfield Instructions

The table below shows the instructions that operate on adjacent sets of bits in registers or bitfields.

Table 12-24. Packing and Unpacking Instructions

Mnemonic	Description
BFC	Bit Field Clear
BFI	Bit Field Insert
SBFX	Signed Bit Field Extract
SXTB	Sign extend a byte
SXTH	Sign extend a halfword
UBFX	Unsigned Bit Field Extract
UXTB	Zero extend a byte
UXTH	Zero extend a halfword

12.6.9.1 BFC and BFI

Bit Field Clear and Bit Field Insert.

Syntax

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

where:

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

Rd — is the destination register.

Rn — is the source register.

lsb — is the position of the least significant bit of the bitfield. *lsb* must be in the range 0 to 31.

width — is the width of the bitfield and must be in the range 1 to 32-*lsb*.

Operation

BFC clears a bitfield in a register. It clears *width* bits in *Rd*, starting at the low bit position *lsb*. Other bits in *Rd* are unchanged.

BFI copies a bitfield into one register from another register. It replaces *width* bits in *Rd* starting at the low bit position *lsb*, with *width* bits from *Rn* starting at bit[0]. Other bits in *Rd* are unchanged.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the flags.

Examples

```
BFC   R4, #8, #12       ; Clear bit 8 to bit 19 (12 bits) of R4 to 0
BFI   R9, R2, #8, #12   ; Replace bit 8 to bit 19 (12 bits) of R9 with
                        ; bit 0 to bit 11 from R2.
```

12.6.9.2 SBFX and UBFX

Signed Bit Field Extract and Unsigned Bit Field Extract.

Syntax

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

where:

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

Rd is the destination register.

Rn is the source register.

lsb is the position of the least significant bit of the bitfield. *lsb* must be in the range 0 to 31.

width is the width of the bitfield and must be in the range 1 to 32-*lsb*.

Operation

SBFX extracts a bitfield from one register, sign extends it to 32 bits, and writes the result to the destination register.

UBFX extracts a bitfield from one register, zero extends it to 32 bits, and writes the result to the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the flags.

Examples

```
SBFX  R0, R1, #20, #4  ; Extract bit 20 to bit 23 (4 bits) from R1 and sign
                       ; extend to 32 bits and then write the result to R0.
UBFX  R8, R11, #9, #10 ; Extract bit 9 to bit 18 (10 bits) from R11 and zero
                       ; extend to 32 bits and then write the result to R8.
```

12.6.9.3 SXT and UXT

Sign extend and Zero extend.

Syntax

```
SXTextend{cond} {Rd,} Rm {, ROR #n}
UXTextend{cond} {Rd}, Rm {, ROR #n}
```

where:

extend is one of:

B Extends an 8-bit value to a 32-bit value.

H Extends a 16-bit value to a 32-bit value.

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

Rd is the destination register.

Rm is the register holding the value to extend.

ROR #*n* is one of:

ROR #8 Value from *Rm* is rotated right 8 bits.

ROR #16 Value from *Rm* is rotated right 16 bits.

ROR #24 Value from *Rm* is rotated right 24 bits.

If ROR #*n* is omitted, no rotation is performed.

Operation

These instructions do the following:

1. Rotate the value from *Rm* right by 0, 8, 16 or 24 bits.
2. Extract bits from the resulting value:
   - SXTB extracts bits[7:0] and sign extends to 32 bits.
   - UXTB extracts bits[7:0] and zero extends to 32 bits.
   - SXTH extracts bits[15:0] and sign extends to 32 bits.
   - UXTH extracts bits[15:0] and zero extends to 32 bits.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the flags.

Examples

```
SXTH  R4, R6, ROR #16  ; Rotate R6 right by 16 bits, then obtain the lower
                       ; halfword of the result and then sign extend to
                       ; 32 bits and write the result to R4.
UXTB  R3, R10          ; Extract lowest byte of the value in R10 and zero
                       ; extend it, and write the result to R3.
```

12.6.10 Branch and Control Instructions

The table below shows the branch and control instructions.

Table 12-25. Branch and Control Instructions

Mnemonic	Description
B	Branch
BL	Branch with Link
BLX	Branch indirect with Link
BX	Branch indirect
CBNZ	Compare and Branch if Non Zero
CBZ	Compare and Branch if Zero
IT	If-Then
TBB	Table Branch Byte
TBH	Table Branch Halfword

12.6.10.1 B, BL, BX, and BLX

Branch instructions.

Syntax

```
B{cond} label
BL{cond} label
BX{cond} Rm
BLX{cond} Rm
```

where:

B	is branch (immediate).
BL	is branch with link (immediate).
BX	is branch indirect (register).
BLX	is branch indirect with link (register).
cond	is an optional condition code, see "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 12-26. Branch Ranges

Instruction	Branch Range
B label	-16 MB to +16 MB
Bcond label (outside IT block)	-1 MB to +1 MB
Bcond 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
```

12.6.10.2 CBZ and CBNZ

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

Syntax

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

where:

Rn is the register holding the operand.

label is the branch destination.

Operation

Use the CBZ or CBNZ instructions to avoid changing the condition code flags and to reduce the number of instructions.

CBZ Rn, label does not change condition flags but is otherwise equivalent to:

```
CMP     Rn, #0
BEQ     label
```

CBNZ Rn, label does not change condition flags but is otherwise equivalent to:

```
CMP     Rn, #0
BNE     label
```

Restrictions

The restrictions are:

- *Rn* must be in the range of R0 to R7.
- The branch destination must be within 4 to 130 bytes after the instruction.
- These instructions must not be used inside an IT block.

Condition Flags

These instructions do not change the flags.

Examples

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

12.6.10.3 IT

If-Then condition instruction.

Syntax

```
IT{x{y{z}}} cond
```

where:

x specifies the condition switch for the second instruction in the IT block.

y specifies the condition switch for the third instruction in the IT block.

z specifies the condition switch for the fourth instruction in the IT block.

cond specifies the condition for the first instruction in the IT block.

The condition switch for the second, third and fourth instruction in the IT block can be either:

T Then. Applies the condition *cond* to the instruction.

E Else. Applies the inverse condition of *cond* to the instruction.

It is possible to use AL (the *always* condition) for *cond* in an IT instruction. If this is done, all of the instructions in the IT block must be unconditional, and each of *x*, *y*, and *z* must be T or omitted but not E.

Operation

The IT instruction makes up to four following instructions conditional. The conditions can be all the same, or some of them can be the logical inverse of the others. The conditional instructions following the IT instruction form the *IT block*.

The instructions in the IT block, including any branches, must specify the condition in the {*cond*} part of their syntax.

The assembler might be able to generate the required IT instructions for conditional instructions automatically, so that the user does not have to write them. See the assembler documentation for details.

A BKPT instruction in an IT block is always executed, even if its condition fails.

Exceptions can be taken between an IT instruction and the corresponding IT block, or within an IT block. Such an exception results in entry to the appropriate exception handler, with suitable return information in LR and stacked PSR.

Instructions designed for use for exception returns can be used as normal to return from the exception, and execution of the IT block resumes correctly. This is the only way that a PC-modifying instruction is permitted to branch to an instruction in an IT block.

Restrictions

The following instructions are not permitted in an IT block:

- IT.
- CBZ and CBNZ.
- CPSID and CPSIE.

Other restrictions when using an IT block are:

- A branch or any instruction that modifies the PC must either be outside an IT block or must be the last instruction inside the IT block. These are:
  - ADD PC, PC, Rm.
  - MOV PC, Rm.
  - B, BL, BX, BLX.
  - Any LDM, LDR, or POP instruction that writes to the PC.
  - TBB and TBH.
- Do not branch to any instruction inside an IT block, except when returning from an exception handler.
- All conditional instructions except B*cond* must be inside an IT block. B*cond* can be either outside or inside an IT block but has a larger branch range if it is inside one.
- Each instruction inside the IT block must specify a condition code suffix that is either the same or logical inverse as for the other instructions in the block.

Your assembler might place extra restrictions on the use of IT blocks, such as prohibiting the use of assembler directives within them.

Condition Flags

This instruction does not change the flags.

Example

```
ITTE   NE           ; Next 3 instructions are conditional
ANDNE  R0, R0, R1   ; ANDNE does not update condition flags
ADDSNE R2, R2, #1   ; ADDSNE updates condition flags
MOVEQ  R2, R3       ; Conditional move


CMP    R0, #9       ; Convert R0 hex value (0 to 15) into ASCII
                    ; ('0'-'9', 'A'-'F')
ITE    GT           ; Next 2 instructions are conditional
ADDGT  R1, R0, #55  ; Convert 0xA -> 'A'
ADDLE  R1, R0, #48  ; Convert 0x0 -> '0'


IT     GT           ; IT block with only one conditional instruction
ADDGT  R1, R1, #1   ; Increment R1 conditionally


ITTEE  EQ           ; Next 4 instructions are conditional
MOVEQ  R0, R1       ; Conditional move
ADDEQ  R2, R2, #10  ; Conditional add
ANDNE  R3, R3, #1   ; Conditional AND
BNE.W  dloop        ; Branch instruction can only be used in the last
                    ; instruction of an IT block


IT     NE           ; Next instruction is conditional
ADD    R0, R0, R1   ; Syntax error: no condition code used in IT block
```

12.6.10.4 TBB and TBH

Table Branch Byte and Table Branch Halfword.

Syntax

```
TBB [Rn, Rm]
TBH [Rn, Rm, LSL #1]
```

where:

Rn is the register containing the address of the table of branch lengths.

If *Rn* is PC, then the address of the table is the address of the byte immediately following the TBB or TBH instruction.

Rm is the index register. This contains an index into the table. For halfword tables, LSL #1 doubles the value in *Rm* to form the right offset into the table.

Operation

These instructions cause a PC-relative forward branch using a table of single byte offsets for TBB, or halfword offsets for TBH. *Rn* provides a pointer to the table, and *Rm* supplies an index into the table. For TBB the branch offset is twice the unsigned value of the byte returned from the table. and for TBH the branch offset is twice the unsigned value of the halfword returned from the table. The branch occurs to the address at that offset from the address of the byte immediately after the TBB or TBH instruction.

Restrictions

The restrictions are:

- *Rn* must not be SP.
- *Rm* must not be SP and must not be PC.
- When any of these instructions is used inside an IT block, it must be the last instruction of the IT block.

Condition Flags

These instructions do not change the flags.

Examples

```
        ADR.W  R0, BranchTable_Byte
        TBB    [R0, R1]             ; R1 is the index, R0 is the base address of the
                                    ; branch table
Case1
; an instruction sequence follows
Case2
; an instruction sequence follows
Case3
; an instruction sequence follows
BranchTable_Byte
        DCB    0                    ; Case1 offset calculation
        DCB    ((Case2-Case1)/2)    ; Case2 offset calculation
        DCB    ((Case3-Case1)/2)    ; Case3 offset calculation


        TBH    [PC, R1, LSL #1]     ; R1 is the index, PC is used as base of the
                                    ; branch table
BranchTable_H
        DCI    ((CaseA - BranchTable_H)/2)  ; CaseA offset calculation
        DCI    ((CaseB - BranchTable_H)/2)  ; CaseB offset calculation
        DCI    ((CaseC - BranchTable_H)/2)  ; CaseC offset calculation

CaseA
; an instruction sequence follows
CaseB
; an instruction sequence follows
CaseC
; an instruction sequence follows
```

12.6.11 Floating-point Instructions

The table below shows the floating-point instructions.

These instructions are only available if the FPU is included, and enabled, in the system. See “Enabling the FPU” for information about enabling the floating-point unit.

Table 12-27. Floating-point Instructions

Mnemonic	Description
VABS	Floating-point Absolute
VADD	Floating-point Add
VCMP	Compare two floating-point registers, or one floating-point register and zero
VCMPE	Compare two floating-point registers, or one floating-point register and zero with Invalid Operation check
VCVT	Convert between floating-point and integer
VCVT	Convert between floating-point and fixed point
VCVTR	Convert between floating-point and integer with rounding
VCVTB	Converts half-precision value to single-precision
VCVTT	Converts single-precision register to half-precision
VDIV	Floating-point Divide
VFMA	Floating-point Fused Multiply Accumulate
VFNMA	Floating-point Fused Negate Multiply Accumulate
VFMS	Floating-point Fused Multiply Subtract
VFNMS	Floating-point Fused Negate Multiply Subtract
VLDM	Load Multiple extension registers
VLDR	Loads an extension register from memory
VLMA	Floating-point Multiply Accumulate
VLMS	Floating-point Multiply Subtract
VMOV	Floating-point Move Immediate
VMOV	Floating-point Move Register
VMOV	Copy ARM core register to single precision
VMOV	Copy 2 ARM core registers to 2 single precision
VMOV	Copies between ARM core register to scalar
VMOV	Copies between Scalar to ARM core register
VMRS	Move to ARM core register from floating-point System Register
VMSR	Move to floating-point System Register from ARM Core register
VMUL	Multiply floating-point
VNEG	Floating-point negate
VNMLA	Floating-point multiply and add
VNMLS	Floating-point multiply and subtract
VNMUL	Floating-point multiply
VPOP	Pop extension registers
VPUSH	Push extension registers
VSQRT	Floating-point square root
VSTM	Store Multiple extension registers
VSTR	Stores an extension register to memory
VSUB	Floating-point Subtract

12.6.11.1 VABS

Floating-point Absolute.

Syntax

```
VABS{cond}.F32 Sd, Sm
```

where:

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

Sd, Sm are the destination floating-point value and the operand floating-point value.

Operation

This instruction:

1. Takes the absolute value of the operand floating-point register.
2. Places the results in the destination floating-point register.

Restrictions

There are no restrictions.

Condition Flags

The floating-point instruction clears the sign bit.

Examples

```
VABS.F32 S4, S6
```

12.6.11.2 VADD

Floating-point Add

Syntax

```
VADD{cond}.F32 {Sd,} Sn, Sm
```

where:

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

Sd is the destination floating-point value.

Sn, Sm are the operand floating-point values.

Operation

This instruction:

1. Adds the values in the two floating-point operand registers.
2. Places the results in the destination floating-point register.

Restrictions

There are no restrictions.

Condition Flags

This instruction does not change the flags.

Examples

```
VADD.F32 S4, S6, S7
```

12.6.11.3 VCMP, VCMPE

Compares two floating-point registers, or one floating-point register and zero.

Syntax

```
VCMP{E}{cond}.F32 Sd, Sm
VCMP{E}{cond}.F32 Sd, #0.0
```

where:

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

E — If present, any NaN operand causes an Invalid Operation exception. Otherwise, only a signaling NaN causes the exception.

Sd — is the floating-point operand to compare.

Sm — is the floating-point operand that is compared with.

Operation

This instruction:

1. Compares:
   - Two floating-point registers.
   - One floating-point register and zero.
2. Writes the result to the FPSCR flags.

Restrictions

This instruction can optionally raise an Invalid Operation exception if either operand is any type of NaN. It always raises an Invalid Operation exception if either operand is a signaling NaN.

Condition Flags

When this instruction writes the result to the FPSCR flags, the values are normally transferred to the ARM flags by a subsequent VMRS instruction. See “VMRS”.

Examples

```
VCMP.F32    S4, #0.0
VCMP.F32    S4, S2
```

12.6.11.4 VCVT, VCVTR between Floating-point and Integer

Converts a value in a register from floating-point to a 32-bit integer.

Syntax

```
VCVT{R}{cond}.Tm.F32 Sd, Sm
VCVT{cond}.F32.Tm Sd, Sm
```

where:

R — If *R* is specified, the operation uses the rounding mode specified by the FPSCR. If *R* is omitted, the operation uses the Round towards Zero rounding mode.

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

Tm — is the data type for the operand. It must be one of:

S32 signed 32- — U32 unsigned 32-bit value.

Sd, Sm — are the destination register and the operand register.

Operation

These instructions:

1. Either
   - Convert a value in a register from floating-point value to a 32-bit integer.
   - Convert from a 32-bit integer to floating-point value.
2. Place the result in a second register.

The floating-point to integer operation normally uses the *Round towards Zero* rounding mode, but can optionally use the rounding mode specified by the FPSCR.

The integer to floating-point operation uses the rounding mode specified by the FPSCR.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

12.6.11.5 VCVT between Floating-point and Fixed-point

Converts a value in a register from floating-point to and from fixed-point.

Syntax

```
VCVT{cond}.Td.F32 Sd, Sd, #fbits
VCVT{cond}.F32.Td Sd, Sd, #fbits
```

where:

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

Td — is the data type for the fixed-point number. It must be one of:

S16 signed 16-bit value.
U16 unsigned 16-bit value.

S32 signed 32-bit value.
U32 unsigned 32-bit value.

Sd — is the destination register and the operand register.

fbits — is the number of fraction bits in the fixed-point number:

If *Td* is S16 or U16, *fbits* must be in the range 0 - 16.
If *Td* is S32 or U32, *fbits* must be in the range 1 - 32.

Operation

These instructions:

1. Either
   - Converts a value in a register from floating-point to fixed-point.
   - Converts a value in a register from fixed-point to floating-point.
2. Places the result in a second register.

The floating-point values are single-precision.

The fixed-point value can be 16-bit or 32-bit. Conversions from fixed-point values take their operand from the low-order bits of the source register and ignore any remaining bits.

Signed conversions to fixed-point values sign-extend the result value to the destination register width.

Unsigned conversions to fixed-point values zero-extend the result value to the destination register width.

The floating-point to fixed-point operation uses the *Round towards Zero* rounding mode. The fixed-point to floating-point operation uses the *Round to Nearest* rounding mode.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

12.6.11.6 VCVTB, VCVTT

Converts between a half-precision value and a single-precision value.

Syntax

```
VCVT{y}{cond}.F32.F16 Sd, Sm
VCVT{y}{cond}.F16.F32 Sd, Sm
```

where:

y — Specifies which half of the operand register *Sm* or destination register *Sd* is used for the operand or destination:

- If y is B, then the bottom half, bits [15:0], of *Sm* or *Sd* is used.
- If *y* is T, then the top half, bits [31:16], of *Sm* or *Sd* is used.

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

Sd is the destination register.

Sm is the operand register.

Operation

This instruction with the .F16.32 suffix:

1. Converts the half-precision value in the top or bottom half of a single-precision. register to single-precision.
2. Writes the result to a single-precision register.

This instruction with the .F32.F16 suffix:

1. Converts the value in a single-precision register to half-precision.
2. Writes the result into the top or bottom half of a single-precision register, preserving the other half of the target register.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

12.6.11.7 VDIV

Divides floating-point values.

Syntax

```
VDIV{cond}.F32 {Sd,} Sn, Sm
```

where:

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

Sd is the destination register.

Sn, Sm are the operand registers.

Operation

This instruction:

1. Divides one floating-point value by another floating-point value.
2. Writes the result to the floating-point destination register.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

12.6.11.8 VFMA, VFMS

Floating-point Fused Multiply Accumulate and Subtract.

Syntax

```
VFMA{cond}.F32 {Sd,} Sn, Sm
VFMS{cond}.F32 {Sd,} Sn, Sm
```

where:

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

Sd is the destination register.

Sn, Sm are the operand registers.

Operation

The VFMA instruction:

1. Multiplies the floating-point values in the operand registers.
2. Accumulates the results into the destination register.

The result of the multiply is not rounded before the accumulation.

The VFMS instruction:

1. Negates the first operand register.
2. Multiplies the floating-point values of the first and second operand registers.
3. Adds the products to the destination register.
4. Places the results in the destination register.

The result of the multiply is not rounded before the addition.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

12.6.11.9 VFNMA, VFNMS

Floating-point Fused Negate Multiply Accumulate and Subtract.

Syntax

```
VFNMA{cond}.F32 {Sd,} Sn, Sm
VFNMS{cond}.F32 {Sd,} Sn, Sm
```

where:

cond	is an optional condition code, see “Conditional Execution”.
Sd	is the destination register.
Sn, Sm	are the operand registers.

Operation

The VFNMA instruction:

1. Negates the first floating-point operand register.
2. Multiplies the first floating-point operand with second floating-point operand.
3. Adds the negation of the floating-point destination register to the product.
4. Places the result into the destination register.

The result of the multiply is not rounded before the addition.

The VFNMS instruction:

1. Multiplies the first floating-point operand with second floating-point operand.
2. Adds the negation of the floating-point value in the destination register to the product.
3. Places the result in the destination register.

The result of the multiply is not rounded before the addition.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

12.6.11.10 VLDM

Floating-point Load Multiple

Syntax

```
VLDM{mode}{cond}{.size} Rn{!}, list
```

where:

mode — is the addressing mode:
- *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 single-word registers, separated by commas and surrounded by brackets.

Operation

This instruction loads:

- Multiple extension registers from consecutive memory locations using an address from an ARM core register as the base address.

Restrictions

The restrictions are:

- If *size* is present, it must be equal to the size in bits, 32 or 64, of the registers in *list*.
- For the base address, the SP can be used.
  In the ARM instruction set, if *!* is not specified the PC can be used.
- List must contain at least one register. If it contains doubleword registers, it must not contain more than 16 registers.
- If using the *Decrement Before addressing* mode, the write back flag, *!*, must be appended to the base register specification.

Condition Flags

These instructions do not change the flags.

12.6.11.11 VLDR

Loads a single extension register from memory

Syntax

```
VLDR{cond}{.64} Dd, [Rn{#imm}]
VLDR{cond}{.64} Dd, label
VLDR{cond}{.64} Dd, [PC, #imm}]
VLDR{cond}{.32} Sd, [Rn {, #imm}]
VLDR{cond}{.32} Sd, label
VLDR{cond}{.32} Sd, [PC, #imm]
```

where:

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

64, 32 — are the optional data size specifiers.

Dd — is the destination register for a doubleword load.

Sd — is the destination register for a singleword load.

Rn — is the base register. The SP can be used.

imm is the + or - immediate offset used to form the address. Permitted address values are multiples of 4 in the range 0 to 1020.

label is the label of the literal data item to be loaded.

Operation

This instruction:

- Loads a single extension register from memory, using a base address from an ARM core register, with an optional offset.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

12.6.11.12 VLMA, VLMS

Multiplies two floating-point values, and accumulates or subtracts the results.

Syntax

```
VLMA{cond}.F32 Sd, Sn, Sm
VLMS{cond}.F32 Sd, Sn, Sm
```

where:

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

Sd is the destination floating-point value.

Sn, Sm are the operand floating-point values.

Operation

The floating-point Multiply Accumulate instruction:

1. Multiplies two floating-point values.
2. Adds the results to the destination floating-point value.

The floating-point Multiply Subtract instruction:

1. Multiplies two floating-point values.
2. Subtracts the products from the destination floating-point value.
3. Places the results in the destination register.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

12.6.11.13 VMOV Immediate

Move floating-point Immediate

Syntax

```
VMOV{cond}.F32 Sd, #imm
```

where:

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

Sd is the branch destination.

imm is a floating-point constant.

Operation

This instruction copies a constant value to a floating-point register.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

12.6.11.14 VMOV Register

Copies the contents of one register to another.

Syntax

```
VMOV{cond}.F64 Dd, Dm
VMOV{cond}.F32 Sd, Sm
```

where:

cond	is an optional condition code, see “Conditional Execution”.
Dd	is the destination register, for a doubleword operation.
Dm	is the source register, for a doubleword operation.
Sd	is the destination register, for a singleword operation.
Sm	is the source register, for a singleword operation.

Operation

This instruction copies the contents of one floating-point register to another.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

12.6.11.15 VMOV Scalar to ARM Core Register

Transfers one word of a doubleword floating-point register to an ARM core register.

Syntax

```
VMOV{cond} Rt, Dn[x]
```

where:

cond	is an optional condition code, see “Conditional Execution”.
Rt	is the destination ARM core register.
Dn	is the 64-bit doubleword register.
x	Specifies which half of the doubleword register to use: - If *x* is 0, use lower half of doubleword register. - If *x* is 1, use upper half of doubleword register.

Operation

This instruction transfers:

- One word from the upper or lower half of a doubleword floating-point register to an ARM core register.

Restrictions

Rt cannot be PC or SP.

Condition Flags

These instructions do not change the flags.

12.6.11.16 VMOV ARM Core Register to Single Precision

Transfers a single-precision register to and from an ARM core register.

Syntax

```
VMOV{cond} Sn, Rt
VMOV{cond} Rt, Sn
```

where:

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

Sn — is the single-precision floating-point register.

Rt — is the ARM core register.

Operation

This instruction transfers:

- The contents of a single-precision register to an ARM core register.
- The contents of an ARM core register to a single-precision register.

Restrictions

Rt cannot be PC or SP.

Condition Flags

These instructions do not change the flags.

12.6.11.17 VMOV Two ARM Core Registers to Two Single Precision

Transfers two consecutively numbered single-precision registers to and from two ARM core registers.

Syntax

```
VMOV{cond} Sm, Sm1, Rt, Rt2
VMOV{cond} Rt, Rt2, Sm, Sm
```

where:

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

Sm — is the first single-precision register.

Sm1 — is the second single-precision register.
This is the next single-precision register after *Sm*.

Rt — is the ARM core register that *Sm* is transferred to or from.

Rt2 — is the The ARM core register that *Sm1* is transferred to or from.

Operation

This instruction transfers:

- The contents of two consecutively numbered single-precision registers to two ARM core registers.
- The contents of two ARM core registers to a pair of single-precision registers.

Restrictions

The restrictions are:

- The floating-point registers must be contiguous, one after the other.
- The ARM core registers do not have to be contiguous.
- *Rt* cannot be PC or SP.

Condition Flags

These instructions do not change the flags.

12.6.11.18 VMOV ARM Core Register to Scalar

Transfers one word to a floating-point register from an ARM core register.

Syntax

```
VMOV{cond}{.32} Dd[x], Rt
```

where:

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

32 is an optional data size specifier.

Dd[x] is the destination, where [x] defines which half of the doubleword is transferred, as follows:
If *x* is 0, the lower half is extracted.
If *x* is 1, the upper half is extracted.

Rt is the source ARM core register.

Operation

This instruction transfers one word to the upper or lower half of a doubleword floating-point register from an ARM core register.

Restrictions

Rt cannot be PC or SP.

Condition Flags

These instructions do not change the flags.

12.6.11.19 VMRS

Move to ARM Core register from floating-point System Register.

Syntax

```
VMRS{cond} Rt, FPSCR
VMRS{cond} APSR_nzcv, FPSCR
```

where:

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

Rt is the destination ARM core register. This register can be R0 - R14.

APSR_nzcv transfers floating-point flags to the APSR flags.

Operation

This instruction performs one of the following actions:

- Copies the value of the FPSCR to a general-purpose register.
- Copies the value of the FPSCR flag bits to the APSR N, Z, C, and V flags.

Restrictions

Rt cannot be PC or SP.

Condition Flags

These instructions optionally change the flags: N, Z, C, V.

12.6.11.20 VMSR

Move to floating-point System Register from ARM Core register.

Syntax

```
VMSR{cond} FPSCR, Rt
```

where:

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

Rt — is the general-purpose register to be transferred to the FPSCR.

Operation

This instruction moves the value of a general-purpose register to the FPSCR. See "Floating-point Status Control Register" for more information.

Restrictions

The restrictions are:

- *Rt* cannot be PC or SP.

Condition Flags

This instruction updates the FPSCR.

12.6.11.21 VMUL

Floating-point Multiply.

Syntax

```
VMUL{cond}.F32 {Sd,} Sn, Sm
```

where:

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

Sd — is the destination floating-point value.

Sn, Sm — are the operand floating-point values.

Operation

This instruction:

1. Multiplies two floating-point values.
2. Places the results in the destination register.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

12.6.11.22 VNEG

Floating-point Negate.

Syntax

```
VNEG{cond}.F32 Sd, Sm
```

where:

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

Sd — is the destination floating-point value.

Sm — is the operand floating-point value.

Operation

This instruction:

1. Negates a floating-point value.
2. Places the results in a second floating-point register.

The floating-point instruction inverts the sign bit.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

12.6.11.23 VNMLA, VNMLS, VNMUL

Floating-point multiply with negation followed by add or subtract.

Syntax

```
VNMLA{cond}.F32 Sd, Sn, Sm
VNMLS{cond}.F32 Sd, Sn, Sm
VNMUL{cond}.F32 {Sd,} Sn, Sm
```

where:

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

Sd — is the destination floating-point register.

Sn, Sm — are the operand floating-point registers.

Operation

The VNMLA instruction:

1. Multiplies two floating-point register values.
2. Adds the negation of the floating-point value in the destination register to the negation of the product.
3. Writes the result back to the destination register.

The VNMLS instruction:

1. Multiplies two floating-point register values.
2. Adds the negation of the floating-point value in the destination register to the product.
3. Writes the result back to the destination register.

The VNMUL instruction:

1. Multiplies together two floating-point register values.
2. Writes the negation of the result to the destination register.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

12.6.11.24 VPOP

Floating-point extension register Pop.

Syntax

```
VPOP{cond}{.size} list
```

where:

cond — is an optional condition code, see "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 single-word registers, separated by commas and surrounded by brackets.

Operation

This instruction loads multiple consecutive extension registers from the stack.

Restrictions

The list must contain at least one register, and not more than sixteen registers.

Condition Flags

These instructions do not change the flags.

12.6.11.25 VPUSH

Floating-point extension register Push.

Syntax

```
VPUSH{cond}{.size} list
```

where:

cond — is an optional condition code, see "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.

12.6.11.26 VSQRT

Floating-point Square Root.

Syntax

```
VSQRT{cond}.F32 Sd, Sm
```

where:

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

Sd — is the destination floating-point value.

Sm — is the operand floating-point value.

Operation

This instruction:

- Calculates the square root of the value in a floating-point register.
- Writes the result to another floating-point register.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

12.6.11.27 VSTM

Floating-point Store Multiple.

Syntax

```
VSTM{mode}{cond}{.size} Rn{!}, list
```

where:

mode — is the addressing mode:
- *IA* Increment After. The consecutive addresses start at the address specified in *Rn*. This is the default and can be omitted.
- *DB* Decrement Before. The consecutive addresses end just before the address specified in *Rn*.

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

size — is an optional data size specifier.
If present, it must be equal to the size in bits, 32 or 64, of the registers in *list*.

Rn — is the base register. The SP can be used.

! — is the function that causes the instruction to write a modified value back to *Rn*.
Required if mode == DB.

list — is a list of the extension registers to be stored, as a list of consecutively numbered doubleword or singleword registers, separated by commas and surrounded by brackets.

Operation

This instruction:

- Stores multiple extension registers to consecutive memory locations using a base address from an ARM core register.

Restrictions

The restrictions are:

- List must contain at least one register.
  If it contains doubleword registers it must not contain more than 16 registers.
- Use of the PC as *Rn* is deprecated.

Condition Flags

These instructions do not change the flags.

12.6.11.28 VSTR

Floating-point Store.

Syntax

```
VSTR{cond}{.32} Sd, [Rn{, #imm}]
VSTR{cond}{.64} Dd, [Rn{, #imm}]
```

where

cond — is an optional condition code, see "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.

12.6.11.29 VSUB

Floating-point Subtract.

Syntax

```
VSUB{cond}.F32 {Sd,} Sn, Sm
```

where:

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

Sd is the destination floating-point value.

Sn, Sm are the operand floating-point value.

Operation

This instruction:

1. Subtracts one floating-point value from another floating-point value.
2. Places the results in the destination floating-point register.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

12.6.12 Miscellaneous Instructions

The table below shows the remaining Cortex-M4 instructions:

Table 12-28. Miscellaneous Instructions

Mnemonic	Description
BKPT	Breakpoint
CPSID	Change Processor State, Disable Interrupts
CPSIE	Change Processor State, Enable Interrupts
DMB	Data Memory Barrier
DSB	Data Synchronization Barrier
ISB	Instruction Synchronization Barrier
MRS	Move from special register to register
MSR	Move from register to special register
NOP	No Operation
SEV	Send Event
SVC	Supervisor Call
WFE	Wait For Event
WFI	Wait For Interrupt

12.6.12.1 BKPT

Breakpoint.

Syntax

```
BKPT #imm
```

where:

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

Operation

The BKPT instruction causes the processor to enter Debug state. Debug tools can use this to investigate system state when the instruction at a particular address is reached.

imm is ignored by the processor. If required, a debugger can use it to store additional information about the breakpoint.

The BKPT instruction can be placed inside an IT block, but it executes unconditionally, unaffected by the condition specified by the IT instruction.

Condition Flags

This instruction does not change the flags.

Examples

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

Note: ARM does not recommend the use of the BKPT instruction with an immediate value set to 0xAB for any purpose other than Semi-hosting.

12.6.12.2 CPS

Change Processor State.

Syntax

```
CPSeffect iflags
```

where:

effect is one of:

- IE Clears the special purpose register.
- ID Sets the special purpose register.

iflags is a sequence of one or more flags:

- i Set or clear PRIMASK.
- f Set or clear FAULTMASK.

Operation

CPS changes the PRIMASK and FAULTMASK special register values. See "Exception Mask Registers" for more information about these registers.

Restrictions

The restrictions are:

- Use CPS only from privileged software, it has no effect if used in unprivileged software.
- CPS cannot be conditional and so must not be used inside an IT block.

Condition Flags

This instruction does not change the condition flags.

Examples

```
CPSID i  ; Disable interrupts and configurable fault handlers (set PRIMASK)
CPSID f  ; Disable interrupts and all fault handlers (set FAULTMASK)
CPSIE i  ; Enable interrupts and configurable fault handlers (clear PRIMASK)
CPSIE f  ; Enable interrupts and fault handlers (clear FAULTMASK)
```

12.6.12.3 DMB

Data Memory Barrier.

Syntax

```
DMB{cond}
```

where:

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

Operation

DMB acts as a data memory barrier. It ensures that all explicit memory accesses that appear, in program order, before the DMB instruction are completed before any explicit memory accesses that appear, in program order, after the DMB instruction. DMB does not affect the ordering or execution of instructions that do not access memory.

Condition Flags

This instruction does not change the flags.

Examples

```
DMB  ; Data Memory Barrier
```

12.6.12.4 DSB

Data Synchronization Barrier.

Syntax

```
DSB{cond}
```

where:

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

Operation

DSB acts as a special data synchronization memory barrier. Instructions that come after the DSB, in program order, do not execute until the DSB instruction completes. The DSB instruction completes when all explicit memory accesses before it complete.

Condition Flags

This instruction does not change the flags.

Examples

```
DSB ; Data Synchronisation Barrier
```

12.6.12.5 ISB

Instruction Synchronization Barrier.

Syntax

```
ISB{cond}
```

where:

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

Operation

ISB acts as an instruction synchronization barrier. It flushes the pipeline of the processor, so that all instructions following the ISB are fetched from memory again, after the ISB instruction has been completed.

Condition Flags

This instruction does not change the flags.

Examples

```
ISB  ; Instruction Synchronisation Barrier
```

12.6.12.6 MRS

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

Syntax

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

where:

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

Rd is the destination register.

spec_reg can be any of: APSR, IPSR, EPSR, IEPSR, IAPSR, EAPSR, PSR, MSP, PSP, PRIMASK, BASEPRI, BASEPRI_MAX, FAULTMASK, or CONTROL.

Operation

Use MRS in combination with MSR as part of a read-modify-write sequence for updating a PSR, for example to clear the Q flag.

In process swap code, the programmers model state of the process being swapped out must be saved, including relevant PSR contents. Similarly, the state of the process being swapped in must also be restored. These operations use MRS in the state-saving instruction sequence and MSR in the state-restoring instruction sequence.

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

See “MSR”.

Restrictions

Rd must not be SP and must not be PC.

Condition Flags

This instruction does not change the flags.

Examples

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

12.6.12.7 MSR

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

Syntax

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

where:

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

Rn — is the source register.

spec_reg — can be any of: APSR, IPSR, EPSR, IEPSR, IAPSR, EAPSR, PSR, MSP, PSP, PRIMASK, BASEPRI, BASEPRI_MAX, FAULTMASK, or CONTROL.

Operation

The register access operation in MSR depends on the privilege level. Unprivileged software can only access the APSR. See "Application Program Status Register". Privileged software can access all special registers.

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

Note: When the user writes to BASEPRI_MAX, the instruction writes to BASEPRI only if either:
Rn is non-zero and the current BASEPRI value is 0.
Rn is non-zero and less than the current BASEPRI value.

See "MRS".

Restrictions

Rn must not be SP and must not be PC.

Condition Flags

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

Examples

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

12.6.12.8 NOP

No Operation.

Syntax

```
NOP{cond}
```

where:

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

Operation

NOP does nothing. NOP is not necessarily a time-consuming NOP. The processor might remove it from the pipeline before it reaches the execution stage.

Use NOP for padding, for example to place the following instruction on a 64-bit boundary.

Condition Flags

This instruction does not change the flags.

Examples

```
NOP  ; No operation
```

12.6.12.9 SEV

Send Event.

Syntax

```
SEV{cond}
```

where:

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

Operation

SEV is a hint instruction that causes an event to be signaled to all processors within a multiprocessor system. It also sets the local event register to 1, see “Power Management”.

Condition Flags

This instruction does not change the flags.

Examples

```
SEV ; Send Event
```

12.6.12.10 SVC

Supervisor Call.

Syntax

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

where:

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

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

Operation

The SVC instruction causes the SVC exception.

imm is ignored by the processor. If required, it can be retrieved by the exception handler to determine what service is being requested.

Condition Flags

This instruction does not change the flags.

Examples

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

12.6.12.11 WFE

Wait For Event.

Syntax

```
WFE{cond}
```

where:

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

Operation

WFE is a hint instruction.

If the event register is 0, WFE suspends execution until one of the following events occurs:

- An exception, unless masked by the exception mask registers or the current priority level.
- An exception enters the Pending state, if SEVONPEND in the System Control Register is set.
- A Debug Entry request, if Debug is enabled.
- An event signaled by a peripheral or another processor in a multiprocessor system using the SEV instruction.

If the event register is 1, WFE clears it to 0 and returns immediately.

For more information, see “Power Management”.

Condition Flags

This instruction does not change the flags.

Examples

```
WFE  ; Wait for event
```

12.6.12.12 WFI

Wait for Interrupt.

Syntax

```
WFI{cond}
```

where:

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

Operation

WFI is a hint instruction that suspends execution until one of the following events occurs:

- An exception.
- A Debug Entry request, regardless of whether Debug is enabled.

Condition Flags

This instruction does not change the flags.

Examples

```
WFI ; Wait for interrupt
```

12.7 Cortex-M4 Core Peripherals

12.7.1 Peripherals

- Nested Vectored Interrupt Controller (NVIC)
  The Nested Vectored Interrupt Controller (NVIC) is an embedded interrupt controller that supports low latency interrupt processing. See Section 12.8 ”Nested Vectored Interrupt Controller (NVIC)”
- System Control Block (SCB)
  The System Control Block (SCB) is the programmers model interface to the processor. It provides system implementation information and system control, including configuration, control, and reporting of system exceptions. See Section 12.9 ”System Control Block (SCB)”
- System Timer (SysTick)
  The System Timer, SysTick, is a 24-bit count-down timer. Use this as a Real Time Operating System (RTOS) tick timer or as a simple counter. See Section 12.10 ”System Timer (SysTick)”
- Memory Protection Unit (MPU)
  The Memory Protection Unit (MPU) improves system reliability by defining the memory attributes for different memory regions. It provides up to eight different regions, and an optional predefined background region. See Section 12.11 ”Memory Protection Unit (MPU)”
- Floating-point Unit (FPU)
  The Floating-point Unit (FPU) provides IEEE754-compliant operations on single-precision, 32-bit, floating-point values. See Section 12.12 ”Floating Point Unit (FPU)”

12.7.2 Address Map

The address map of the *Private peripheral bus* (PPB) is given in the following table:

Table 12-29. Core Peripheral Register Regions

Address	Core Peripheral
0xE000E008 - 0xE000E00F	System Control Block
0xE000E010 - 0xE000E01F	System Timer
0xE000E100 - 0xE000E4EF	Nested Vectored Interrupt Controller
0xE000ED00 - 0xE000ED3F	System Control block
0xE000ED90 - 0xE000EDB8	Memory Protection Unit
0xE000EF00 - 0xE000EF03	Nested Vectored Interrupt Controller
0xE000EF30 - 0xE000EF44	Floating-point Unit

In register descriptions:

- The *required privilege* gives the privilege level required to access the register, as follows:
  - Privileged: Only privileged software can access the register.
  - Unprivileged: Both unprivileged and privileged software can access the register.

12.8 Nested Vectored Interrupt Controller (NVIC)

This section describes the NVIC and the registers it uses. The NVIC supports:

- Up to 41 interrupts.
- A programmable priority level of 0 - 15 for each interrupt. A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority.
- Level detection of interrupt signals.
- Dynamic reprioritization of interrupts.
- Grouping of priority values into group priority and subpriority fields.
- Interrupt tail-chaining.
- An external *Non-maskable interrupt* (NMI).

The processor automatically stacks its state on exception entry and unstacks this state on exception exit, with no instruction overhead. This provides low latency exception handling.

12.8.1 Level-sensitive Interrupts

The processor supports level-sensitive interrupts. A level-sensitive interrupt is held asserted until the peripheral deasserts the interrupt signal. Typically, this happens because the ISR accesses the peripheral, causing it to clear the interrupt request.

When the processor enters the ISR, it automatically removes the pending state from the interrupt (see “Hardware and Software Control of Interrupts”). For a level-sensitive interrupt, if the signal is not deasserted before the processor returns from the ISR, the interrupt becomes pending again, and the processor must execute its ISR again. This means that the peripheral can hold the interrupt signal asserted until it no longer requires servicing.

12.8.1.1 Hardware and Software Control of Interrupts

The Cortex-M4 latches all interrupts. A peripheral interrupt becomes pending for one of the following reasons:

- The NVIC detects that the interrupt signal is HIGH and the interrupt is not active.
- The NVIC detects a rising edge on the interrupt signal.
- A software writes to the corresponding interrupt set-pending register bit, see “Interrupt Set-pending Registers”, or to the NVIC_STIR to make an interrupt pending, see “Software Trigger Interrupt Register”.

A pending interrupt remains pending until one of the following:

- The processor enters the ISR for the interrupt. This changes the state of the interrupt from pending to active. Then:
  - For a level-sensitive interrupt, when the processor returns from the ISR, the NVIC samples the interrupt signal. If the signal is asserted, the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR. Otherwise, the state of the interrupt changes to inactive.
- Software writes to the corresponding interrupt clear-pending register bit.
  For a level-sensitive interrupt, if the interrupt signal is still asserted, the state of the interrupt does not change. Otherwise, the state of the interrupt changes to inactive.

12.8.2 NVIC Design Hints and Tips

Ensure that the software uses correctly aligned register accesses. The processor does not support unaligned accesses to NVIC registers. See the individual register descriptions for the supported access sizes.

A interrupt can enter a pending state even if it is disabled. Disabling an interrupt only prevents the processor from taking that interrupt.

Before programming SCB_VTOR to relocate the vector table, ensure that the vector table entries of the new vector table are set up for fault handlers, NMI and all enabled exception like interrupts. For more information, see the "Vector Table Offset Register".

12.8.2.1 NVIC Programming Hints

The software uses the CPSIE I and CPSID I instructions to enable and disable the interrupts. The CMSIS provides the following intrinsic functions for these instructions:

void __disable_irq(void) // Disable Interrupts

void __enable_irq(void) // Enable Interrupts

In addition, the CMSIS provides a number of functions for NVIC control, including:

Table 12-30. CMSIS Functions for NVIC Control

CMSIS Interrupt Control Function	Description
void NVIC_SetPriorityGrouping(uint32_t priority_grouping)	Set the priority grouping
void NVIC_EnableIRQ(IRQn_t IRQn)	Enable IRQn
void NVIC_DisableIRQ(IRQn_t IRQn)	Disable IRQn
uint32_t NVIC_GetPendingIRQ (IRQn_t IRQn)	Return true (IRQ-Number) if IRQn is pending
void NVIC_SetPendingIRQ (IRQn_t IRQn)	Set IRQn pending
void NVIC_ClearPendingIRQ (IRQn_t IRQn)	Clear IRQn pending status
uint32_t NVIC_GetActive (IRQn_t IRQn)	Return the IRQ number of the active interrupt
void NVIC_SetPriority (IRQn_t IRQn, uint32_t priority)	Set priority for IRQn
uint32_t NVIC_GetPriority (IRQn_t IRQn)	Read priority of IRQn
void NVIC_SystemReset (void)	Reset the system

The input parameter IRQn is the IRQ number. For more information about these functions, see the CMSIS documentation.

To improve software efficiency, the CMSIS simplifies the NVIC register presentation. In the CMSIS:

- The Set-enable, Clear-enable, Set-pending, Clear-pending and Active Bit registers map to arrays of 32-bit integers, so that:
  - The array ISER[0] 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 - IPR10) 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 12-31 shows how the interrupts, or IRQ numbers, map onto the interrupt registers and corresponding CMSIS variables that have one bit per interrupt.

Table 12-31. Mapping of Interrupts

Interrupts	CMSIS Array Elements[1]				
	Set-enable	Clear-enable	Set-pending	Clear-pending	Active Bit
0 - 31	ISER[0]	ICER[0]	ISPR[0]	ICPR[0]	IABR[0]
32 - 41	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.

12.8.3 Nested Vectored Interrupt Controller (NVIC) User Interface

Table 12-32. Nested Vectored Interrupt Controller (NVIC) Register Mapping

Offset	Register	Name	Access	Reset
0xE000E100	Interrupt Set-enable Register 0	NVIC_ISER0	Read/Write	0x00000000
...	...	...	...	...
0xE000E11C	Interrupt Set-enable Register 7	NVIC_ISER7	Read/Write	0x00000000
0XE000E180	Interrupt Clear-enable 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
...	...	...	...	...
0xE000E428	Interrupt Priority Register 10	NVIC_IPR10	Read/Write	0x00000000
0xE000EF00	Software Trigger Interrupt Register	NVIC_STIR	Write-only	0x00000000

12.8.3.1 Interrupt Set-enable Registers

Name: NVIC_ISERx [x=0..7]

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
SETENA							

23	22	21	20	19	18	17	16
SETENA							

15	14	13	12	11	10	9	8
SETENA							

7	6	5	4	3	2	1	0
SETENA							

These registers enable interrupts and show which interrupts are enabled.

- **SETENA: Interrupt Set-enable**

Write:

0: No effect.

1: Enables the interrupt.

Read:

0: Interrupt disabled.

1: Interrupt enabled.

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

12.8.3.2 Interrupt Clear-enable Registers

Name: NVIC_ICERx [x=0..7]

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
CLRENA							

23	22	21	20	19	18	17	16
CLRENA							

15	14	13	12	11	10	9	8
CLRENA							

7	6	5	4	3	2	1	0
CLRENA							

These registers disable interrupts, and show which interrupts are enabled.

- **CLRENA: Interrupt Clear-enable**

Write:

0: No effect.

1: Disables the interrupt.

Read:

0: Interrupt disabled.

1: Interrupt enabled.

12.8.3.3 Interrupt Set-pending Registers

Name: NVIC_ISPRx [x=0..7]

Access: Read/Write

Reset: 0x00000000

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.

12.8.3.4 Interrupt Clear-pending Registers

Name: NVIC_ICPRx [x=0..7]

Access: Read/Write

Reset: 0x00000000

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.

12.8.3.5 Interrupt Active Bit Registers

Name: NVIC_IABRx [x=0..7]

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
ACTIVE							

23	22	21	20	19	18	17	16
ACTIVE							

15	14	13	12	11	10	9	8
ACTIVE							

7	6	5	4	3	2	1	0
ACTIVE							

These registers indicate which interrupts are active.

- **ACTIVE: Interrupt Active Flags**

0: Interrupt is not active.

1: Interrupt is active.

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

12.8.3.6 Interrupt Priority Registers

Name: NVIC_IPRx [x=0..10]

Access: Read/Write

Reset: 0x00000000

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_IPR10 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[40].

- **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 [40] interrupt priority array, that provides the software view of the interrupt priorities, see Table 12-30, “CMSIS Functions for NVIC Control” .
3. The corresponding IPR number *n* is given by *n* = *m* DIV 4.
4. The byte offset of the required Priority field in this register is *m* MOD 4.

12.8.3.7 Software Trigger Interrupt Register

Name: NVIC_STIR

Access: Write-only

Reset: 0x00000000

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

Write to this register to generate an interrupt from the software.

- **INTID: Interrupt ID**

Interrupt ID of the interrupt to trigger, in the range 0 - 239. For example, a value of 0x03 specifies interrupt IRQ3.

12.9 System Control Block (SCB)

The System Control Block (SCB) provides system implementation information, and system control. This includes configuration, control, and reporting of the system exceptions.

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

- Except for the SCB_CFSR and SCB_SHPR1 - SCB_SHPR3 registers, it must use aligned word accesses.
- For the SCB_CFSR and SCB_SHPR1 - SCB_SHPR3 registers, it can use byte or aligned halfword or word accesses.

The processor does not support unaligned accesses to system control block registers.

In a fault handler, to determine the true faulting address:

1. Read and save the MMFAR or SCB_BFAR value.
2. Read the MMARVALID bit in the MMFSR subregister, or the BFARVALID bit in the BFSR subregister. The SCB_MMFAR or SCB_BFAR address is valid only if this bit is 1.

The software must follow this sequence because another higher priority exception might change the SCB_MMFAR or SCB_BFAR value. For example, if a higher priority handler preempts the current fault handler, the other fault might change the SCB_MMFAR or SCB_BFAR value.

12.9.1 System Control Block (SCB) User Interface

Table 12-33. System Control Block (SCB) Register Mapping

Offset	Register	Name	Access	Reset
0xE000E008	Auxiliary Control Register	SCB_ACTLR	Read/Write	0x00000000
0xE000ED00	CPUID Base Register	SCB_CPUID	Read-only	0x410FC240
0xE000ED04	Interrupt Control and State Register	SCB_ICSR	Read/Write(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).

12.9.1.1 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.

12.9.1.2 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.

12.9.1.3 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.

12.9.1.4 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.

12.9.1.5 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.

12.9.1.6 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.

12.9.1.7 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	–	USERSETM PEND	NONBASET HRDENA

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”.

12.9.1.8 System Handler Priority Registers

The SCB_SHPR1 - SCB_SHPR3 registers set the priority level, 0 to 15 of the exception handlers that have configurable priority. They are byte-accessible.

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

Table 12-34. System Fault Handler Priority Fields

Handler	Field	Register Description
Memory management fault (MemManage)	PRI_4	"System Handler Priority Register 1"
Bus fault (BusFault)	PRI_5	
Usage fault (UsageFault)	PRI_6	
SVCall	PRI_11	"System Handler Priority Register 2"
PendSV	PRI_14	"System Handler Priority Register 3"
SysTick	PRI_15	

Each PRI_N field is 8 bits wide, but the processor implements only bits [7:4] of each field, and bits [3:0] read as zero and ignore writes.

12.9.1.9 System Handler Priority Register 1

Name: SCB_SHPR1

Access: Read/Write

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.

12.9.1.10 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.

12.9.1.11 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.

12.9.1.12 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.

12.9.1.13 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".

12.9.1.14 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 12.9.1.13.

- **BFSR: Bus Fault Status Subregister**

The flags in the BFSR subregister indicate the cause of a bus access fault. See bitfield [14..8] description in Section 12.9.1.13.

- **UFSR: Usage Fault Status Subregister**

The flags in the UFSR subregister indicate the cause of a usage fault. See bitfield [31..15] description in Section 12.9.1.13.

Note: The UFSR bits are sticky. This means that as one or more fault occurs, the associated bits are set to 1. A bit that is set to 1 is cleared to 0 only by writing 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.

12.9.1.15 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 writing 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 writing a 1 to that bit, or by a reset.

12.9.1.16 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”.

12.9.1.17 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”.

12.10 System Timer (SysTick)

The processor has a 24-bit system timer, SysTick, that counts down from the reload value to zero, reloads (wraps to) the value in the SYST_RVR on the next clock edge, then counts down on subsequent clocks.

When the processor is halted for debugging, the counter does not decrement.

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

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

The SysTick counter reload and current value are undefined at reset; the correct initialization sequence for the SysTick counter is:

1. Program the reload value.
2. Clear the current value.
3. Program the Control and Status register.

12.10.1 System Timer (SysTick) User Interface

Table 12-35. System Timer (SYST) Register Mapping

Offset	Register	Name	Access	Reset
0xE000E010	SysTick Control and Status Register	SYST_CSR	Read/Write	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	0x000030D4

12.10.1.1 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.

12.10.1.2 SysTick Reload Value Register

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.

12.10.1.3 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.

12.10.1.4 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 0x000030D4 (12500 decimal).

In order to achieve a 1 ms timebase on SysTick, 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 0x00001770 (48000 kHz/8).

12.11 Memory Protection Unit (MPU)

The MPU divides the memory map into a number of regions, and defines the location, size, access permissions, and memory attributes of each region. It supports:

- Independent attribute settings for each region.
- Overlapping regions.
- Export of memory attributes to the system.

The memory attributes affect the behavior of memory accesses to the region. The Cortex-M4 MPU defines:

- Eight separate memory regions, 0 - 7.
- A background region.

When memory regions overlap, a memory access is affected by the attributes of the region with the highest number. For example, the attributes for region 7 take precedence over the attributes of any region that overlaps region 7.

The background region has the same memory access attributes as the default memory map, but is accessible from privileged software only.

The Cortex-M4 MPU memory map is unified. This means that instruction accesses and data accesses have the same region settings.

If a program accesses a memory location that is prohibited by the MPU, the processor generates a memory management fault. This causes a fault exception, and might cause the termination of the process in an OS environment.

In an OS environment, the kernel can update the MPU region setting dynamically based on the process to be executed. Typically, an embedded OS uses the MPU for memory protection.

The configuration of MPU regions is based on memory types (see “Memory Regions, Types and Attributes”).

Table 12-36 shows the possible MPU region attributes. These include Shareability 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 12-36. 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	-	Normal memory that is shared between several processors.
	Non-shared	-	Normal memory that only a single processor uses.

12.11.1 MPU Access Permission Attributes

This section describes the MPU access permission attributes. The access permission bits (TEX, C, B, S, AP, and XN) of the MPU_RASR control the access to the corresponding memory region. If an access is made to an area of memory without the required permissions, then the MPU generates a permission fault.

The table below shows the encodings for the TEX, C, B, and S access permission bits.

Table 12-37. TEX, C, B, and S Encoding

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

Notes: 1. The MPU ignores the value of this bit.

Table 12-38 shows the cache policy for memory attribute encodings with a TEX value is in the range 4 - 7.

Table 12-38. 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 12-39 shows the AP encodings that define the access permissions for privileged and unprivileged software.

Table 12-39. 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

12.11.1.1 MPU Mismatch

When an access violates the MPU permissions, the processor generates a memory management fault, see "Exceptions and Interrupts". The MMFSR indicates the cause of the fault. See "MMFSR: Memory Management Fault Status Subregister" for more information.

12.11.1.2 Updating an MPU Region

To update the attributes for an MPU region, update the MPU_RNR, MPU_RBAR and MPU_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.

12.11.1.3 Updating an MPU Region Using Separate Words

Simple code to configure one region:

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
LDR R0,=MPU_RNR         ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]      ; Region Number
STR R4, [R0, #0x4]      ; Region Base Address
STRH R2, [R0, #0x8]     ; Region Size and Enable
STRH R3, [R0, #0xA]     ; Region Attribute
```

Disable a region before writing new region settings to the MPU, if the region being changed was previously enabled. For example:

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
LDR R0,=MPU_RNR          ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]       ; Region Number
BIC R2, R2, #1           ; Disable
STRH R2, [R0, #0x8]      ; Region Size and Enable
STR R4, [R0, #0x4]       ; Region Base Address
STRH R3, [R0, #0xA]      ; Region Attribute
ORR R2, #1               ; Enable
STRH R2, [R0, #0x8]      ; Region Size and Enable
```

The software must use memory barrier instructions:

- Before the MPU setup, if there might be outstanding memory transfers, such as buffered writes, that might be affected by the change in MPU settings.
- After the MPU setup, if it includes memory transfers that must use the new MPU settings.

However, memory barrier instructions are not required if the MPU setup process starts by entering an exception handler, or is followed by an exception return, because the exception entry and exception return mechanisms cause memory barrier behavior.

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

For example, if the user wants all of the memory access behavior to take effect immediately after the programming sequence, a DSB instruction and an ISB instruction must be used. A DSB is required after changing MPU settings, such as at the end of a context switch. An ISB is required if the code that programs the MPU region or regions is entered using a branch or call. If the programming sequence is entered using a return from exception, or by taking an exception, then an ISB is not required.

12.11.1.4 Updating an MPU Region Using Multi-word Writes

The user can program directly using multi-word writes, depending on how the information is divided. Consider the following reprogramming:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
LDR R0, =MPU_RNR    ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]  ; Region Number
STR R2, [R0, #0x4]  ; Region Base Address
STR R3, [R0, #0x8]  ; Region Attribute, Size and Enable
```

Use an STM instruction to optimize this:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
LDR R0, =MPU_RNR    ; 0xE000ED98, MPU region number register
STM R0, {R1-R3}     ; Region Number, address, attribute, size and enable
```

This can be done in two words for pre-packed information. This means that the MPU_RBAR contains the required region number and had the VALID bit set to 1. See “MPU Region Base Address Register”. Use this when the data is statically packed, for example in a boot loader:

```
; R1 = address and region number in one
; R2 = size and attributes in one
LDR R0, =MPU_RBAR   ; 0xE000ED9C, MPU Region Base register
STR R1, [R0, #0x0]  ; Region base address and
                    ; region number combined with VALID (bit 4) set to 1
STR R2, [R0, #0x4]  ; Region Attribute, Size and Enable
```

Use an STM instruction to optimize this:

```
; R1 = address and region number in one
; R2 = size and attributes in one
LDR R0,=MPU_RBAR    ; 0xE000ED9C, MPU Region Base register
STM R0, {R1-R2}     ; Region base address, region number and VALID bit,
                    ; and Region Attribute, Size and Enable
```

12.11.1.5 Subregions

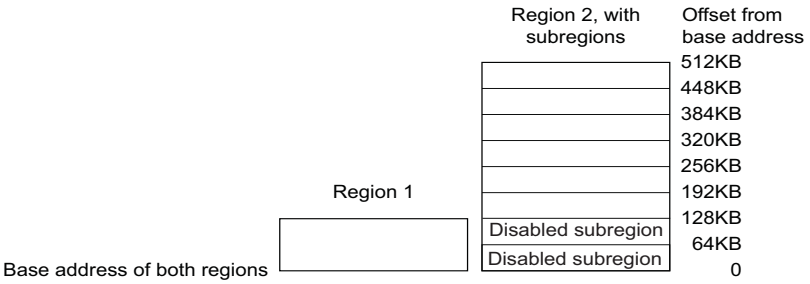
Regions of 256 bytes or more are divided into eight equal-sized subregions. Set the corresponding bit in the SRD field of the MPU_RASR field to disable a subregion. See "MPU Region Attribute and Size Register". The least significant bit of SRD controls the first subregion, and the most significant bit 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.

12.11.1.6 Example of SRD Use

Two regions with the same base address overlap. Region 1 is 128 KB, and region 2 is 512 KB. To ensure the attributes from region 1 apply to the first 128 KB region, set the SRD field for region 2 to b00000011 to disable the first two subregions, as in Figure 12-13 below:

Figure 12-13. SRD Use



12.11.1.7 MPU Design Hints And Tips

To avoid unexpected behavior, disable the interrupts before updating the attributes of a region that the interrupt handlers might access.

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

- Except for the MPU_RASR, 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 12-40. 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.

12.11.2 Memory Protection Unit (MPU) User Interface

Table 12-41. 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

12.11.2.1 MPU Type Register

Name: MPU_TYPE

Access: Read-only

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
IREGION							

15	14	13	12	11	10	9	8
DREGION							

7	6	5	4	3	2	1	0
–							SEPARATE

The MPU_TYPE register indicates whether the MPU is present, and if so, how many regions it supports.

- **IREGION: Instruction Region**

Indicates the number of supported MPU instruction regions.

Always contains 0x00. The MPU memory map is unified and is described by the DREGION field.

- **DREGION: Data Region**

Indicates the number of supported MPU data regions:

0x08 = Eight MPU regions.

- **SEPARATE: Separate Instruction**

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

0: Unified.

12.11.2.2 MPU Control Register

Name: MPU_CTRL

Access: Read/Write

31	30	29	28	27	26	25	24
–							
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.

12.11.2.3 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.

12.11.2.4 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.

12.11.2.5 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 12-39.

- **TEX, C, B: Memory Access Attributes**

See Table 12-37.

- **S: Shareable**

See Table 12-37.

- **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”.

12.11.2.6 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.

12.11.2.7 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 12-39.

- **TEX, C, B: Memory Access Attributes**

See Table 12-37.

- **S: Shareable**

See Table 12-37.

- **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”.

12.11.2.8 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.

12.11.2.9 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 12-39.

- **TEX, C, B: Memory Access Attributes**

See Table 12-37.

- **S: Shareable**

See Table 12-37.

- **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”.

12.11.2.10 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.

12.11.2.11 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 12-39.

- **TEX, C, B: Memory Access Attributes**

See Table 12-37.

- **S: Shareable**

See Table 12-37.

- **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”.

12.12 Floating Point Unit (FPU)

The Cortex-M4F FPU implements the FPv4-SP floating-point extension.

The FPU fully supports single-precision add, subtract, multiply, divide, multiply and accumulate, and square root operations. It also provides conversions between fixed-point and floating-point data formats, and floating-point constant instructions.

The FPU provides floating-point computation functionality that is compliant with the ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic, referred to as the IEEE 754 standard.

The FPU contains 32 single-precision extension registers, which can also be accessed as 16 doubleword registers for load, store, and move operations.

12.12.1 Enabling the FPU

The FPU is disabled from reset. It must be enabled before any floating-point instructions can be used. An example code sequence for enabling the FPU in both privileged and user modes is showed below. The processor must be in privileged mode to read from and write to the CPACR.

Example of Enabling the FPU:

```
; CPACR is located at address 0xE000ED88
LDR.W R0, =0xE000ED88
; Read CPACR
LDR R1, [R0]
; Set bits 20-23 to enable CP10 and CP11 coprocessors
ORR R1, R1, #(0xF << 20)
; Write back the modified value to the CPACR
STR R1, [R0]; wait for store to complete
DSB
;reset pipeline now the FPU is enabled
ISB
```

12.12.2 Floating Point Unit (FPU) User Interface

Table 12-42. 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

12.12.2.1 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.

12.12.2.2 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.

12.12.2.3 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.

12.12.2.4 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.

12.12.2.5 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.

12.13 Glossary

This glossary describes some of the terms used in technical documents from ARM.

Abort	A mechanism that indicates to a processor that the value associated with a memory access is invalid. An abort can be caused by the external or internal memory system as a result of attempting to access invalid instruction or data memory.
Aligned	A data item stored at an address that is divisible by the number of bytes that defines the data size is said to be aligned. Aligned words and halfwords have addresses that are divisible by four and two respectively. The terms word-aligned and halfword-aligned therefore stipulate addresses that are divisible by four and two respectively.
Banked register	A register that has multiple physical copies, where the state of the processor determines which copy is used. The Stack Pointer, SP (R13) is a banked register.
Base register	In instruction descriptions, a register specified by a load or store instruction that is used to hold the base value for the instruction's address calculation. Depending on the instruction and its addressing mode, an offset can be added to or subtracted from the base register value to form the address that is sent to memory. *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.
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.
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.

13. Debug and Test Features

13.1 Description

A number of complementary debug and test capabilities are available to users. The Serial Wire/JTAG Debug Port (SWJ-DP) combining a Serial Wire Debug Port (SW-DP) and a JTAG Debug Port (JTAG-DP) is used for standard debugging functions, such as downloading code and single-stepping through programs. It also embeds a serial wire trace.

13.2 Associated Documentation

The SAM4CP implements the standard ARM CoreSight™ Macrocell. For information on CoreSight, the following reference documents are available from the ARM website:

- Cortex-M4/M4F Technical Reference Manual (ARM DDI 0439C).
- CoreSight Technology System Design Guide (ARM DGI 0012D).
- CoreSight Components Technical Reference Manual (ARM DDI 0314H).
- ARM Debug Interface v5 Architecture Specification (Doc. ARM IHI 0031A).
- ARMv7-M Architecture Reference Manual (ARM DDI 0403D).

13.3 Embedded Characteristics

- Dual Core Debugging with common Serial Wire Debug Port (SW-DP) and Serial Wire JTAG Debug Port (SWJ-DP) debug access port connected to both cores.
- Star Topology AHB-AP Debug Access Port Implementation with common SW-DP / SWJ-DP providing higher performance than daisy-chain topology.
- Possibility to halt each core on debug event on the other core (hardware).
- Possibility to restart each core when the other core has restarted (hardware).
- Synchronization and software cross-triggering with Debugger.
- Instrumentation Trace Macrocell (ITM) on both core for support of ‘printf’ style debugging.
- Mux 2-1 to trace chosen core (limit the number of out put pin).
- Single wire Viewer or Clock mode (4-bit parallel output ports).
- 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.
- Flash Patch and Breakpoint (FPB) unit for implementing breakpoints and code patches.
- Data Watchpoint and Trace (DWT) unit for implementing watch points, data tracing, and system profiling.
- IEEE 1149.1 JTAG Boundary scan on All Digital Pins.

Figure 13-1. Debug and Test Block Diagram

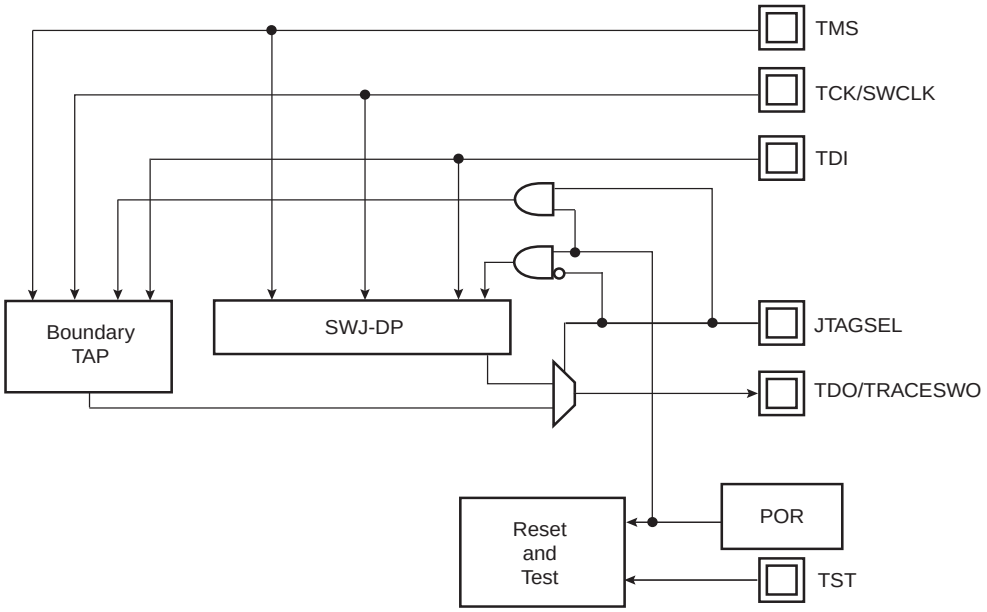
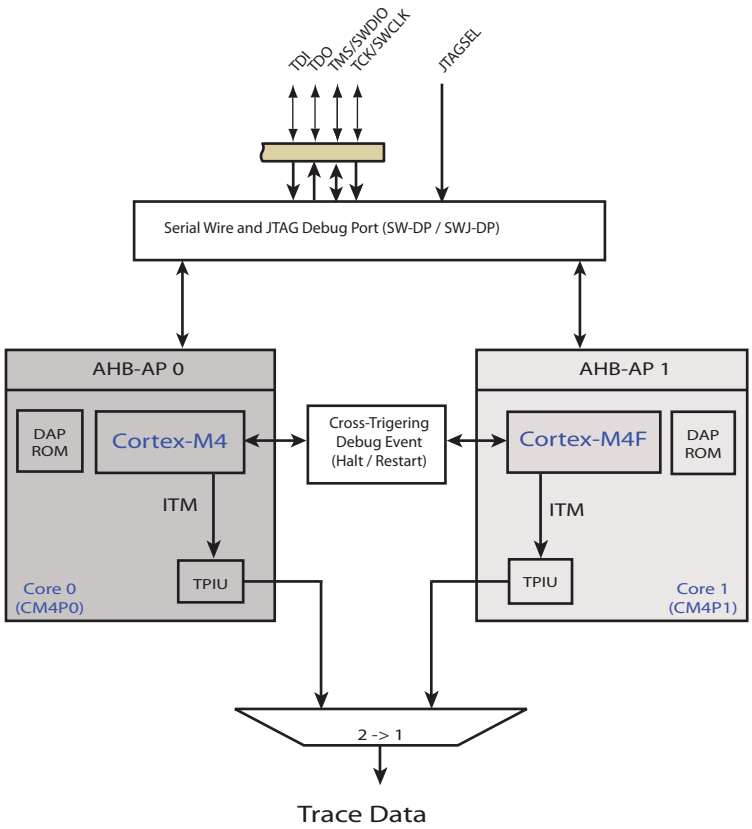


Figure 13-2 illustrates the dual core debug implementation using only one SW-JTAG/SW-DP Debug Access Port. Star topology has been used to connect the AHB-AP 0 (Core 0) and AHB-AP 1 (Core) rather than legacy daisy chaining method. Star topology provides higher performance than daisy-chain topology. This core debug architecture is fully supported by debug tools vendors.

Figure 13-2. Dual Core Debug Architecture



13.4 Cross Triggering Debug Events

Cross Triggering (CT) as shown in Figure 13-2 is an Atmel module that enables two cores to send and receive debug events to and from each other. This module is used to debug two applications at the same time (one application running on each core).

CT enables core 0 (or 1) to trigger a debug event (halt) to core 1 (or 0) to enter Debug mode. The debug event can be sent when the core 0 (or 1) enters Debug mode (such as breakpoint) or at run-time.

Once core 0 (or 1) gets out of Debug mode, it releases core 1 (0) from Debug mode as well.

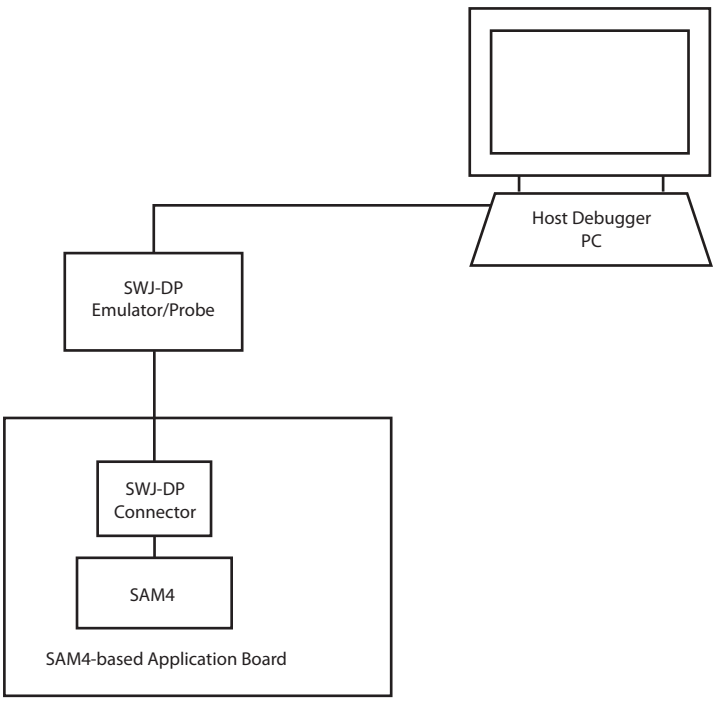
The Cross Triggering configuration is located in the Special Function Register in the Matrix User Interface.

13.5 Application Examples

13.5.1 Debug Environment

Figure 13-3 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, as well as viewing core and peripheral registers.

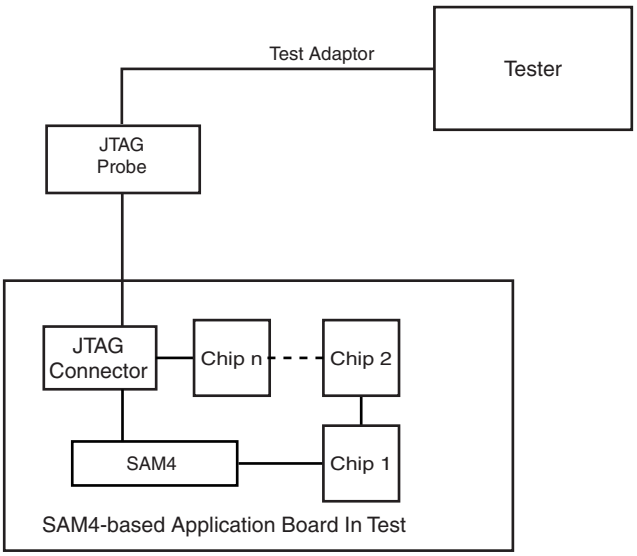
Figure 13-3. Application Debug Environment Example



13.5.2 Test Environment

Figure 13-4 shows an example of a test environment (JTAG Boundary scan). Test vectors are sent and interpreted by the tester. In this example, the “board in test” is designed using a number of JTAG-compliant devices. These devices can be connected to form a single scan chain.

Figure 13-4. Application Test Environment Example



13.6 Debug and Test Pin Description

Table 13-1. 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

13.7 Functional Description

13.7.1 Test Pin

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

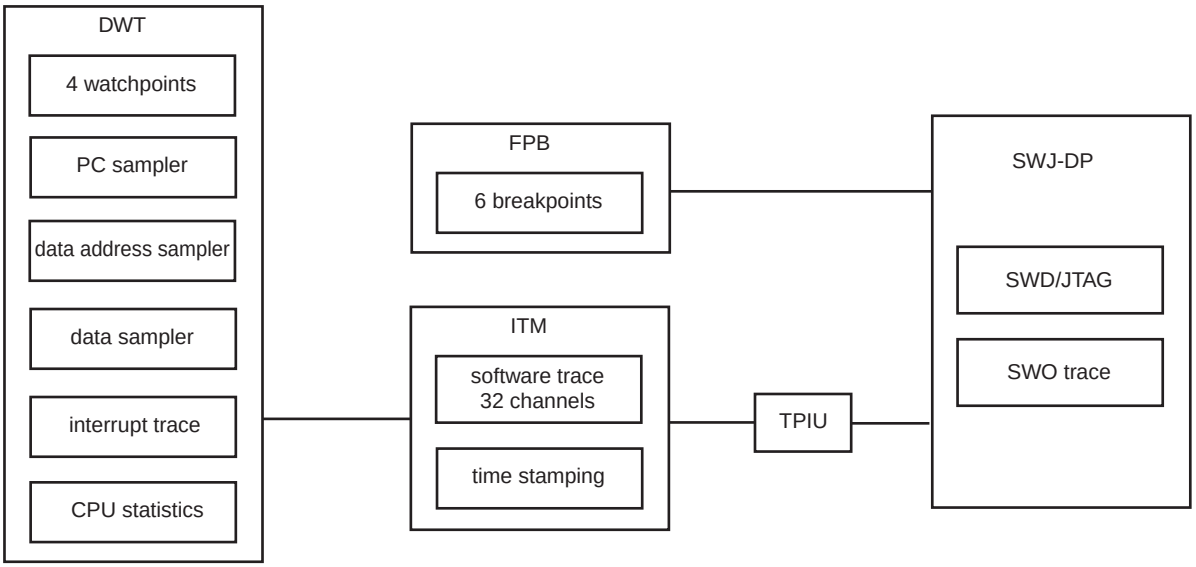
13.7.2 Debug Architecture

Figure 13-5 illustrates the debug architecture. The Cortex-M4 embeds four functional units for debug:

- SWJ-DP (Serial Wire/JTAG Debug Port).
- FPB (Flash Patch Breakpoint).
- DWT (Data Watchpoint and Trace).
- ITM (Instrumentation Trace Macrocell).
- TPIU (Trace Port Interface Unit).

The information that follows is mainly dedicated to developers of SWJ-DP Emulators/Probes and debugging tool vendors for Cortex-M4 based microcontrollers. For further details on SWJ-DP, see the Cortex-M4 technical reference manual.

Figure 13-5. Debug Architecture



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

The Cortex-M4 embeds a SWJ-DP Debug port which is the standard CoreSight debug port. It combines the Serial Wire Debug Port (SW-DP), from 2 to 3 pins, and the JTAG Debug Port (JTAG-DP), 5 pins.

By default, the JTAG-DP is active. If the host debugger needs to switch to the SW-DP, it must provide a dedicated JTAG sequence on TMS/SWDIO and TCK/SWCLK. This disables JTAG-DP and enables SW-DP.

When SW-DP is active, TDO/TRACESWO can be used for trace. The asynchronous TRACE output (TRACESWO) is multiplexed with TDO and thus the asynchronous trace can only be used with SW-DP.

Table 13-2. SWJ-DP Pin List

Pin Name	JTAG Port	Serial Wire Debug Port
TMS/SWDIO	TMS	SWDIO
TCK/SWCLK	TCK	SWCLK
TDI	TDI	-
TDO/TRACESWO	TDO	TRACESWO (optional: trace)

SW-DP or JTAG-DP mode is selected when JTAGSEL is low. It is not possible to switch directly between SWJ-DP and JTAG boundary scan operations. A chip reset must be performed after JTAGSEL is changed.

13.7.3.1 SW-DP and JTAG-DP Selection

Debug port selection is done by sending a specific **SWDIOTMS** sequence. The JTAG-DP is selected by default after reset.

- Switch from JTAG-DP to SW-DP. The sequence is:
  - Send more than 50 **SWCLKTCK** cycles with **SWDIOTMS** = 1
  - Send the 16-bit sequence on **SWDIOTMS** = 0111100111100111 (0x79E7 MSB first)
  - Send more than 50 **SWCLKTCK** cycles with **SWDIOTMS** = 1
- Switch from SWD to JTAG. The sequence is:
  - Send more than 50 **SWCLKTCK** cycles with **SWDIOTMS** = 1
  - Send the 16-bit sequence on **SWDIOTMS** = 0011110011100111 (0x3CE7 MSB first)
  - Send more than 50 **SWCLKTCK** cycles with **SWDIOTMS** = 1

13.7.4 FPB (Flash Patch Breakpoint)

The FPB:

- Implements hardware breakpoints.
- Patches code and data from code space to system space.

The FPB unit contains:

- Two literal comparators for matching against literal loads from Code space, and remapping to a corresponding area in System space.
- Six instruction comparators for matching against instruction fetches from Code space, and remapping to a corresponding area in System space.
- Alternatively, comparators can also be configured to generate a Breakpoint instruction to the processor core on a match.

13.7.5 DWT (Data Watchpoint and Trace)

The DWT contains four comparators which can be configured to generate the following:

- PC sampling packets at set intervals.
- PC or Data watchpoint packets.
- Watchpoint event to halt core.

The DWT contains counters for the items that follow:

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

13.7.6 ITM (Instrumentation Trace Macrocell)

The ITM is an application-driven trace source that supports 'printf' style debugging to trace operating system (OS) and application events, and provides diagnostic system information. The ITM transmits the 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 13.7.6.1 "How to Configure the ITM".
- **Hardware trace**: The ITM transmits packets generated by the DWT.
- **Time stamping**: Timestamps are transmitted relative to packets. The ITM contains a 21-bit counter to generate the timestamp.

13.7.6.1 How to Configure the ITM

The following example describes how to output trace data in Asynchronous Trace mode.

- Configure the TPIU for Asynchronous Trace mode (refer to Section 13.7.6.3 "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.
  - Set 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 results 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 ITM.

  The TPIU formats and transmits trace data off-chip at frequencies asynchronous to the core.

13.7.6.2 Asynchronous Mode

The TPIU is configured in Asynchronous mode, trace data are output using the single TRACESWO pin. The TRACESWO signal is multiplexed with the TDO signal of the JTAG Debug Port. As a consequence, Asynchronous Trace mode is only available when the Serial Wire Debug mode is selected since TDO signal is used in JTAG Debug mode.

Two encoding formats are available for the single pin output:

- Manchester encoded stream. This is the reset value.
- NRZ_based UART byte structure.

13.7.6.3 How to Configure the TPIU

This example is applicable with Asynchronous Trace mode only.

- Set the TRCENA bit to 1 into the Debug Exception and Monitor Register (0xE000EDFC) to enable the use of trace and debug blocks.
- Write 0x2 into the Selected Pin Protocol Register
  - Select the Serial Wire Output – NRZ
- Write 0x100 into the Formatter and Flush Control Register.
- Set the suitable clock prescaler value into the Async Clock Prescaler Register to scale the baud rate of the asynchronous output (this can be done automatically by the debugging tool).

13.7.7 IEEE 1149.1 JTAG Boundary Scan

With IEEE 1149.1 JTAG Boundary Scan, the pin-level access is independent of the device packaging technology.

IEEE 1149.1 JTAG Boundary Scan is enabled when TST is tied low, while JTAGSEL is high and INTEST7 is tied low during the power-up, and must be kept in this state during the whole boundary scan operation. The SAMPLE, EXTEST and BYPASS functions are implemented. In SWD/JTAG Debug mode, the ARM processor responds with a non-JTAG chip ID that identifies the processor. This is not IEEE 1149.1 JTAG-compliant.

It is not possible to switch directly between JTAG Boundary Scan and SWJ Debug Port operations. A chip reset must be performed after JTAGSEL is changed.

A Boundary-scan Descriptor Language (BSDL) file is provided on Atmel's web site to set up the test.

13.7.7.1 JTAG Boundary-scan Register

The Boundary-scan Register (BSR) contains a number of bits which correspond to active pins and the 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, refer to Boundary Scan Description Language (BDSL) files available for the SAM4 Series.

13.7.8 ID Code Register

Access: Read-only

31	30	29	28	27	26	25	24
VERSION				PART NUMBER			

23	22	21	20	19	18	17	16
PART NUMBER							

15	14	13	12	11	10	9	8
PART NUMBER				MANUFACTURER IDENTITY			

7	6	5	4	3	2	1	0
MANUFACTURER IDENTITY							1

- **VERSION[31:28]: Product Version Number**

Set to 0x0.

- **PART NUMBER[27:12]: Product Part Number**

Chip Name	Chip ID
SAM4CP	0x05B34

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

Set to 0x01F.

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

Set to 0x1.

Chip Name	JTAG ID Code
SAM4CP	0x05B3_403F

14. SAM4CP Boot Program

14.1 Description

The SAM-BA Boot Program includes an array of programs used to download and/or upload data into the different memories of the product.

14.2 Hardware and Software Constraints

- SAM-BA Boot uses the first 4096 bytes of the SRAM for variables and stacks. The remaining available size can be used for user code.
- UART0 requirements: None.

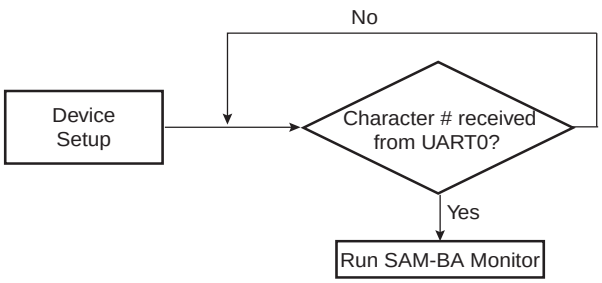
Table 14-1. Pins Driven during Boot Program Execution

Peripheral	Pin	PIO Line
UART0	URXD0	PB4
UART0	UTXD0	PB5

14.3 Flow Diagram

The Boot Program implements the algorithm depicted in Figure 14-1.

Figure 14-1. 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 whether the frequency is one of the supported external frequencies. If no clock or frequency other than one of the supported external frequencies is found, the internal 12-MHz RC oscillator is used as the main clock. The frequency drift of the 12-MHz RC oscillator should be taken into account for high precision applications.

14.4 Device Initialization

Initialization follows the steps described below:

1. Stack setup
2. Setup the Embedded Flash Controller
3. External clock detection (crystal or external clock on XIN)
4. Main oscillator frequency detection if no external clock detected
5. Switch master clock on main oscillator
6. C variable initialization
7. PLLB setup: PLLB clock is initialized to generate a 48 MHz on MCK
8. Disable the watchdogs

9. Initialization of UART0 (115200 bauds, 8, N, 1)
10. Wait for a character on UART0
11. Jump to SAM-BA monitor (see Section 14.5 ”SAM-BA Monitor”)

14.5 SAM-BA Monitor

The SAM-BA boot principle: once the communication interface is identified, running in an infinite loop waiting for different commands as shown in Table 14-2.

Table 14-2. 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>+'>'.

14.5.1 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 14.2 ”Hardware and Software Constraints”.

14.5.2 Xmodem Protocol

The supported Xmodem protocol 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 a successful transmission. Each block of the transfer looks like:

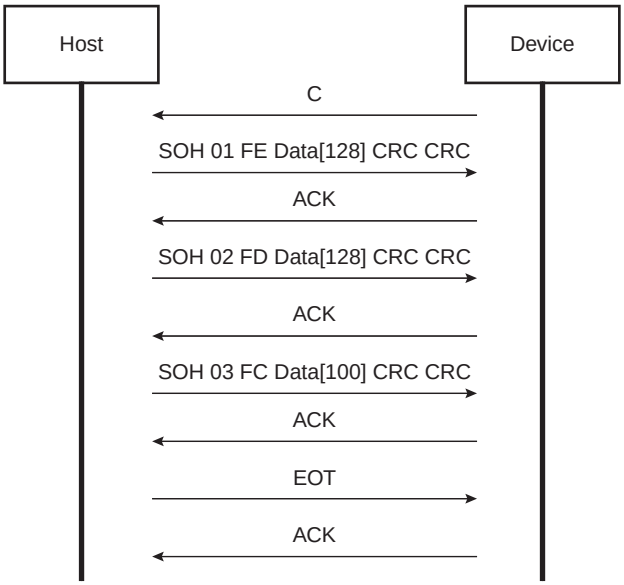
<SOH><blk #><255-blk #><--128 data bytes--><checksum>

where:

- <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 14-2 shows a transmission using this protocol.

Figure 14-2. Xmodem Transfer Example



14.5.3 In Application Programming (IAP) Feature

The IAP feature is a function located in the ROM that can be called by any software application.

When called, IAP sends the required 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 register).

Since this function is executed from ROM, Flash programming (such as sector write) can be performed by code running in Flash.

The IAP function entry point is retrieved by reading the NMI vector in the ROM (0x02000008).

This function takes one argument in parameter: the command to be sent to the EEFC.

This function returns the value of the EEFC_FSR register.

IAP software code example:

```
(unsigned int) (*IAP_Function)(unsigned long);
void main (void){

                  unsigned long FlashSectorNum = 200; //
                  unsigned long flash_cmd = 0;
                  unsigned long flash_status = 0;
                  unsigned long EFCIndex = 0; // 0:EEFC0, 1: EEFC1

/* Initialize the function pointer (retrieve function address from NMI
vector) */

                  IAP_Function = ((unsigned long) (*)(unsigned long))
0x02000008;

/* 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);

}
```

15. Reset Controller (RSTC)

15.1 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.

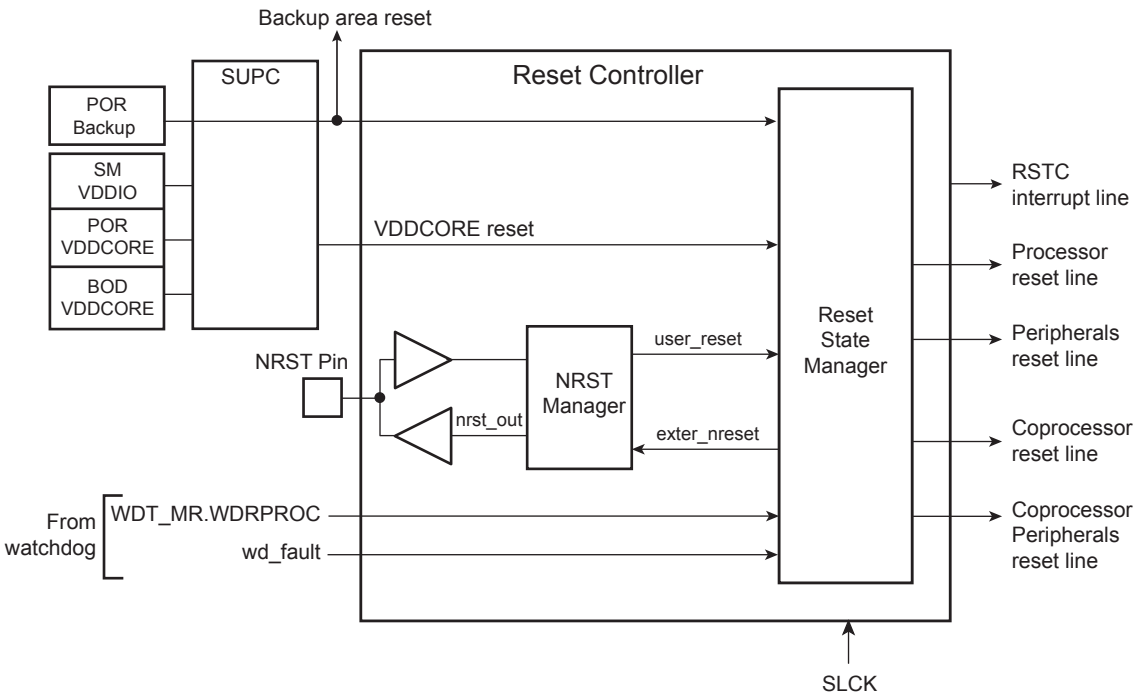
The RSTC also drives independently or simultaneously the external reset and the peripheral and processor resets.

15.2 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 and Coprocessor (second processor)
  - Peripheral Set
- Reset Source Status
  - Status of the Last Reset
  - Either VDDCORE and VDDBU POR Reset, Software Reset, User Reset, Watchdog Reset
- External Reset Signal Control and Shaping

15.3 Block Diagram

Figure 15-1. Reset Controller Block Diagram



15.4 Functional Description

15.4.1 Overview

The RSTC is made up of an NRST Manager and a Reset State Manager. It runs at SLCK frequency and generates the following reset signals:

- proc_nreset: processor reset line (also resets the Watchdog Timer)
- coproc_nreset: coprocessor (second processor) reset line
- periph_nreset: affects the whole set of embedded peripherals
- coproc_periph_nreset: affects the whole set of embedded peripherals driven by the Coprocessor
- nrst_out: drives the NRST pin

These reset signals are asserted by the RSTC, 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.

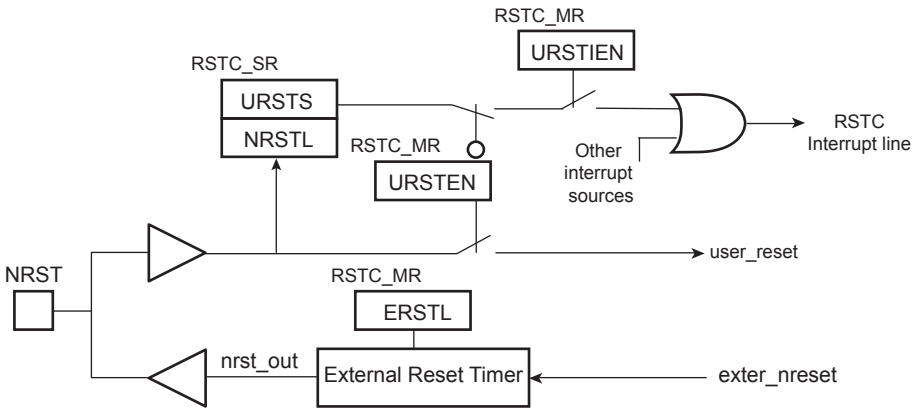
The NRST Manager shapes the NRST assertion during a programmable time, thus controlling external device resets.

The RSTC Mode register (RSTC_MR), used to configure the RSTC, is powered with VDDBU, so that its configuration is saved as long as VDDBU is on.

15.4.2 NRST Manager

The NRST Manager samples the NRST input pin and drives this pin low when required by the Reset State Manager. Figure 15-2 shows the block diagram of the NRST Manager.

Figure 15-2. NRST Manager



15.4.2.1 NRST Signal or Interrupt

The NRST Manager samples the NRST pin at SLCK speed. When the NRST line is low for more than three clock cycles, 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 RSTC_MR.URSTEN disables the User Reset trigger.

The level of the pin NRST can be read at any time in the bit NRSTL in the RSTC Status Register (RSTC_SR). As soon as the NRST pin is asserted, RSTC_SR.URSTS is written to ‘1’. This bit is cleared only when the RSTC_SR is read.

The RSTC can also be programmed to generate an interrupt instead of generating a reset. To do so, RSTC_MR.URSTIEN must be set.

15.4.2.2 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 RSTC_MR.ERSTL. This assertion duration, named External Reset Length, lasts $2^{(ERSTL+1)}$ SLCK 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 RSTC 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 value of ERSTL to shape the system power-up reset for devices requiring a longer startup time than that of the MCU.

15.4.3 Reset States

The Reset State Manager handles the different reset sources and generates the internal reset signals. It reports the reset status in RSTTYP of the Status Register (RSTC_SR). The update of RSTC_SR.RSTTYP is performed when the processor reset is released.

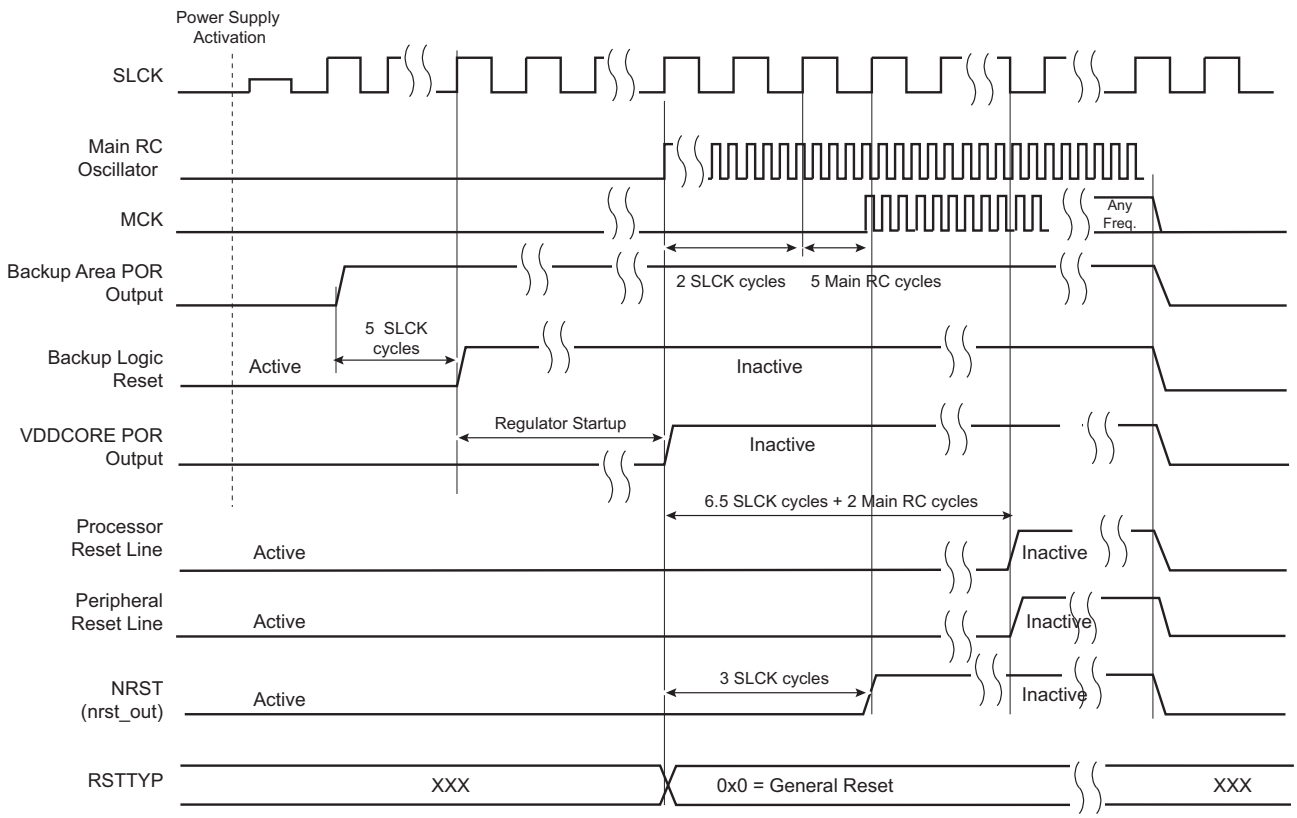
15.4.3.1 General Reset

A general reset occurs when a VDDBU power-on-reset is detected, a Brownout or a Voltage regulation loss is detected by the Supply controller. The vddcore_nreset signal is asserted by the Supply Controller when a general reset occurs.

All the reset signals are released and 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 15-3 shows how the General Reset affects the reset signals.

Figure 15-3. General Reset Timing Diagram



15.4.3.2 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.

15.4.3.3 Watchdog Reset

The Watchdog Reset is entered when a watchdog fault occurs. This reset lasts three SLCK cycles.

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

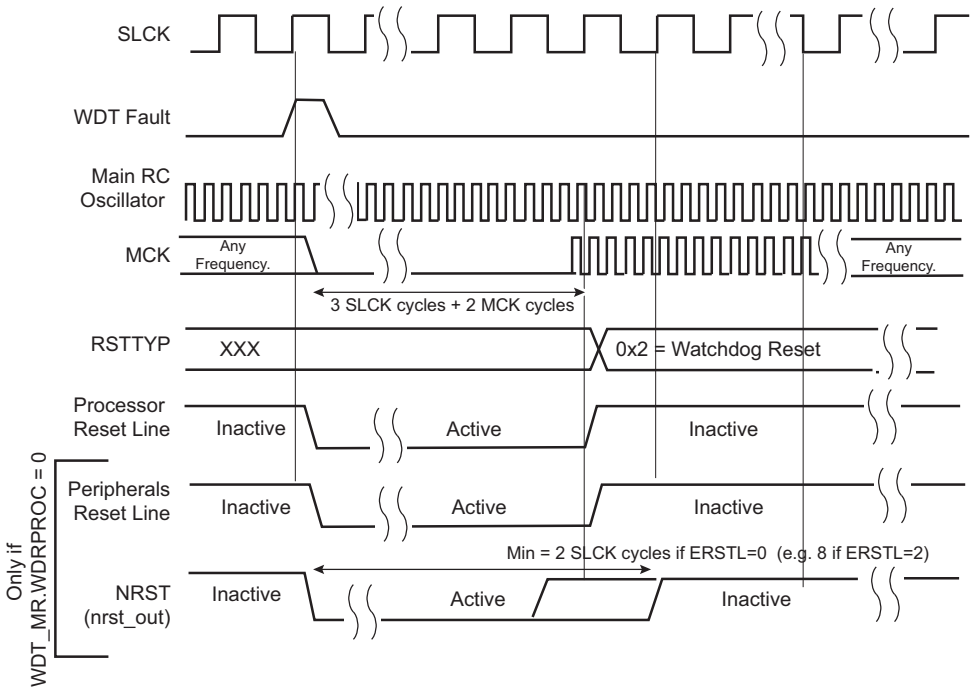
- If WDRPROC = 0, the Processor Reset and the Peripheral Reset are asserted. The NRST line is also asserted, depending on how field RSTC_MR.ERSTL is programmed. However, the resulting low level on NRST does not result in a User Reset state.
- If WDRPROC = 1, only the processor reset is asserted.

The Watchdog Timer is reset by the proc_nreset signal. As the watchdog fault always causes a processor reset if WDT_MR.WDRSTEN 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 WDT_MR.WDRSTEN is written to '0', the watchdog fault has no impact on the RSTC.

After a watchdog overflow occurs, the report on the RSTC_SR.RSTTYP 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 15-4. Watchdog Reset Timing Diagram



15.4.3.4 Software Reset

The RSTC offers commands to assert the different reset signals. These commands are performed by writing the Control register (RSTC_CR) or Coprocessor Mode register (RSTC_CPMR) 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 associated to processor whereas the coprocessor peripherals are not reset, 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_CPMR.CPROCEN: Writing a ‘0’ to CPROCEN resets the coprocessor only.
- RSTC_CPMR.CPEREN: Writing a ‘0’ to CPEREN resets all the embedded peripherals associated to coprocessor whereas the processor peripherals are not reset.
- 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 SLCK cycles.

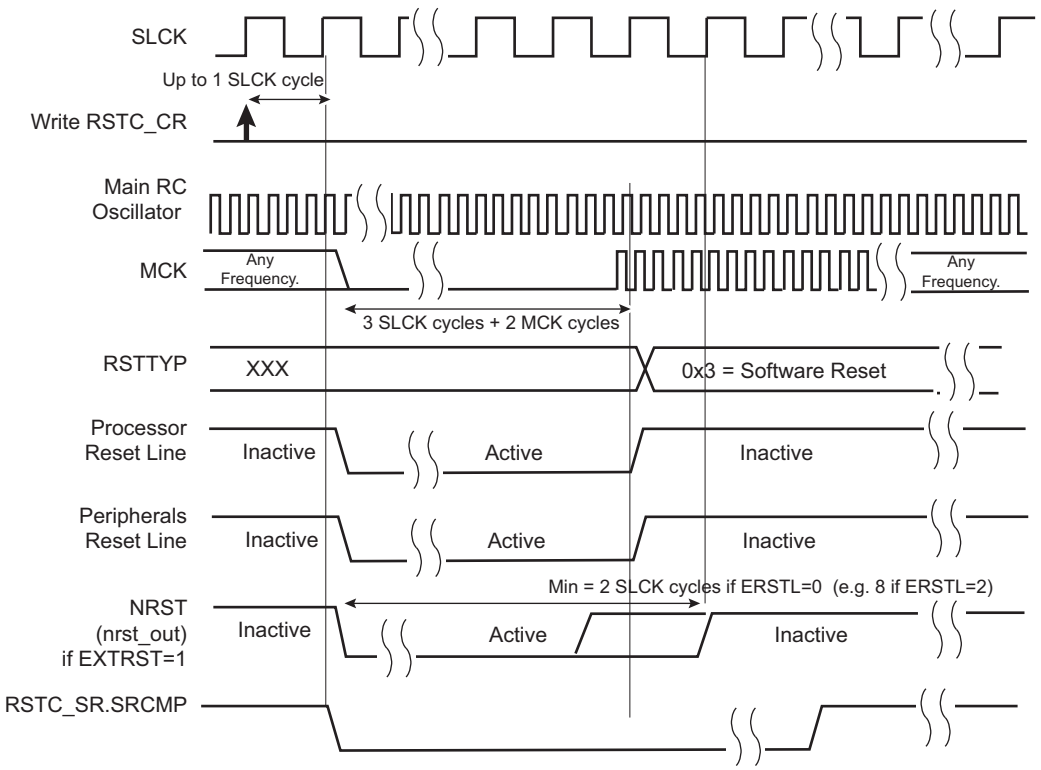
The internal reset signals are asserted as soon as the register write is performed. This is detected on the Master Clock (MCK). They are released when the software reset has ended, i.e., synchronously to SLCK.

If EXTRST is written to ‘1’, the nrst_out signal is asserted depending on the configuration of RSTC_MR.ERSTL. However, the resulting falling edge on NRST does not lead to a User Reset.

If and only if the RSTC_CR.PROCRST is written to ‘1’, the RSTC 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, RSTC_SR.SRCMP is written to ‘1’. SRCMP is cleared at the end of the software reset. No other software reset can be performed while SRCMP is written to ‘1’, and writing any value in the RSTC_CR has no effect.

Figure 15-5. Software Reset Timing Diagram



15.4.3.5 User Reset

A User Reset is generated when a low level is detected on the NRST pin and RSTC_MR.URSTEN 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.

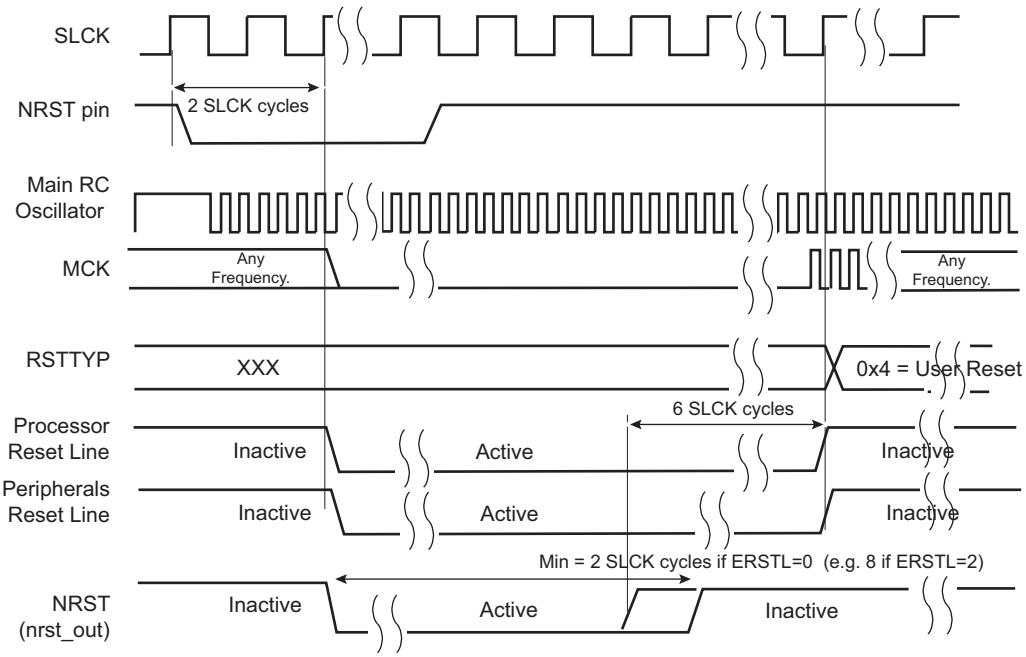
The User Reset is triggered 2 SLCK cycles after a low level is detected on NRST. The Processor and Coprocessor Reset and the Peripheral Resets are asserted.

The User Reset ends when NRST rises, after a two-cycle resynchronization time and a three-cycle processor startup. The processor clock is reenabled as soon as NRST is confirmed high.

When the processor reset signal is released, RSTC_SR.RSTTYP is loaded with the value ‘4’, indicating a User Reset.

The NRST Manager guarantees that the NRST line is asserted for External Reset Length SLCK cycles, as configured in 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 15-6. User Reset Timing Diagram



15.4.4 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:

- General Reset
- Backup Reset
- Watchdog Reset
- Software Reset
- User Reset

Specific cases are listed below:

- When in User Reset:
  - A watchdog event is impossible because the Watchdog Timer is being reset by the proc_nreset signal.
  - A software reset is impossible, since the processor reset is being activated.
- When in Software Reset:
  - A watchdog event has priority over the current state.
  - The NRST has no effect.
- When in Watchdog Reset:
  - The processor reset is active and so a Software Reset cannot be programmed.
  - A User Reset cannot be entered.

15.4.5 Managing Reset at Application Level

As described, the device embeds only one system and power management controller shared by the two subsystems, i.e.:

- Subsystem 0 (SUB-S0 - Application Master/Core)
- Subsystem 1 (SUB-S1 - Metrology/Co-Processor Core)

After a power-up, the SUB-S0 application configures the SUB-S1 system clock (PMC). Then, the application code can be downloaded into the SUB-S1 boot memory (SRAM1) and SUB-S0 can afterwards deassert the SUB-S1 reset lines through the RSTC.

Once the two subsystems are up and running, each one executes its firmware independently. In some application use case, Subsystem 1 must not be reset even if Subsystem 0 is allowed to, due to a firmware upgrade or due to a reset request from one of the following reset sources:

- a User Reset (NRST pin), a Software Reset, a VDDIO Supply Monitor Reset
- a Watchdog Reset

Since the PMC can be reset from one of the above reset sources, if a reset occurs, the PMC is reset, leading to switching off the SUB-S1 clocks.

To avoid this, the SUB-S0 application must be in charge of the reset management of the complete system in the following manner:

1. User Reset (NRST pin) and VDDIO Supply Monitor Reset must be configured to generate an interrupt, and not a reset. This allows to reset only the SUB-S0 processor and not the peripherals (so, not the PMC). This is mandatory to avoid any stop of a metrology part due to a reset on clocks.
2. Watchdog Reset must be configured to generate a SUB-S0 processor reset only.

Note: The Core Brownout detector reset (general reset source) and the Backup reset are not taken into account in the reset management considerations described above, as they are related, respectively, to a power loss detection or to a wake-up from Backup mode or low level on VDDBU. (Note that in Backup mode, all digital logic, except the backup zone, is shut down).

15.5 Reset Controller (RSTC) User Interface

Table 15-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	RSTC_CR	Write-only	–
0x04	Status Register	RSTC_SR	Read-only	0x0000_0000(1)
0x08	Mode Register	RSTC_MR	Read/Write	0x0000 0001
0x0C	Coprocessor Mode Register	RSTC_CPMR	Read/Write	0x0000_0000

Notes: 1. This value assumes that a general reset has been performed, subject to change if other types of reset are generated.

15.5.1 RSTC Control Register

Name: RSTC_CR

Address: 0x400E1400

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 = 0xA5, resets the processor.

- **PERRST: Peripheral Reset**

0: No effect.

1: If KEY = 0xA5, resets the peripherals.

- **EXTRST: External Reset**

0: No effect.

1: If KEY = 0xA5, asserts the NRST pin.

- **KEY: System Reset Key**

Value	Name	Description
0xA5	PASSWD	Writing any other value in this field aborts the write operation.

15.5.2 RSTC Status Register

Name: RSTC_SR

Address: 0x400E1404

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. This transition is also detected on the MCK rising edge. If the user reset is disabled (URSTEN = 0 in RSTC_MR) and if the interrupt is enabled by RSTC_MR.URSTIEN, URSTS triggers an interrupt. Reading the RSTC_SR resets URSTS 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**

Registers the NRST pin level sampled on each 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 RSTC. The RSTC is ready for a software command.

1: A software reset command is being performed by the RSTC. The RSTC is busy.

15.5.3 RSTC Mode Register

Name: RSTC_MR

Address: 0x400E1408

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: RSTC_SR.USRTS at ‘1’ has no effect on the RSTC interrupt line.

1: RSTC_SR.USRTS at ‘1’ asserts the RSTC interrupt line if URSTEN = 0.

- **ERSTL: External Reset Length**

This field defines the external reset length. The external reset is asserted during a time of $2^{(ERSTL+1)}$ SLCK 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.

15.5.4 RSTC Coprocessor Mode Register

Name: RSTC_CPMR

Address: 0x400E140C

Access: Read/Write

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

- **CPROCEN: Coprocessor (Second Processor) Enable**

0: If CPKEY = 0x5A, resets the coprocessor (power-on default value).

1: If CPKEY = 0x5A, deasserts the reset of the coprocessor.

- **CPEREN: Coprocessor Peripheral Enable**

0: If CPKEY = 0x5A, resets the coprocessor peripherals.

1: If CPKEY = 0x5A, deasserts the reset of the coprocessor peripherals.

- **CPKEY: Coprocessor System Enable Key**

Value	Name	Description
0x5A	PASSWD	Writing any other value in this field aborts the write operation.

16. Real-time Timer (RTT)

16.1 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 1 Hz RTC signal, thus taking advantage of a calibrated 1 Hz clock.

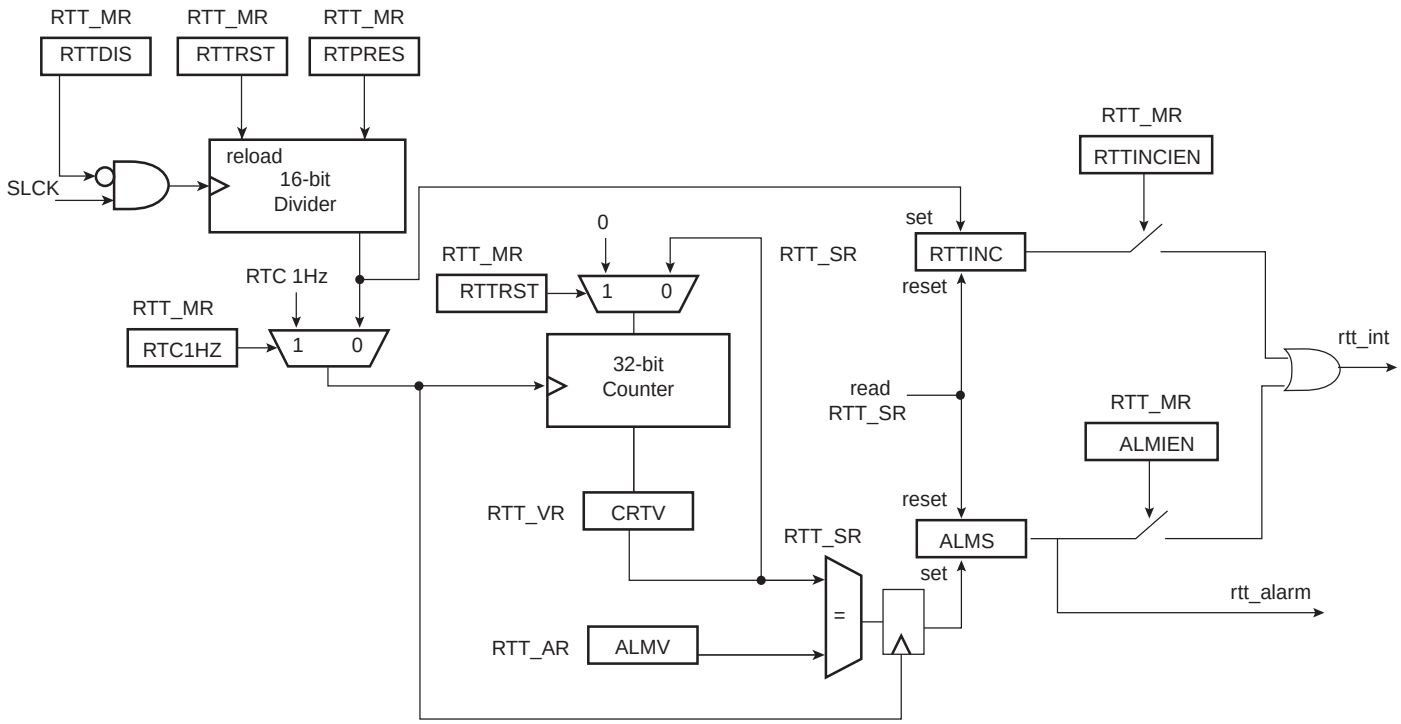
The slow clock source can be fully disabled to reduce power consumption when only an elapsed seconds count is required.

16.2 Embedded Characteristics

- 32-bit Free-running Counter on prescaled slow clock or RTC calibrated 1 Hz clock
- 16-bit Configurable Prescaler
- Interrupt on Alarm or Counter Increment

16.3 Block Diagram

Figure 16-1. Real-time Timer



16.4 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 16-2).

The real-time 32-bit counter can also be supplied by the 1 Hz 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 16-1).

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.

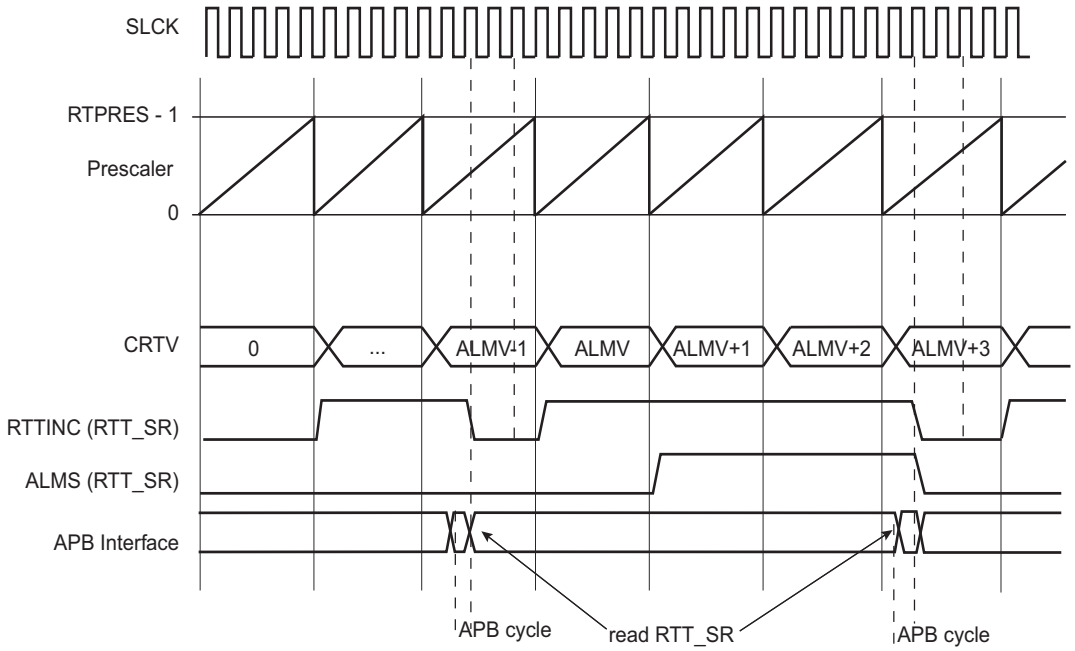
The RTTINCIEN bit must be cleared prior to writing a new RTPRES value in the RTT_MR.

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 16-2. RTT Counting



16.5 Real-time Timer (RTT) User Interface

Table 16-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Mode Register	RTT_MR	Read/Write	0x0000_8000
0x04	Alarm Register	RTT_AR	Read/Write	0xFFFF_FFFF
0x08	Value Register	RTT_VR	Read-only	0x0000_0000
0x0C	Status Register	RTT_SR	Read-only	0x0000_0000

16.5.1 Real-time Timer Mode Register

Name: RTT_MR

Address: 0x400E1430

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 1 Hz RTC clock.

Note: RTC1HZ is write only.

16.5.2 Real-time Timer Alarm Register

Name: RTT_AR

Address: 0x400E1434

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 16-2).

Note: The alarm interrupt must be disabled (ALMIEN must be cleared in RTT_MR) when writing a new ALMV value.

16.5.3 Real-time Timer Value Register

Name: RTT_VR

Address: 0x400E1438

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.

16.5.4 Real-time Timer Status Register

Name: RTT_SR

Address: 0x400E143C

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.

17. Real-time Clock (RTC)

17.1 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.

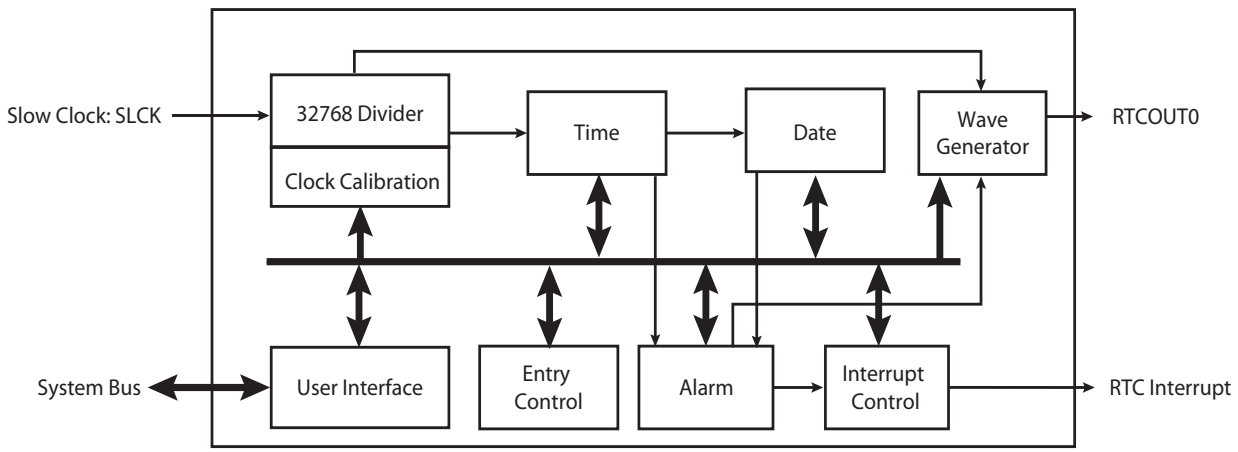
Timestamping capability reports the first and last occurrences of tamper events.

17.2 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
- Tamper Timestamping Registers
- Register Write Protection

17.3 Block Diagram

Figure 17-1. Real-time Clock Block Diagram



17.4 Product Dependencies

17.4.1 Power Management

The Real-time Clock is continuously clocked at 32.768 kHz. The Power Management Controller has no effect on RTC behavior.

17.4.2 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 17-1. Peripheral IDs

Instance	ID
RTC	2

17.5 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 output.

17.5.1 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.

17.5.2 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.

17.5.3 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.

17.5.4 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.

17.5.5 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 RTC 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.

17.5.6 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 17-2 ”Time/Calendar Update Timing Diagram”

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 17-2. Time/Calendar Update Timing Diagram

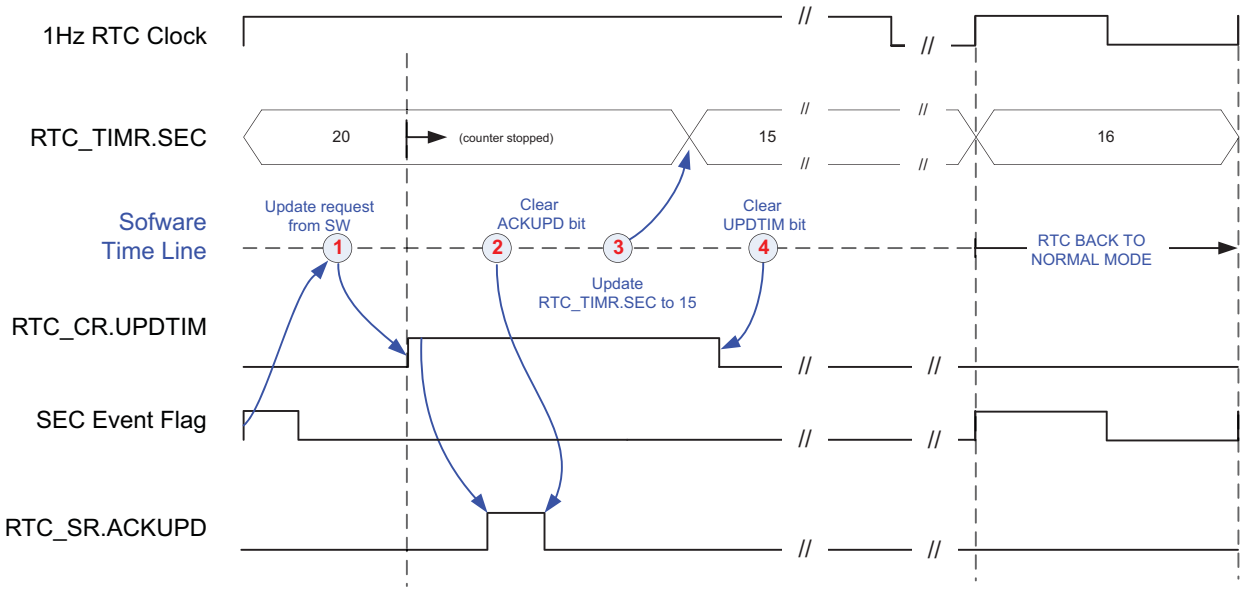
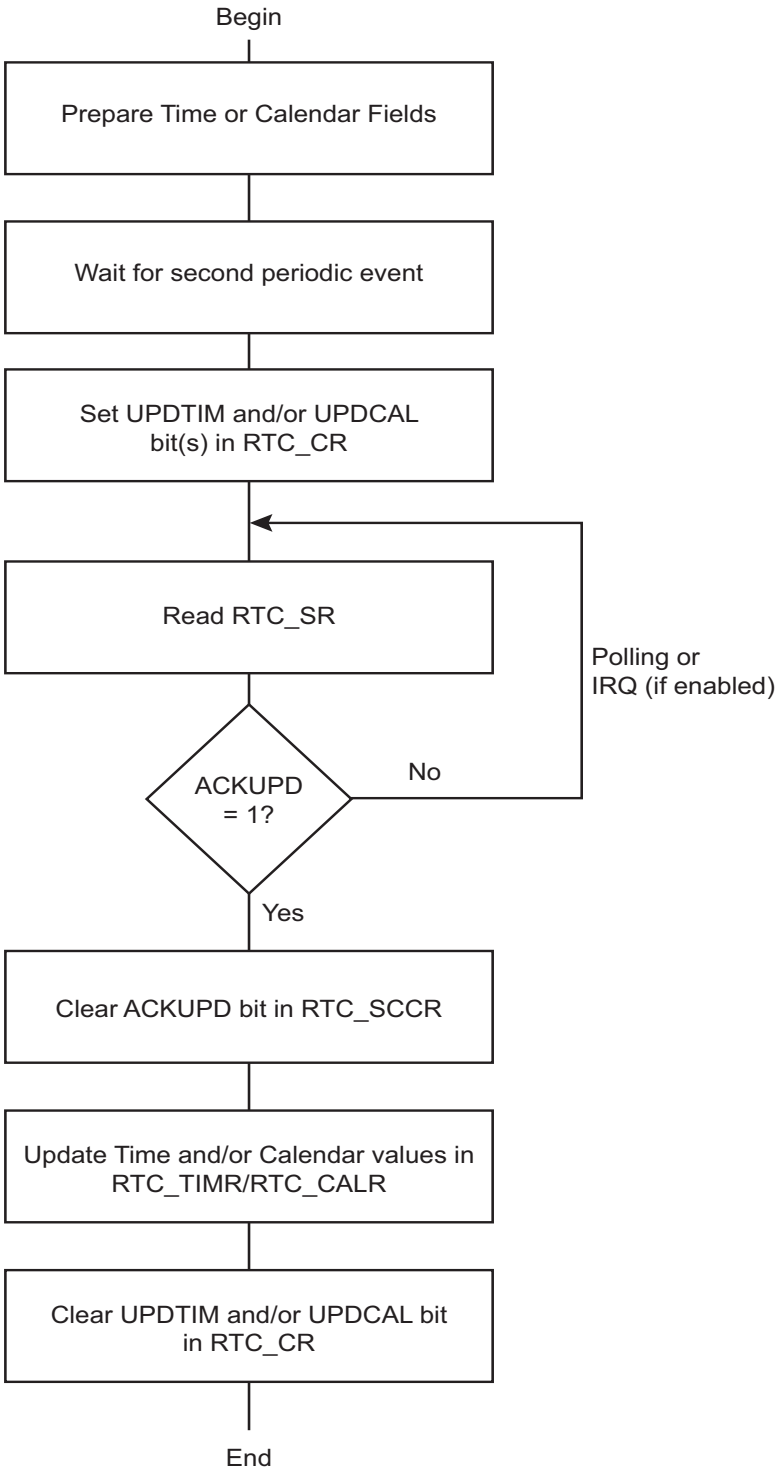


Figure 17-3. Update Sequence



17.5.7 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 17-4. Calibration Circuitry

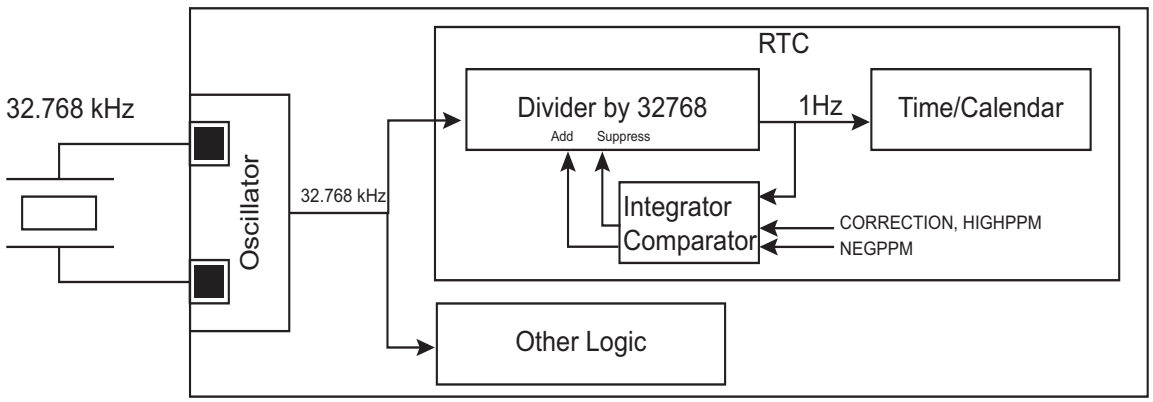
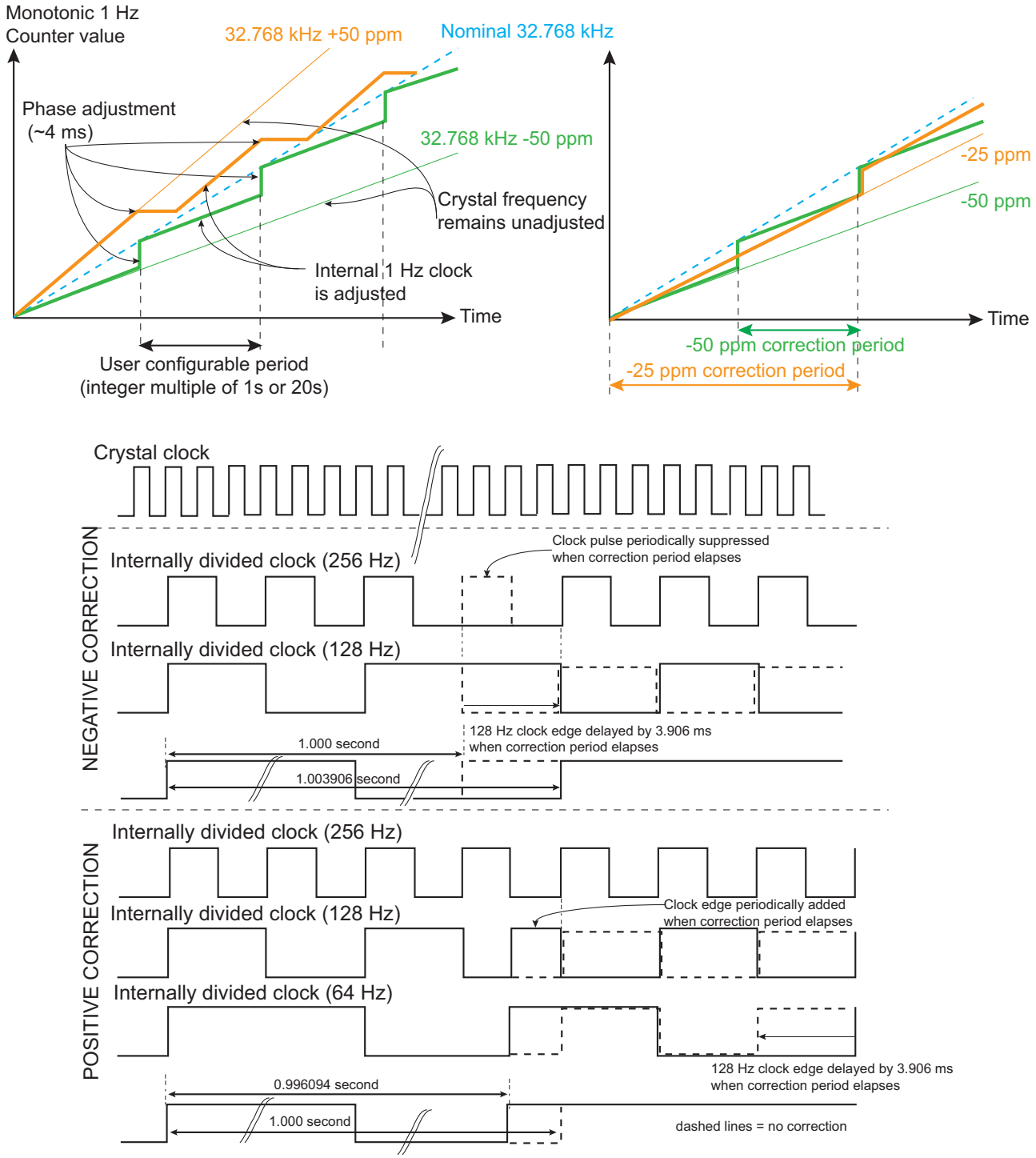


Figure 17-5. Calibration Circuitry Waveforms



The inaccuracy of a crystal oscillator at typical room temperature (±20 ppm at 20 - 25 ºC) 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.

17.5.8 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 output (RTCOUT0) has 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.

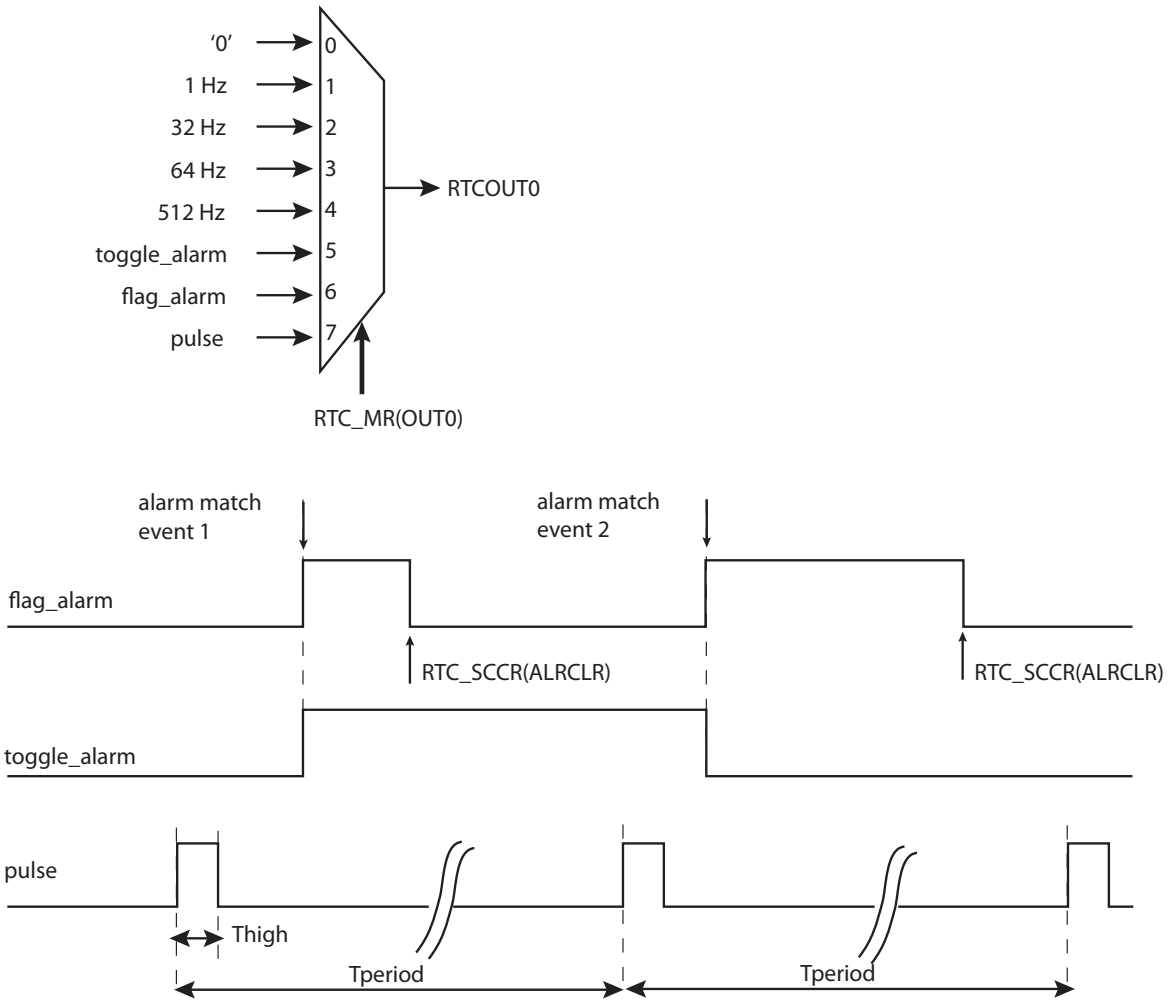
Selection choice 5 provides a toggling signal when the RTC alarm is reached.

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 line associated to RTC output is automatically selecting these waveforms as soon as RTC_MR corresponding fields OUT0 differ from 0.

Figure 17-6. Waveform Generation



17.5.9 Tamper Timestamping

As soon as a tamper is detected, the tamper counter is incremented and the RTC stores the time of the day, the date and the source of the tamper event in registers located in the backup area. Up to two tamper events can be stored.

The tamper counter saturates at 15. Once this limit is reached, the exact number of tamper occurrences since the last read of stamping registers cannot be known.

The first set of timestamping registers (RTC_TSTR0, RTC_TSDR0, RTC_TSSR0) cannot be overwritten, so once they have been written all data are stored until the registers are reset.Therefore these registers are storing the first tamper occurrence after a read.

The second set of timestamping registers (RTC_TSTR1, RTC_TSDR1, RTC_TSSR1) are overwritten each time a tamper event is detected. Thus the date and the time data of the first and the second stamping registers may be equal. This occurs when the tamper counter value carried on field TEVCNT in RTC_TSTR0 equals 1. Thus this second set of registers stores the last occurrence of tamper before a read.

Reading a set of timestamping registers requires three accesses, one for the time of the day, one for the date and one for the tamper source.

Reading the third part (RTC_TSSR0/1) of a timestamping register set clears the whole content of the registers (time, date and tamper source) and makes the timestamping registers available to store a new event.

17.6 Real-time Clock (RTC) User Interface

Table 17-2. 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	0x01E111220
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
0xB0	TimeStamp Time Register 0	RTC_TSTR0	Read-only	0x00000000
0xB4	TimeStamp Date Register 0	RTC_TSDR0	Read-only	0x00000000
0xB8	TimeStamp Source Register 0	RTC_TSSR0	Read-only	0x00000000
0xBC	TimeStamp Time Register 1	RTC_TSTR1	Read-only	0x00000000
0xC0	TimeStamp Date Register 1	RTC_TSDR1	Read-only	0x00000000
0xC4	TimeStamp Source Register 1	RTC_TSSR1	Read-only	0x00000000
0xC8	Reserved	–	–	–
0xCC	Reserved	–	–	–
0xD0	Reserved	–	–	–
0xD4 – 0xE0	Reserved	–	–	–
0xE4	Write Protection Mode Register	RTC_WPMR	Read/Write	0x00000000
0xE8 – 0xF8	Reserved	–	–	–
0xFC	Reserved	–	–	–

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

17.6.1 RTC Control Register

Name: RTC_CR

Address: 0x400E1460

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 “RTC Write Protection Mode Register”

- **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

17.6.2 RTC Mode Register

Name: RTC_MR

Address: 0x400E1464

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	TPERIOD		–	THIGH		

23	22	21	20	19	18	17	16
–	–	–	–	–	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 “RTC Write Protection Mode Register”

- **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 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
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

17.6.3 RTC Time Register

Name: RTC_TIMR

Address: 0x400E1468

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.

17.6.4 RTC Calendar Register

Name: RTC_CALR

Address: 0x400E146C

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.

17.6.5 RTC Time Alarm Register

Name: RTC_TIMALR

Address: 0x400E1470

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 “RTC Write Protection Mode Register”

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.

17.6.6 RTC Calendar Alarm Register

Name: RTC_CALALR

Address: 0x400E1474

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 “RTC Write Protection Mode Register”

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.

17.6.7 RTC Status Register

Name: RTC_SR

Address: 0x400E1478

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.

17.6.8 RTC Status Clear Command Register

Name: RTC_SCCR

Address: 0x400E147C

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).

17.6.9 RTC Interrupt Enable Register

Name: RTC_IER

Address: 0x400E1480

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.

17.6.10 RTC Interrupt Disable Register

Name: RTC_IDR

Address: 0x400E1484

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.

17.6.11 RTC Interrupt Mask Register

Name: RTC_IMR

Address: 0x400E1488

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.

17.6.12 RTC Valid Entry Register

Name: RTC_VER

Address: 0x400E148C

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.

17.6.13 RTC TimeStamp Time Register 0

Name: RTC_TSTR0

Address: 0x400E1510

Access: Read-only

31	30	29	28	27	26	25	24
BACKUP	–	–	–	TEVCNT			

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						

RTC_TSTR0 reports the timestamp of the first tamper event after reading RTC_TSSR0.

This register is cleared by reading RTC_TSSR0.

- **SEC: Seconds of the Tamper**
- **MIN: Minutes of the Tamper**
- **HOUR: Hours of the Tamper**
- **AMPM: AM/PM Indicator of the Tamper**
- **TEVCNT: Tamper Events Counter**

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**

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.

17.6.14 RTC TimeStamp Time Register 1

Name: RTC_TSTR1

Address: 0x400E151C

Access: Read-only

31	30	29	28	27	26	25	24
BACKUP	–	–	–	–	–	–	–

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						

RTC_TSTR1 reports the timestamp of the last tamper event.

This register is cleared by reading RTC_TSSR1.

- **SEC: Seconds of the Tamper**
- **MIN: Minutes of the Tamper**
- **HOUR: Hours of the Tamper**
- **AMPM: AM/PM Indicator of the Tamper**
- **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.

17.6.15 RTC TimeStamp Date Register

Name: RTC_TSDRx

Address: 0x400E1514 [0], 0x400E1520 [1]

Access: Read-only

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						

RTC_TSDR0 reports the timestamp of the first tamper event after reading RTC_TSSR0, and RTC_TSDR1 reports the timestamp of the last tamper event.

This register is cleared by reading RTC_TSSR.

- **CENT: Century of the Tamper**
- **YEAR: Year of the Tamper**
- **MONTH: Month of the Tamper**
- **DAY: Day of the Tamper**
- **DATE: Date of the Tamper**

The fields contain the date and the source of a tamper occurrence if the TEVCNT is not null.

17.6.16 RTC TimeStamp Source Register

Name: RTC_TSSRx

Address: 0x400E1518 [0], 0x400E1524 [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
–	–	–	–	–	–	TSRC	

- **TSRC: Tamper Source**

This field contains the tamper source. It is valid only if the TEVCNT is not null.

This register is cleared after read and the read access also performs a clear on RTC_TSTRx and RTC_TSDRx.

17.6.17 RTC Write Protection Mode Register

Name: RTC_WPMR

Address: 0x400E1544

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).

The following registers can be write-protected:

- “RTC Mode Register”
- “RTC Time Alarm Register”
- “RTC Calendar Alarm Register”

- **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.

18. Watchdog Timer (WDT)

18.1 Description

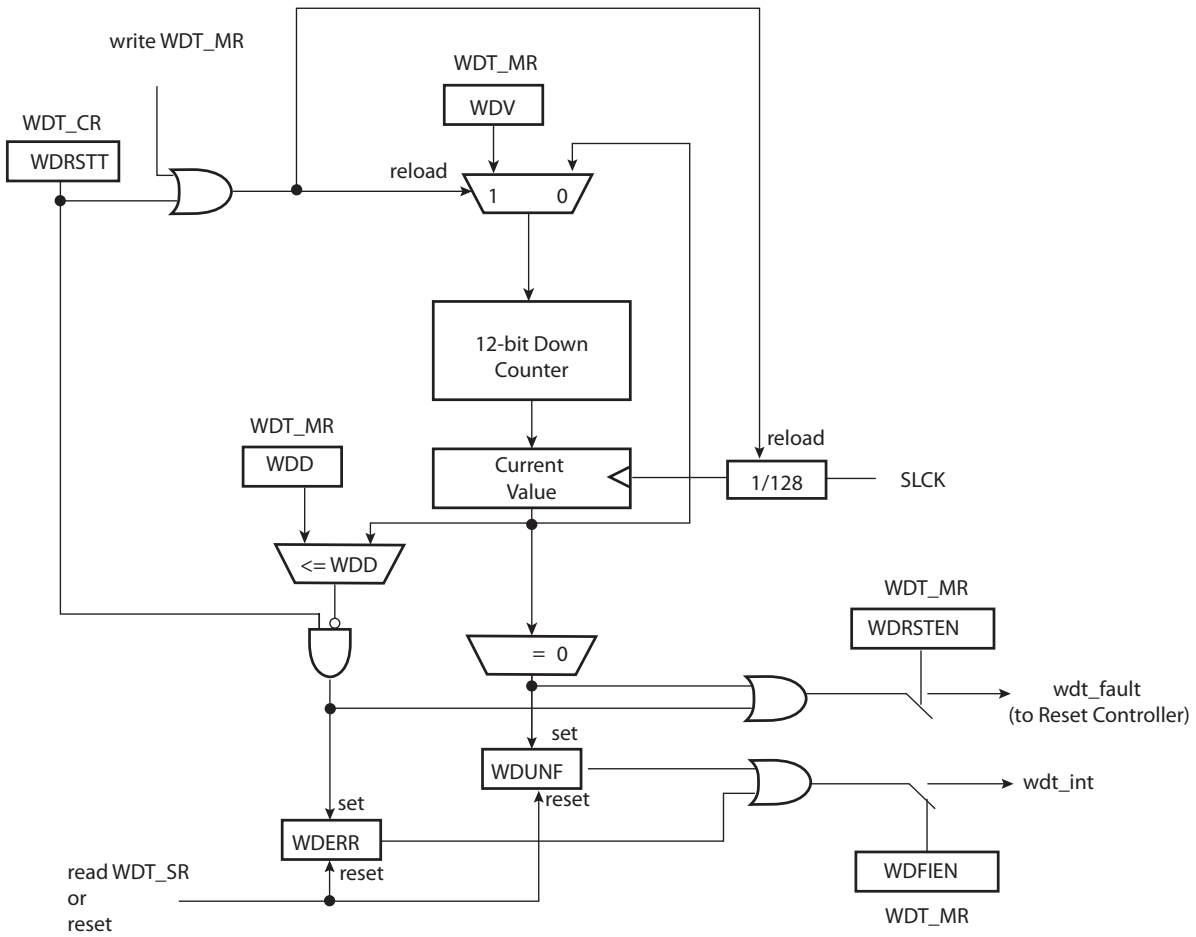
The Watchdog Timer (WDT) is used to prevent system lock-up if the software becomes trapped in a deadlock. It features a 12-bit down counter that allows a watchdog period of up to 16 seconds (slow clock around 32 kHz). It can generate a general reset or a processor reset only. In addition, it can be stopped while the processor is in Debug mode or Sleep mode (Idle mode).

18.2 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.

18.3 Block Diagram

Figure 18-1. Watchdog Timer Block Diagram



18.4 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.

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).

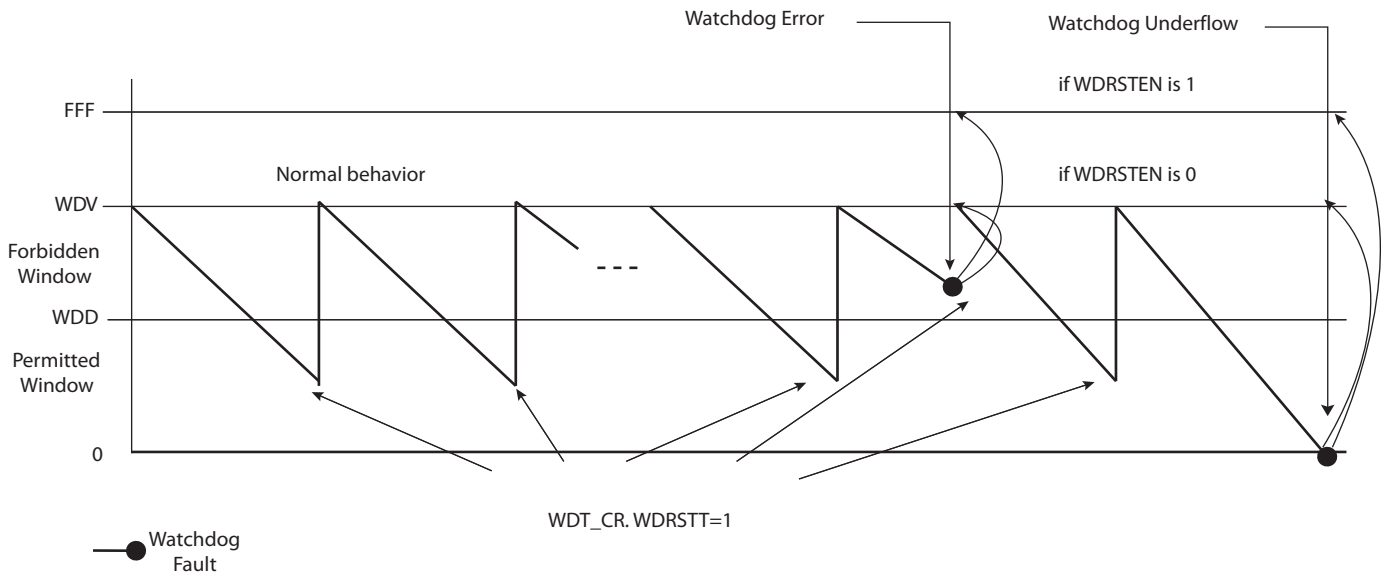
The status bits WDUNF (Watchdog Underflow) and WDERR (Watchdog Error) trigger an interrupt, provided the bit WDT_MR.WDFIEN is set. The signal “wdt_fault” to the reset controller causes a watchdog reset if the WDRSTEN bit is set as already explained in the reset controller documentation. In this case, 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 18-2. Watchdog Behavior



18.5 Watchdog Timer (WDT) User Interface

Table 18-1. 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

18.5.1 Watchdog Timer Control Register

Name: WDT_CR

Address: 0x400E1450

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

Notes: 1. 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.

18.5.2 Watchdog Timer Mode Register

Name: WDT_MR

Address: 0x400E1454

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.

- **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.

18.5.3 Watchdog Timer Status Register

Name: WDT_SR

Address: 0x400E1458

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.

19. Reinforced Safety Watchdog Timer (RSWDT)

19.1 Description

The Reinforced Safety Watchdog Timer (RSWDT) works in parallel with the Watchdog Timer (WDT) to reinforce safe watchdog operations.

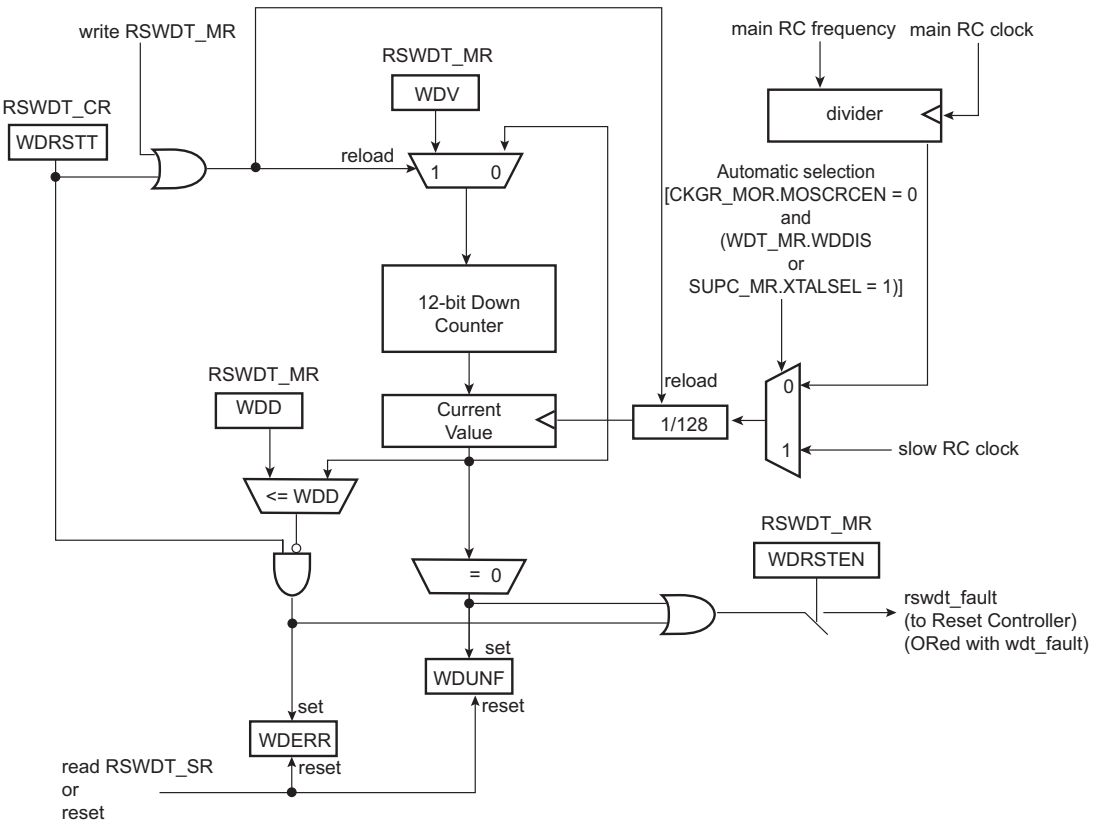
The RSWDT can be used to reinforce the safety level provided by the WDT in order to prevent system lock-up if the software becomes trapped in a deadlock. The RSWDT works in a fully operable mode, independent of the WDT. Its clock source is automatically selected from either the Slow RC oscillator clock, or from the 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 regardless 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.

19.2 Embedded Characteristics

- Automatically selected reliable RSWDT clock source (independent of WDT clock source).
- 12-bit key-protected programmable counter.
- Provides Reset Signal to the system.
- Counter may be stopped while processor is in debug state or idle mode.

19.3 Block Diagram

Figure 19-1. Reinforced Safety Watchdog Timer Block Diagram



19.4 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 RSWDT_MR.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 RSWDT_MR.WDRSTEN is set. Moreover, WDUNF (Watchdog Underflow) is set in the Status Register (RSWDT_SR).

The signal “wdt_fault” to the Reset Controller causes a Watchdog reset if the WDRSTEN bit. For details, refer to the section “Reset Controller (RSTC)”. In this case, the processor and the RSWDT are reset, and the WDUNF and WDERR flags are reset.

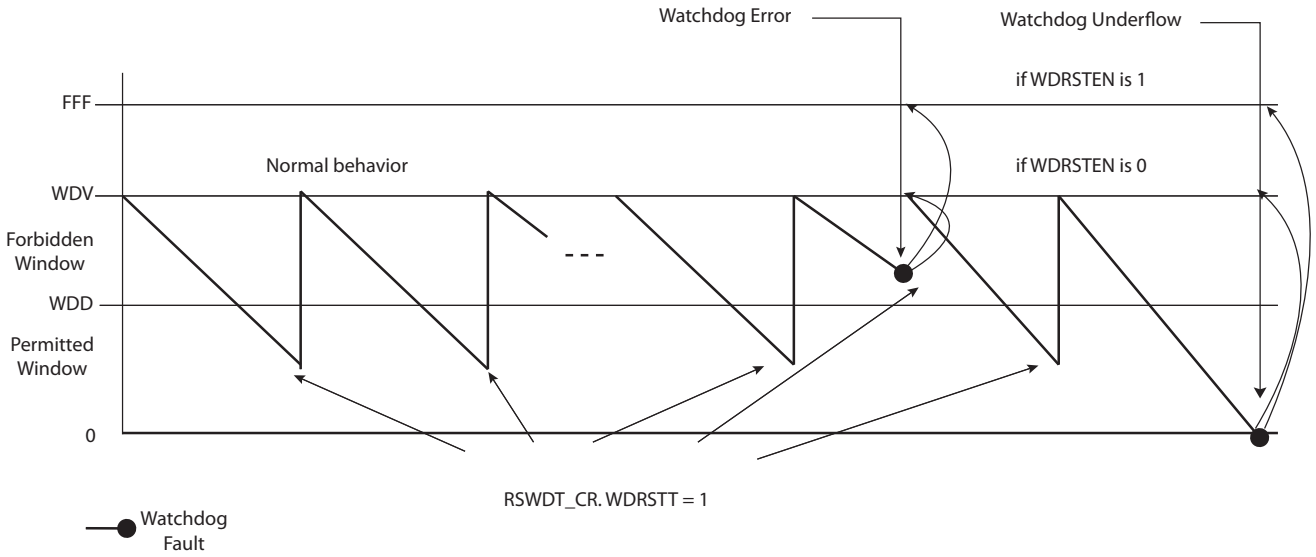
If a reset is generated, or if RSWDT_SR is read, the status bits are reset, and the “wdt_fault” signal to the reset controller is deasserted.

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 19-2. Watchdog Behavior



19.5 Reinforced Safety Watchdog Timer (RSWDT) User Interface

Table 19-1. 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

19.5.1 Reinforced Safety Watchdog Timer Control Register

Name: RSWDT_CR

Address: 0x400E1500

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.

19.5.2 Reinforced Safety Watchdog Timer Mode Register

Name: RSWDT_MR

Address: 0x400E1504

Access: Read/Write Once

31	30	29	28	27	26	25	24
–	–	WDIDLEHLT	WDDBGHLT	ALLONES			

23	22	21	20	19	18	17	16
ALLONES							

15	14	13	12	11	10	9	8
WDDIS	WDRPROC	WDRSTEN	-	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 WDV value 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.

- **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.

- **ALLONES: Must Always Be Written with 0xFFF**

- **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.

19.5.3 Reinforced Safety Watchdog Timer Status Register

Name: RSWDT_SR

Address: 0x400E1508

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
–	–	–	–	–	–	–	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.

20. Supply Controller (SUPC)

20.1 Description

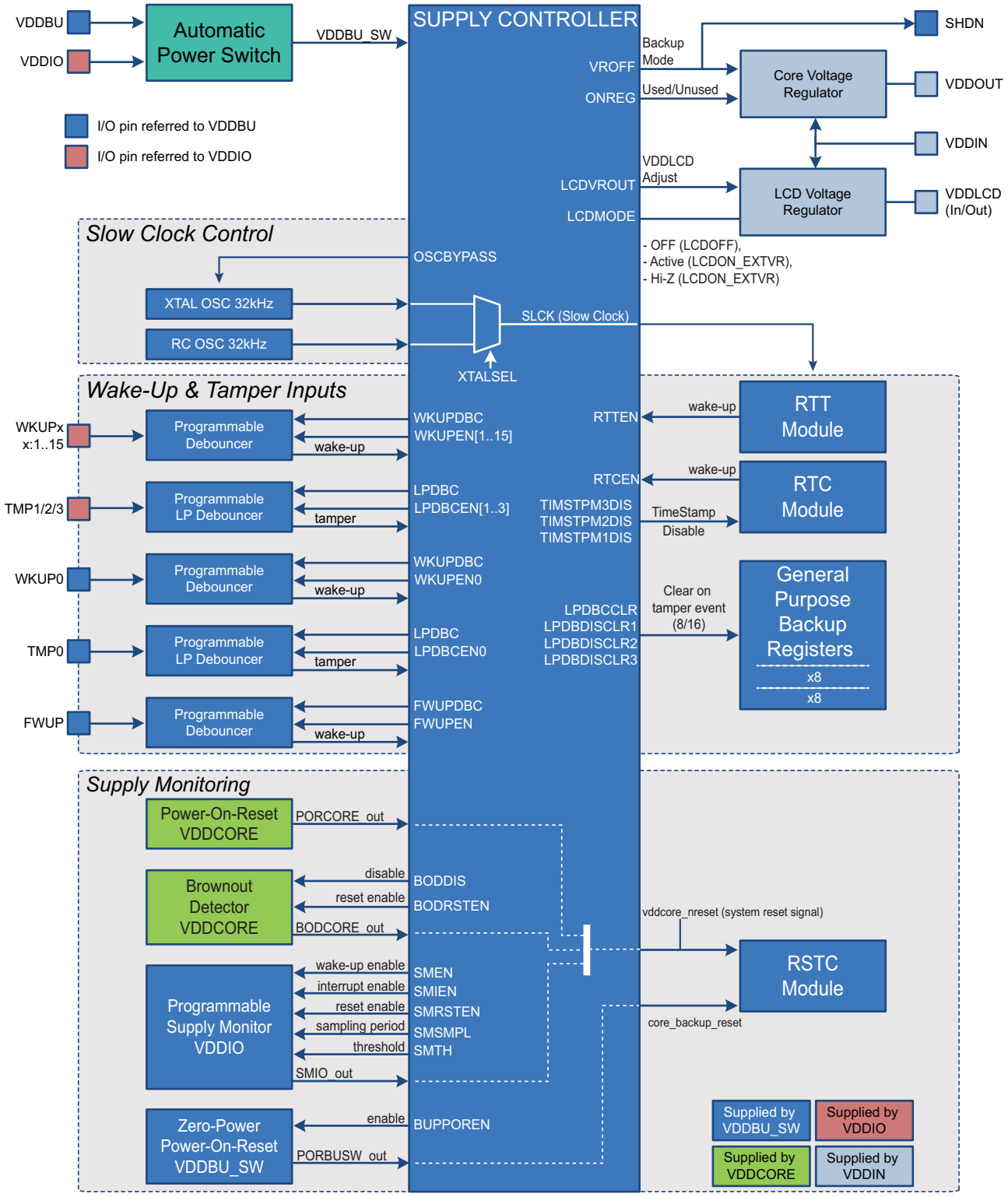
The Supply Controller (SUPC) controls the supply voltages of the system and manages the backup Low-power mode. In this mode, current consumption is reduced to less than 1 µA (typ) for backup power retention. Exit from this mode is possible on multiple wake-up sources. The SUPC also generates the slow clock by selecting either the low-power RC oscillator or the low-power crystal oscillator.

20.2 Embedded Characteristics

- Manages VDDCORE and the Backup Low-Power Mode by Controlling the Embedded Voltage Regulator
- Manages the LCD Power Supply VDDLCD and the Backup Low-Power Mode by Controlling the Embedded LCD Voltage Regulator
- A Supply Monitor Detection on VDDIO or a Brownout Detection on VDDCORE Triggers a System Reset
- A Zero-power Power-on Reset on VDDBU_SW Triggers a System Reset
- Generates the Slow Clock SLCK by Selecting Either the 32 kHz Low-Power RC Oscillator or the 32 kHz Low-Power Crystal Oscillator
- Supports Multiple Wake-up Sources for Exit from Backup Low-Power Mode
  - Force Wake-up Pin, with Programmable Debouncing
  - Up to 16 Wake-up Inputs (including Tamper Inputs), with Programmable Debouncing
  - Real-time Clock Alarm
  - Real-time Timer Alarm
  - Supply Monitor Detection on VDDIO with Programmable Scan Period and Voltage Threshold

20.3 Block Diagram

Figure 20-1. Supply Controller Block Diagram



Note: TMPx signals and WKUPx signals are multiplexed on the same pins (i.e., TMP0/WKUP0, TMP1/WKUP10, etc). This generates a wake-up event only, a tamper event only or a wake-up and a tamper event.

20.4 Supply Controller Functional Description

20.4.1 Supply Controller Overview

The device can be divided into two power supply areas:

- The backup VDDBU_SW power supply that includes the Supply Controller, a part of the Reset Controller, the slow clock switch, the general-purpose backup registers, the supply monitor and the clock which includes the Real-time Timer and the Real-time Clock.
- The core power supply that includes the other part of the Reset Controller, the Brownout Detector, the processor, the SRAM memory, the Flash memory and the peripherals.

The SUPC controls the core power supply and intervenes when the VDDBU_SW power supply rises (when the system is starting) or when the backup Low-power mode is entered.

The SUPC also integrates the slow clock generator which is based on a 32 kHz crystal oscillator and an embedded 32 kHz RC oscillator. The slow clock defaults to the RC oscillator, but the software can enable the crystal oscillator and select it as the slow clock source.

The SUPC and the VDDBU_SW power supply have a reset circuitry based on a zero-power power-on reset cell. The zero-power power-on reset allows the SUPC to start properly as soon as the VDDBU_SW voltage becomes valid.

At start-up of the system, once the backup voltage VDDBU_SW is valid and the embedded 32 kHz RC oscillator is stabilized, the SUPC starts up the core voltage regulator and ties the SHDN pin to VDDBU. Once the VDDCORE voltage is valid, it releases the system reset signal (vddcore_nreset) to the RSTC. The RSTC module then releases the subsystem 0 reset signals (proc_nreset and periph_nreset). Note that the subsystem 1 remains in reset after power-up.

Once the system has started, the user can program a supply monitor and/or a brownout detector. If a powerfail condition occurs on either VDDIO or on VDDCORE power supplies, the SUPC asserts the system reset signal (vddcore_nreset). This signal is released when the powerfail condition is cleared.

When the backup Low-power mode is entered, the SUPC sequentially asserts the system reset signal and disables the voltage regulator, in order to maintain only the VDDBU_SW power supply. Current consumption is reduced to less than one microamp for the backup part retention. Exit from this mode is possible on multiple wake-up sources including an event on the FWUP pin or WKUPx pins, or a clock alarm. To exit this mode, the SUPC operates in the same way as system start-up by enabling the core voltage regulator and the SHDN pin.

20.4.2 Slow Clock Generator

The SUPC embeds a slow clock generator that is supplied with the VDDBU_SW power supply. As soon as VDDBU_SW is supplied, both the crystal oscillator and the embedded RC oscillator are powered up, but only the embedded RC oscillator is enabled. This allows the slow clock to be valid in a short time (about 100 µs).

The user can select the crystal oscillator to be the source of the slow clock, as it provides a more accurate frequency. The command is executed by writing the Supply Controller Control register (SUPC_CR) with the XTALSEL bit at 1, resulting in the following sequence:

1. The crystal oscillator is enabled.
2. A number of slow RC oscillator clock periods is counted to cover the start-up time of the crystal oscillator (refer to the electrical characteristics for information on 32 kHz crystal oscillator start-up time).
3. The slow clock is switched to the output of the crystal oscillator.
4. The RC oscillator is disabled to save power.

The switching time may vary depending on the slow RC oscillator clock frequency range. The switch of the slow clock source is glitch-free. The OSCSEL bit of the Supply Controller Status register (SUPC_SR) indicates that the switchover has completed.

Reverting to the RC oscillator is only possible by shutting down the VDDBU_SW power supply.

If the crystal oscillator is not needed, the XIN32 and XOUT32 pins should be left unconnected.

The user can also put 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. For details of input characteristics of the XIN32 pin, see the section “Electrical Characteristics”. To enter Bypass mode, the OSCBYPASS bit of the Supply Controller Mode register (SUPC_MR) must be set to 1 before writing a 1 to the bit XTALSEL.

20.4.3 Core Voltage Regulator Control/Low-Power Backup 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. For details, see the section “Electrical Characteristics”.

The voltage regulator can be switched off and the device put in Backup mode by setting the bit VROFF in SUPC_CR.

This asserts the system reset signal after the write resynchronization time, which lasts two slow clock cycles (worst case). Once the system reset signal is asserted, the processor and the peripherals are stopped one slow clock cycle before the core voltage regulator shuts off and the SHDN pin is pulled down to ground.

When the embedded voltage regulator is not used and VDDCORE is supplied via an external supply, the voltage regulator can be disabled. This is done by clearing the ONREG bit in SUPC_MR.

20.4.4 Segmented LCD Voltage Regulator Control

The SUPC can be used to select the power supply source of the Segmented LCD (SLCD) voltage regulator.

This selection is done by the LCDMODE field in SUPC_MR. After a backup reset, the LCDMODE field is at 0. No power supply source is selected and the SLCD reset signal is asserted.

The status of the SLCD Controller (SLCDC) reset is given by the LCDS field in SUPC_ SR.

- If LCDMODE is written to 2 while it is at 0, after the write resynchronization time (about 2 slow clock cycles), the external power supply source is selected, then after one slow clock cycle, the SLCDC reset signal is released.
- If LCDMODE is written to 0 while it is at 2, after the write resynchronization time (about 2 slow clock cycles), the SLCDC reset signal is asserted, then after one slow clock cycle, the external power supply source is deselected.
- If LCDMODE is written to 3 while it is at 0, after the write resynchronization time (about 2 slow clock cycles), the internal power supply source is selected and the embedded regulator is turned on, then after 15 slow clock cycles, the SLCDC reset signal is released.
- If LCDMODE is written to 0 while it is at 3, after the write resynchronization time (about 2 slow clock cycles), the SLCDC reset signal, then after one slow clock cycle, the internal power supply source is deselected.

There are several restrictions concerning the write of the LCDMODE field:

- The user must check that the previous power supply selection is done before writing LCDMODE again. To do so, the user must check that the LCDS flag has the correct value. If LCDMODE is cleared, the LCDS flag is cleared. If LCDMODE is set to 2 or 3, the LCDS flag is set.
- Writing LCDMODE to 2 while it is at 3 or writing LCDMODE to 3 while it is at 2 is forbidden and has no effect.
- Before writing LCDMODE to 2, the user must ensure that the external power supply is ready and supplies the VDDLCD pin.
- Before writing LCDMODE to 3, the user must ensure that the external power supply does not supply the VDDLCD pin.

The SLCD can be used in all low-power modes.

20.4.5 Using Backup Battery/Automatic Power Switch

The power switch automatically selects either VDDBU or VDDIO as a power source.

As soon as VDDIO is present (higher than 1.9V), it supplies the backup area of the device (VDDBU_SW = VDDIO) even if the voltage of VDDBU is higher than VDDIO. If not, the backup area is supplied by the VDDBU voltage source (VDDBU_SW = VDDBU). For more information on power supply schematics, refer to the section “Power Supplies”.

20.4.6 Supply Monitor

The SUPC embeds a supply monitor located in the VDDBU_SW power domain and which monitors VDDIO power supply.

The supply monitor can be used to prevent the processor from falling into an unpredictable state if the main power supply drops below a certain level.

The threshold of the supply monitor is programmable. It can be selected from 1.9V to 3.4V by steps of 100 mV. This threshold is configured in the SMTH field of the Supply Controller Supply Monitor Mode register (SUPC_SMMR).

The supply monitor can also be enabled during one slow clock period on every one of either 32, 256 or 2048 slow clock periods, 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 or 128, respectively, if continuous monitoring of the VDDIO power supply is not required.

A supply monitor detection can either generate a system reset (vddcore_nreset signal is asserted) or a system wake-up. Generating a system reset when a supply monitor detection occurs is enabled by setting the SMRSTEN bit in SUPC_SMMR.

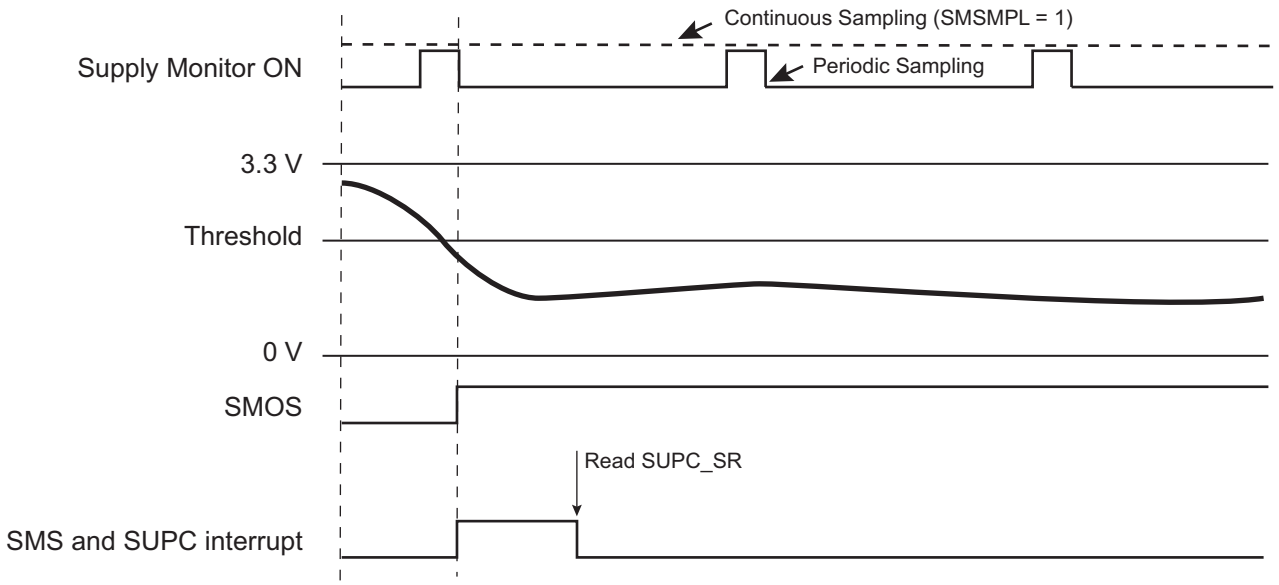
Waking up the system when a supply monitor detection occurs is enabled by setting the SMEN bit in the Supply Controller Wake-up Mode register (SUPC_WUMR).

The SUPC provides two status bits for the supply monitor in the SUPC_SR. These bits 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 bit generates an interrupt if the SMIEN bit is set in SUPC_SMMR.

Figure 20-2. Supply Monitor Status Bit and Associated Interrupt



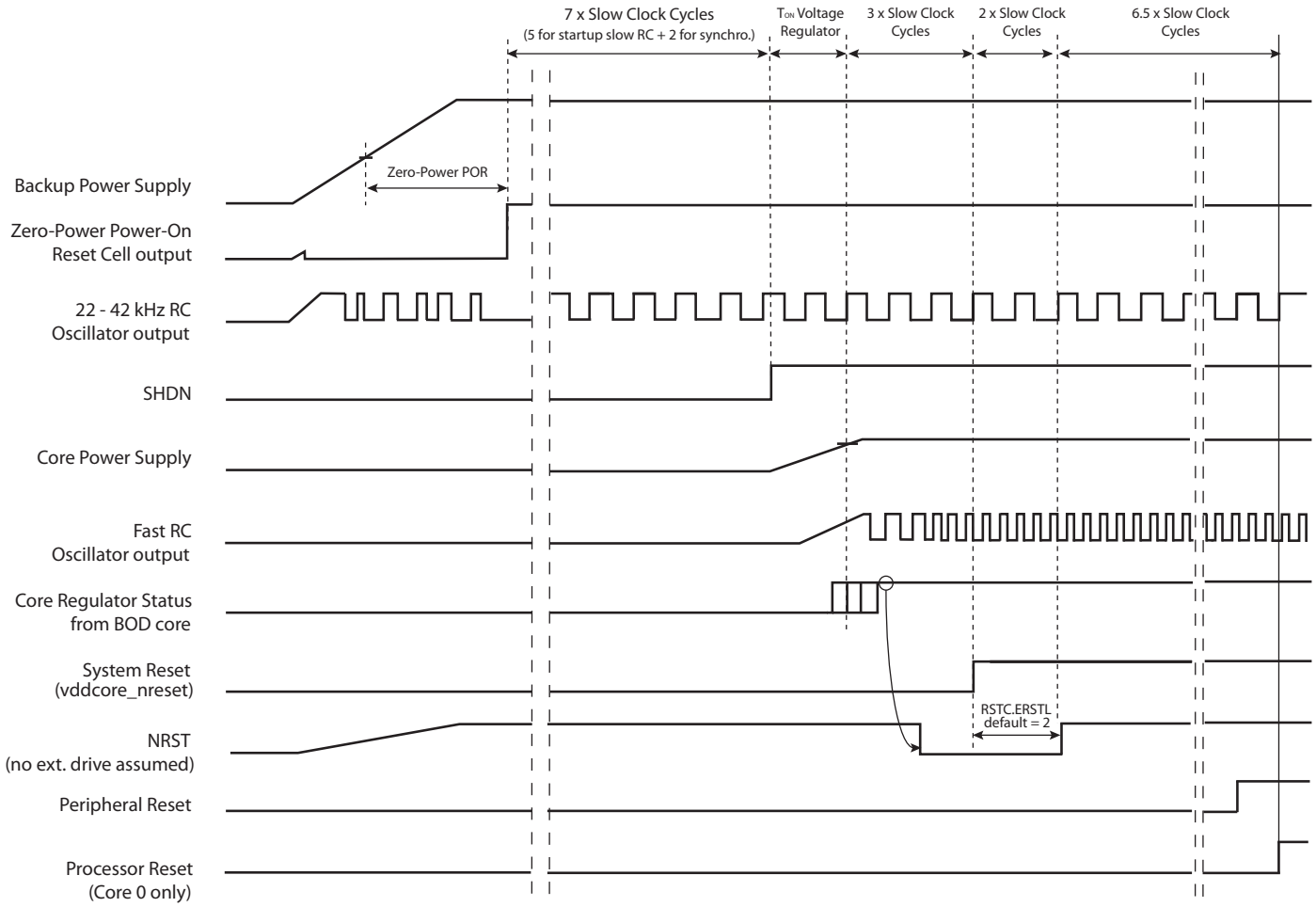
20.4.7 Backup Power Supply Reset

20.4.7.1 Raising the Backup Power Supply

As soon as the backup voltage VDDBU_SW rises, the 32 kHz RC oscillator is powered up and the zero-power power-on reset cell maintains its output low as long as VDDBU_SW has not reached its target voltage. During this time, the SUPC is reset. When the VDDBU_SW voltage becomes valid and zero-power power-on reset signal is released, a counter is started for five slow clock cycles. This is the period required for the 32 kHz RC oscillator to stabilize.

After this time, the SHDN pin is asserted high and the core voltage regulator is enabled. The core power supply rises and the brownout detector provides the core regulator status as soon as the core voltage VDDCORE is valid. The system reset signal is then released to the Reset Controller after the core voltage status has been confirmed as being valid for at least one slow clock cycle.

Figure 20-3. Raising the VDDBU_SW Power Supply



Note: After processor reset rising, the core starts fetching instructions from Flash at 4 MHz.

20.4.7.2 SHDN Output Pin

The SHDN pin is designed to drive the enable pin of an external voltage regulator. This pin is controlled by the VROFF bit in SUPC_CR. When the device goes into Backup mode (bit VROFF set), the SHDN pin is asserted low. Upon a wake-up event, the SHDN pin is released (VDDBU level).

20.4.8 System Reset

The SUPC manages the system reset signal (vddcore_nreset) to the Reset Controller, as described in Section 20.4.7 "Backup Power Supply Reset". The system reset 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 the system reset signal:

- a supply monitor detection
- a brownout detection

20.4.8.1 Supply Monitor Reset

The supply monitor can generate a reset of the system. This can be enabled by setting the SMRSTEN bit in SUPC_SMMR.

The output of the supply monitor is synchronized on SLCK. If SMRSTEN is set and if a supply monitor detection occurs, the system reset is asserted one or two slow clock cycles after the detection.

20.4.8.2 Brownout Detector Reset

The brownout detector provides the core voltage status signal (BODCORE_out) to the SUPC which indicates that the voltage regulation is operating as programmed. If this signal is lost for longer than one slow clock period while the voltage regulator is enabled, the SUPC can assert a system reset signal. This feature is enabled by setting BODRSTEN (Brownout Detector Reset Enable) in SUPC_MR.

If BODRSTEN is set and the voltage regulation is lost (output voltage of the regulator too low), the system reset signal is asserted for a minimum of one slow clock cycle and then released if the core voltage status has been reactivated. The BODRSTS bit is set in SUPC_SR, indicating the source of the last reset.

The system reset signal remains active as long as the core voltage status signal (BODCORE_out) indicates a powerfail condition.

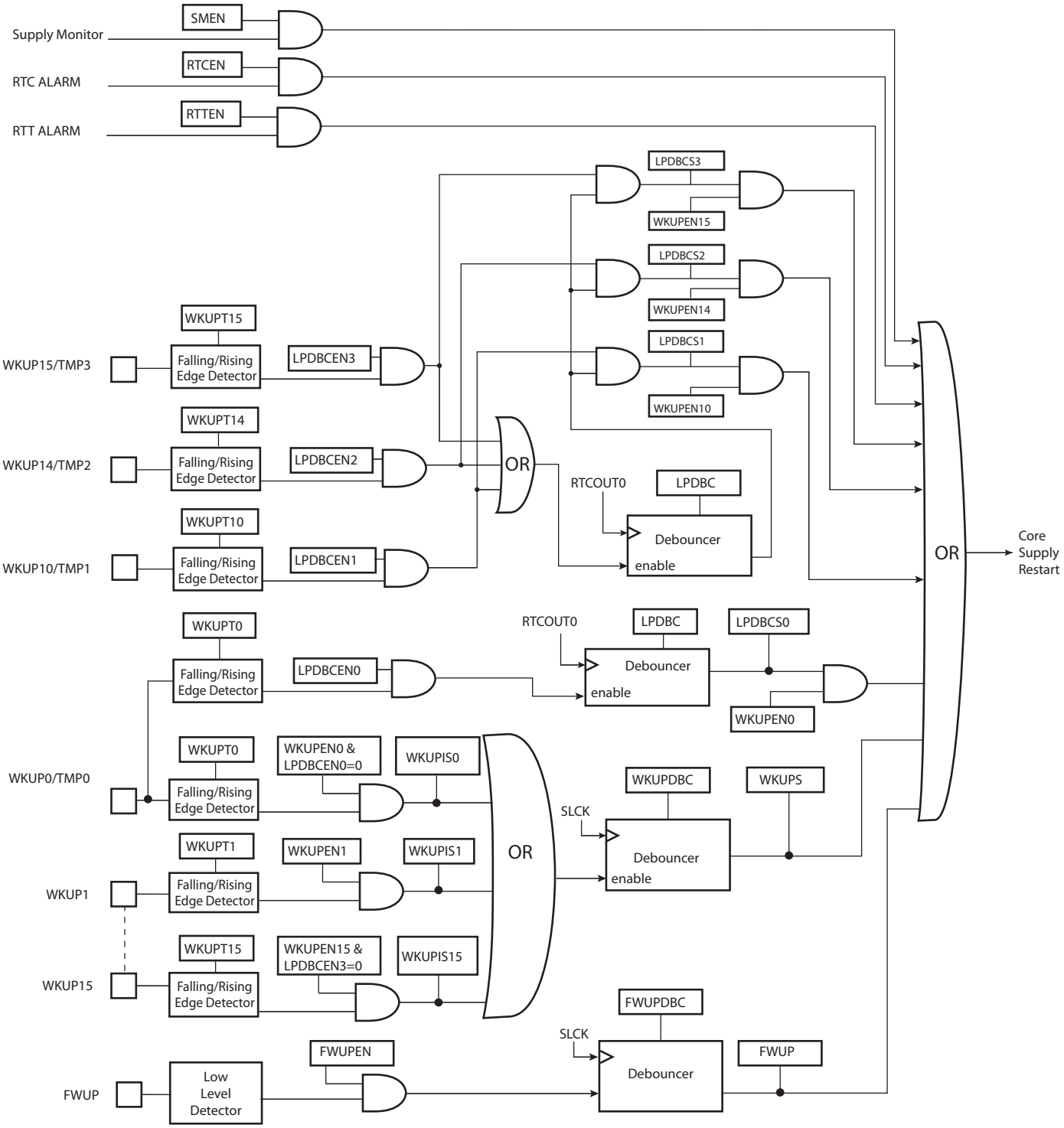
20.4.8.3 Power-on-Reset on VDDBU_SW

The power-on-reset monitors VDDBU_SW. It is active by default and monitors voltage at start-up but also during power-down. It can be deactivated by clearing the BUPPOREN bit in SUPC_MR. If VDDBU_SW goes below the threshold voltage, the chip is reset. Note that due to the automatic power switch, VDDBU_SW can be either VDDIO or VDDBU.

20.4.9 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 which automatically re-enables the core power supply.

Figure 20-4. Wake-up Sources



20.4.9.1 Force Wake-up

The FWUP pin is enabled as a wake-up source by setting the FWUPEN bit in SUPC_WUMR. The FWUPDBC field in the same register then selects the debouncing period, which can be selected between 3, 32, 512, 4,096 or 32,768 slow clock cycles. This corresponds to about 100 µs, about 1 ms, about 16 ms, about 128 ms and about 1 second, respectively (for a typical slow clock frequency of 32 kHz). Configuring FWUPDBC to 0 selects an immediate wake-up, i.e., the FWUP pin must be low during at least one slow clock period to wake up the system.

If the FWUP pin is asserted for a time longer than the debouncing period, a system wake-up is started and the FWUPS bit in SUPC_SR is set and remains high until the register is read.

20.4.9.2 Wake-up Inputs

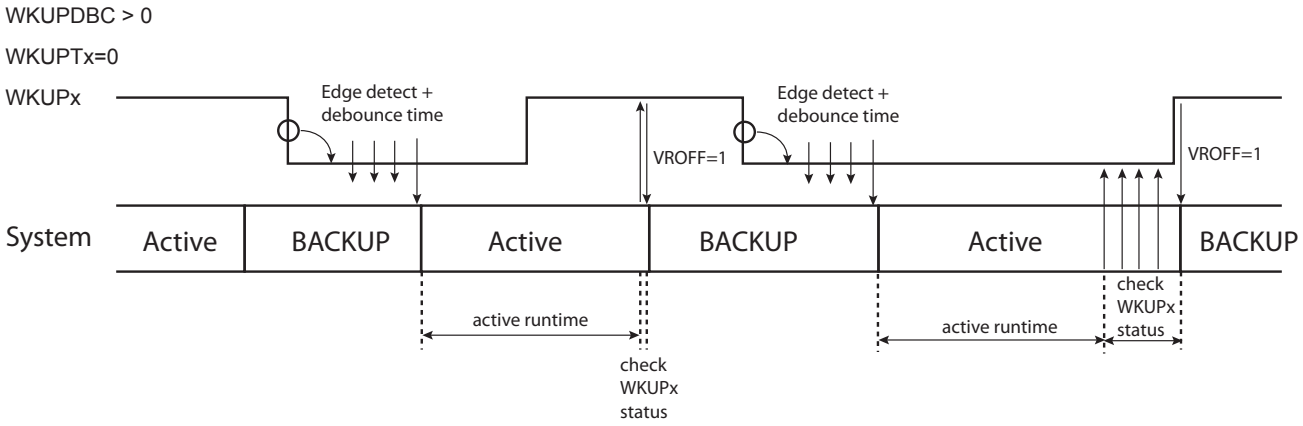
The wake-up inputs WKUPx can be programmed to perform a system wake-up. Each input can be enabled by setting the corresponding bit, WKUPENx, in the Wake-up Inputs register (SUPC_WUIR). The wake-up level can be selected with the corresponding polarity bit, WKUPTx, also located in SUPC_WUIR.

A logical OR combination of all the resulting signals triggers a debouncing counter. The WKUPDBC field can be configured to select a debouncing period of 3, 32, 512, 4,096 or 32,768 slow clock cycles. This corresponds, respectively, to about 100 µs, about 1 ms, about 16 ms, about 128 ms and about 1 second (for a typical slow clock frequency of 32 kHz). Configuring WKUPDBC to 0 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 WKUPx pin holds the active polarity for a time longer than the debouncing period, a system wake-up is started and the flags WKUPISx, as shown in Figure 20-4, are reported 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.

Prior to instructing the system to enter Backup mode, if the field WKUPDBC > 0, it must be verified that none of the WKUPx pins, enabled for a wake-up (exit of Backup mode), holds an active polarity. The verification can be made 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 20-5. Entering and Exiting Backup Mode with a WKUP pin



20.4.9.3 Low-power Debouncer Inputs (Tamper Detection Pins)

Low-power debouncer inputs are dedicated to 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 can be intermittently powered, thus, a specific waveform must be generated.

The waveform can be generated using pin RTCOUT0 in all modes, including Backup mode. Refer to the section “Real-time Clock (RTC)” for waveform generation.

For SAM4CP16C devices, separate debouncers are embedded, one for WKUP0/TMP0 input and a shared one for WKUP10/TMP1, WKUP14/TMP2, WKUP15/TMP3 inputs. See Figure 20-4.

The WKUP0/TMP0 and/or WKUP10/TMP1, WKUP14/TMP2, WKUP15/TMP3 inputs can be programmed to perform a system wake-up with a debouncing done by RTCOUT0. This can be enabled by setting LPDBCEN0/1/2/3 in SUPC_WUMR.

These inputs can be also used when VDDCORE is powered to obtain the tamper detection function with a low-power debounce function and to raise an interrupt.

The low-power debounce mode of operation requires the RTC output (RTCOUT0) to be configured to generate a duty cycle programmable pulse (i.e., OUT0 = 0x7 in RTC_MR) in order to create the sampling points of both debouncers. The sampling point is the falling edge of the RTCOUT0 waveform.

Figure 20-6 shows an example of an application where two tamper switches are used. RTCOUT0 powers the external pull-up used by the tampers.

Figure 20-6. Low-Power Debouncer (Push-to-Make Switch, Pull-up Resistors)

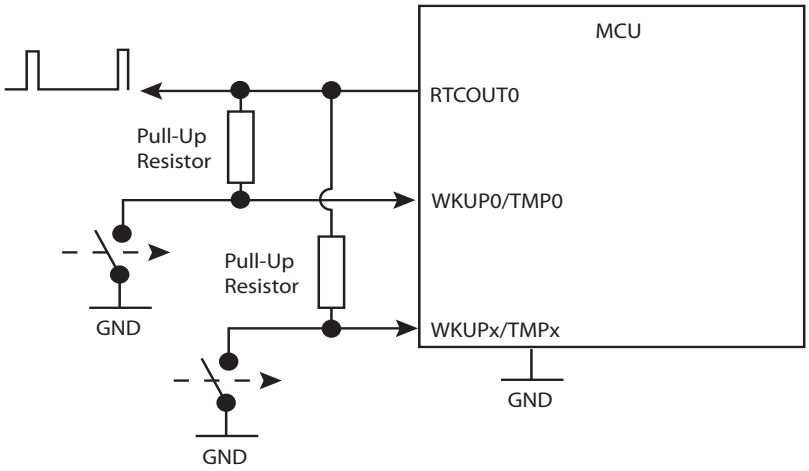
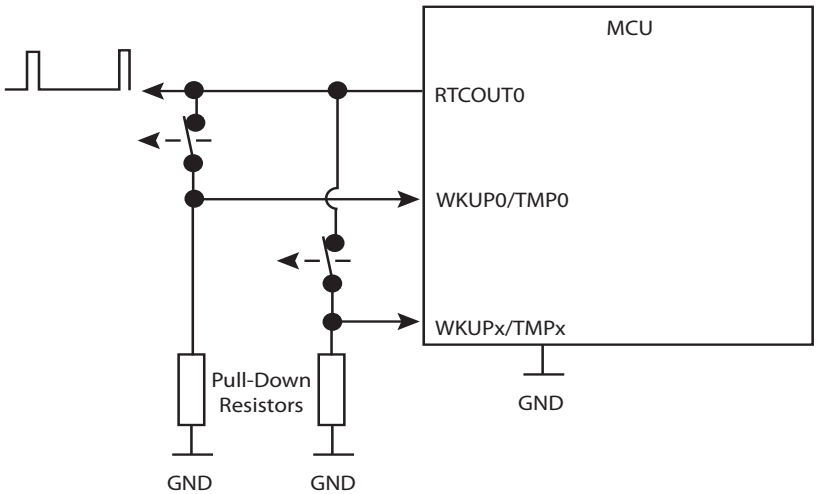


Figure 20-7. Low-Power Debouncer (Push-to-Break Switch, Pull-down Resistors)



The duration of the debouncing period is configurable. The period is identical for all debouncers (i.e., the duration cannot be adjusted separately 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 in 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/WKUPT10/WKUPT14 /WKUPT15 bits in SUPC_WUIR.

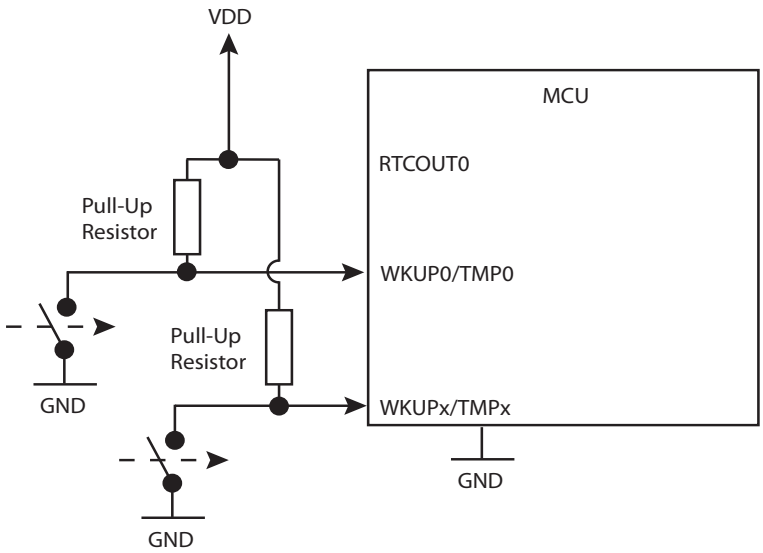
In order to determine which wake-up/tamper pin triggers the system wake-up, a status flag LPDBCSx is associated to each low-power debouncer. These flags can be read in the SUPC_SR.

A debounce event (tamper detection) can perform an immediate clear (0 delay) on the first half of the general-purpose backup registers (GPBR). The LPDBCCLR bit must be set in SUPC_WUMR. The clear capability for TMP1/TMP2/TMP3 can be individually disabled by setting the corresponding bit DISTMPCLR1/2/3.

Note that it is not mandatory to use the RTCOUT0 pin when using the WKUP0/WKUP10/WKUP14/WKUP15 pins as tampering inputs (TMP0/TMP1/TMP2/TMP3) in any mode. Using the RTCOUT0 pin provides a “sampling mode” to further reduce the power consumption of the tamper detection circuitry. If RTCOUT0 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 the RTC_MR.

Figure 20-8 illustrates the use of WKUPx/TMPx without the RTCOUT0 pin.

Figure 20-8. Using WKUP/TMP Pins Without RTCOUT Pins



20.4.9.4 Clock Alarms

The RTC and the RTT alarms generate a system wake-up. This can be enabled by setting bits RTCEN and RTTEN in SUPC_WUMR.

The SUPC does not provide any status, as the information is available in the user interface of either the Real-Time Timer or the Real-Time Clock.

20.4.9.5 Supply Monitor Detection

The supply monitor can generate a system wake-up. See Section 20.4.6 ”Supply Monitor”.

20.5 Register Write Protection

To prevent any single software error from corrupting SUPC 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

20.6 Supply Controller (SUPC) User Interface

The user interface of the SUPC is part of the System Controller user interface.

20.6.1 System Controller (SYSC) User Interface

Table 20-1. System Controller Peripheral Offsets

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	-
0xFC	Reserved	-
0x100 - 0x10C	Reinforced Safety Watchdog Timer	RSWDT
0x110 - 0x124	Time Stamping Registers	RTC

20.6.2 Supply Controller (SUPC) User Interface

Table 20-2. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Supply Controller Control Register	SUPC_CR	Write-only	-
0x04	Supply Controller Supply Monitor Mode Register	SUPC_SMMR	Read/Write	0x0000_0000
0x08	Supply Controller Mode Register	SUPC_MR	Read/Write	0x0000_DA00
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	-	-	-
0xFC	Reserved	-	-	-

20.6.3 Supply Controller Control Register

Name: SUPC_CR

Address: 0x400E1410

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_WPMR).

- **VROFF: Voltage Regulator Off**

0 (NO_EFFECT): No effect.

1 (STOP_VREG): If KEY is correct, asserts the system reset signal and stops the voltage regulator.

- **XTALSEL: Crystal Oscillator Select**

0 (NO_EFFECT): No effect.

1 (CRYSTAL_SEL): If KEY is correct, switches the slow clock on the crystal oscillator output.

- **KEY: Password**

Value	Name	Description
0xA5	PASSWD	Writing any other value in this field aborts the write operation.

20.6.4 Supply Controller Supply Monitor Mode Register

Name: SUPC_SMMR

Address: 0x400E1414

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 can only be written if the WPEN bit is cleared in the System Controller Write Protection Mode Register (SYSC_WPMR).

- **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
0	SMD	Supply Monitor disabled
1	CSM	Continuous Supply Monitor
2	32SLCK	Supply Monitor enabled one SLCK period every 32 SLCK periods
3	256SLCK	Supply Monitor enabled one SLCK period every 256 SLCK periods
4	2048SLCK	Supply Monitor enabled one SLCK period every 2,048 SLCK periods

- **SMRSTEN: Supply Monitor Reset Enable**

0 (NOT_ENABLE): The system reset signal is not affected when a supply monitor detection occurs.

1 (ENABLE): The system reset signal is asserted when a supply monitor detection occurs.

- **SMIEN: Supply Monitor Interrupt Enable**

0 (NOT_ENABLE): The SUPC interrupt signal is not affected when a supply monitor detection occurs.

1 (ENABLE): The SUPC interrupt signal is asserted when a supply monitor detection occurs.

20.6.5 Supply Controller Mode Register

Name: SUPC_MR

Address: 0x400E1418

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
BUPPOREN	ONREG	BODDIS	BODRSTEN	–	–	–	–

7	6	5	4	3	2	1	0
–	–	LCDMODE		LCDVROUT			

This register can only be written if the WPEN bit is cleared in the System Controller Write Protection Mode Register (SYSC_WPMR).

- **LCDVROUT: LCD Voltage Regulator Output**

Adjusts the output voltage of the LCD Voltage Regulator. Refer to the section “Electrical Characteristics” for voltage values.

- **LCDMODE: LCD Controller Mode of Operation**

Value	Name	Description
0	LCDOFF	The internal supply source and the external supply source are both deselected (OFF mode).
1	-	Reserved
2	LCDON_EXTVR	The external supply source for LCD (VDDLCD) is selected (the LCD voltage regulator is in Hi-Z mode).
3	LCDON_INVR	The internal supply source for LCD (the LCD Voltage Regulator) is selected (Active mode).

- **BODRSTEN: Brownout Detector Reset Enable**

0 (NOT_ENABLE): The system reset signal is not affected when a brownout detection occurs.

1 (ENABLE): The system reset signal is asserted when a brownout detection occurs.

- **BODDIS: Brownout Detector Disable**

0 (ENABLE): The core brownout detector is enabled.

1 (DISABLE): The core brownout detector is disabled.

- **ONREG: Voltage Regulator enable**

0 (ONREG_UNUSED): Internal voltage regulator is not used (external power supply is used).

1 (ONREG_USED): Internal voltage regulator is used.

- **BUPPOREN: Backup Area Power-On Reset Enable**

0 (BUPPOR_DISABLE): Disables the backup POR.

1 (BUPPOR_ENABLE): Enables the backup POR.

Note: The value written in BUPPOREN is effective when BUPPORS has the same value in ”Supply Controller Status Register”.

- **OSCBYPASS: Oscillator Bypass**

0 (NO_EFFECT): No effect. Clock selection depends on XTALSEL value.

1 (BYPASS): The 32 kHz crystal oscillator is bypassed if XTALSEL = 1. OSCBYPASS must be set before setting XTALSEL.

- **KEY: Password Key**

Value	Name	Description
0xA5	PASSWD	Writing any other value in this field aborts the write operation.

20.6.6 Supply Controller Wake-up Mode Register

Name: SUPC_WUMR

Address: 0x400E141C

Access: Read/Write

31	30	29	28	27	26	25	24
–	DISTSTMP3	DISTSTMP2	DISTSTMP1	–	DISTMPCLR3	DISTMPCLR2	DISTMPCLR1
23	22	21	20	19	18	17	16
–	–	LPDBCEN3	LPDBCEN2	–	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 can only be written if the WPEN bit is cleared in the System Controller Write Protection Mode Register (SYSC_WPMR).

- **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 a system wake-up.

- **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 a system wake-up.

- **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 a system wake-up.

- **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 a system wake-up.

- **LPDBCEN0: Low-Power Debouncer Enable WKUP0/TMP0**

0 (NOT_ENABLE): The WKUP0/TMP0 input pin is not connected to the low-power debouncer.

1 (ENABLE): The WKUP0/TMP0 input pin is connected to the low-power debouncer and can force a system wake-up.

- **LPDBCEN1: Low-Power Debouncer Enable WKUP10/TMP1**

0 (NOT_ENABLE): The WKUP10/TMP1 input pin is not connected to the low-power debouncer.

1 (ENABLE): The WKUP10/TMP1 input pin is connected to the low-power debouncer and can force 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/TMP0 or WKUP10/14/15/TMP1/2/3 (if DISTMPCLR1/2/3 is cleared) 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_SLCK	FWUP shall be low for at least 3 SLCK periods
2	32_SLCK	FWUP shall be low for at least 32 SLCK periods
3	512_SLCK	FWUP shall be low for at least 512 SLCK periods
4	4096_SLCK	FWUP shall be low for at least 4,096 SLCK periods
5	32768_SLCK	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_SLCK	WKUPx shall be in its active state for at least 3 SLCK periods
2	32_SLCK	WKUPx shall be in its active state for at least 32 SLCK periods
3	512_SLCK	WKUPx shall be in its active state for at least 512 SLCK periods
4	4096_SLCK	WKUPx shall be in its active state for at least 4,096 SLCK periods
5	32768_SLCK	WKUPx shall be in its active state for at least 32,768 SLCK periods

- **LPDBC: Low-Power Debouncer Period**

Value	Name	Description
0	DISABLE	Disable the low-power debouncers
1	2_RTCOUT0	WKUP0/10/14/15/TMP0/1/2/3 in active state for at least 2 RTCOUT0 periods
2	3_RTCOUT0	WKUP0/10/14/15/TMP0/1/2/3 in active state for at least 3 RTCOUT0 periods
3	4_RTCOUT0	WKUP0/10/14/15/TMP0/1/2/3 in active state for at least 4 RTCOUT0 periods
4	5_RTCOUT0	WKUP0/10/14/15/TMP0/1/2/3 in active state for at least 5 RTCOUT0 periods
5	6_RTCOUT0	WKUP0/10/14/15/TMP0/1/2/3 in active state for at least 6 RTCOUT0 periods
6	7_RTCOUT0	WKUP0/10/14/15/TMP0/1/2/3 in active state for at least 7 RTCOUT0 periods
7	8_RTCOUT0	WKUP0/10/14/15/TMP0/1/2/3 in active state for at least 8 RTCOUT0 periods

- **LPDBCEN2: Low-Power Debouncer Enable WKUP14/TMP2**

0 (NOT_ENABLE): The WKUP14/TMP2 input pin is not connected to the low-power debouncer.

1 (ENABLE): The WKUP14/TMP2 input pin is connected to the low-power debouncer and can force a system wake-up.

- **LPDBCEN3: Low-Power Debouncer Enable WKUP15/TMP3**

0 (NOT_ENABLE): The WKUP15/TMP3 input pin is not connected to the low-power debouncer.

1 (ENABLE): The WKUP15/TMP3 input pin is connected to the low-power debouncer and can force a system wake-up.

- **DISTMPCLR1: Disable GPBR Clear Command from WKUP10/TMP1 pin**

0 (ENABLE): The WKUP10/TMP1 input pin can clear the GPBR (if LPDBCCLR is enabled) when tamper is detected.

1 (DISABLE): The WKUP10/TMP1 input pin has no effect on the GPBR value (no clear on tamper detection).

- **DISTMPCLR2: Disable GPBR Clear Command from WKUP14/TMP2 Pin**

0 (ENABLE): The WKUP14/TMP2 input pin can clear the GPBR (if LPDBCCLR is enabled) when tamper is detected.

1 (DISABLE): The WKUP14/TMP2 input pin has no effect on the GPBR value (no clear on tamper detection).

- **DISTMPCLR3: Disable GPBR Clear Command from WKUP15/TMP3 Pin**

0 (ENABLE): The WKUP15/TMP3 input pin can clear the GPBR (if LPDBCCLR is enabled) when tamper is detected.

1 (DISABLE): The WKUP15/TMP3 input pin has no effect on the GPBR value (no clear on tamper detection).

- **DISTSTMP1: Disable Timestamp from WKUP10/TMP1 Pin**

0 (ENABLE): A tamper detection on WKUP10/TMP1 pin generates a timestamp.

1 (DISABLE): A tamper detection on WKUP10/TMP1 does NOT generate a report in timestamp register.

- **DISTSTMP2: Disable Timestamp from WKUP14/TMP2 Pin**

0 (ENABLE): A tamper detection on WKUP14/TMP2 pin generates a timestamp.

1 (DISABLE): A tamper detection on WKUP14/TMP2 does NOT generate a report in timestamp register.

- **DISTSTMP3: Disable Timestamp from WKUP15/TMP3 Pin**

0 (ENABLE): A tamper detection on WKUP15/TMP3 pin generates a timestamp.

1 (DISABLE): A tamper detection on WKUP15/TMP3 does NOT generate a report in timestamp register.

20.6.7 Supply Controller Wake-up Inputs Register

Name: SUPC_WUIR

Address: 0x400E1420

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

- **WKUPENx: WKUPx Input Enable**

0 (DISABLE): The corresponding wake-up input has no wake-up effect.

1 (ENABLE): The corresponding wake-up input is enabled for a system wake-up.

- **WKUPTx: WKUPx Input Type**

0 (LOW): A falling edge followed by a low level for a period defined by WKUPDBC in SUPC_WUMR on the corresponding wake-up input forces a system wake-up.

1 (HIGH): A rising edge followed by a high level for a period defined by WKUPDBC in SUPC_WUMR on the corresponding wake-up input forces a system wake-up.

20.6.8 Supply Controller Status Register

Name: SUPC_SR

Address: 0x400E1424

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
BUPPORS	LPDBCS1	LPDBCS0	FWUPIS	–	LPDBCS3	LPDBCS2	LCDS
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 two slow clock cycles after the read of the SUPC_SR.

- **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 SUPC_SR.

1 (PRESENT): At least one brownout output rising edge event has been detected since the last read of 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 system reset since the last read of SUPC_SR.

1 (PRESENT): At least one supply monitor detection has generated a system reset since the last read of 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.

- **LCDS: LCD Status**

0 (DISABLED): LCD controller is disabled.

1 (ENABLED): LCD controller is enabled.

- **LPDBCS2: Low-Power Debouncer Tamper Status on WKUP14/TMP2 (cleared on read)**

0 (NO): No tamper detection or wake-up due to the assertion of the WKUP14/TMP2 pin has occurred since the last read of SUPC_SR.

1 (PRESENT): At least one tamper detection and wake-up (if enabled by WKUPEN14) due to the assertion of the WKUP14/TMP2 pin has occurred since the last read of SUPC_SR. The SUPC interrupt line is asserted while LPDBCS2 is 1.

- **LPDBCS3: Low-Power Debouncer Tamper Status on WKUP15/TMP3 (cleared on read)**

0 (NO): No tamper detection or wake-up due to the assertion of the WKUP15/TMP3 pin has occurred since the last read of SUPC_SR.

1 (PRESENT): At least one tamper detection and wake-up (if enabled by WKUPEN15) due to the assertion of the WKUP15/TMP3 pin has occurred since the last read of SUPC_SR. The SUPC interrupt line is asserted while LPDBCS2 is 1.

- **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/TMP0 (cleared on read)**

0 (NO): No tamper detection or wake-up due to the assertion of the WKUP0/TMP0 pin has occurred since the last read of SUPC_SR.

1 (PRESENT): At least one tamper detection and wake-up (if enabled by WKUPEN0) due to the assertion of the WKUP0/TMP0 pin has occurred since the last read of SUPC_SR. The SUPC interrupt line is asserted while LPDBCS0 is 1.

- **LPDBCS1: Low-Power Debouncer Wake-up Status on WKUP10/TMP1 (cleared on read)**

0 (NO): No tamper detection or wake-up due to the assertion of the WKUP10 pin has occurred since the last read of SUPC_SR.

1 (PRESENT): At least one tamper detection and wake-up (if enabled by WKUPEN10) due to the assertion of the WKUP10/TMP1 pin has occurred since the last read of SUPC_SR. The SUPC interrupt line is asserted while LPDBCS1 is 1.

- **BUPPORS: Backup Area Power-On Reset Status**

0 (BUPPOR_DISABLED): Backup POR is disabled.

1 (BUPPOR_ENABLED): Backup POR is enabled.

Note: The value written in BUPPOREN is effective when BUPPORENS has the same value in ”Supply Controller Status Register”.

- **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.

20.6.9 System Controller Write Protection Mode Register

Name: SYSC_WPMR

Access: Read/Write

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

For more information on Write Protection registers, refer to Section 20.5 ”Register Write Protection”.

- **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 20.5 ”Register Write Protection” for the list of registers that can be 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.

21. General Purpose Backup Registers (GPBR)

21.1 Description

The System Controller embeds 16 General Purpose Backup registers.

It is possible to generate an immediate clear of the content of General Purpose Backup registers 0 to 7 (first half), if a Tamper event is detected on one of the tamper pins, TMP0 to TMP3. The content of the other General Purpose Backup registers (second half) remains unchanged. A Tamper event on pin TMP0 always performs an immediate clear, while tamper event on other tamper pin can be enabled or disabled in the Supply Controller.

The Supply Controller module must be programmed accordingly. In the register SUPC_WUMR in the Supply Controller module, bits LPDBCCLR and LPDBCEN[0..3] 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 LPDBCSx flags are not cleared in the Supply Controller Status Register (SUPC_SR).

21.2 Embedded Characteristics

- 16 32-bit General Purpose Backup Registers
- Immediate Clear on Tamper Event

21.3 General Purpose Backup Registers (GPBR) User Interface

Table 21-1. Register Mapping

Offset	Register	Name	Access	Reset
0x0	General Purpose Backup Register 0	SYS_GPBR0	Read/Write	0x00000000
...	...	...	...	...
0xCC	General Purpose Backup Register 15	SYS_GPBR15	Read/Write	0x00000000

21.3.1 General Purpose Backup Register x

Name: SYS_GPBRx

Address: 0x400E1490

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 VDDBU_SW.

- **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 LPDBCS3 flag has not been cleared in the Supply Controller Status Register (SUPC_SR).

22. Enhanced Embedded Flash Controller (EEFC)

22.1 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.

22.2 Embedded Characteristics

- Increases Performance in Thumb-2 Mode with 128-bit or 64-bit wide Memory Interface up to 100 MHz
- Code Loop Optimization
- 128 Lock Bits, Each Protecting a Lock Region
- 2 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
- ECC Single and Multiple Error Flags Report
- Supports Read of the Calibration Bits

22.3 Product Dependencies

22.3.1 Power Management

The Enhanced Embedded Flash Controller (EEFC) is continuously clocked. The Power Management Controller has no effect on its behavior.

22.3.2 Interrupt Sources

The 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 22-1. Peripheral IDs

Instance	ID
EFC	6

22.4 Functional Description

22.4.1 Embedded Flash Organization

The embedded Flash interfaces directly with the 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 22.4.3.1 ”Get Flash Descriptor Command”).

Figure 22-1. Flash Memory Areas

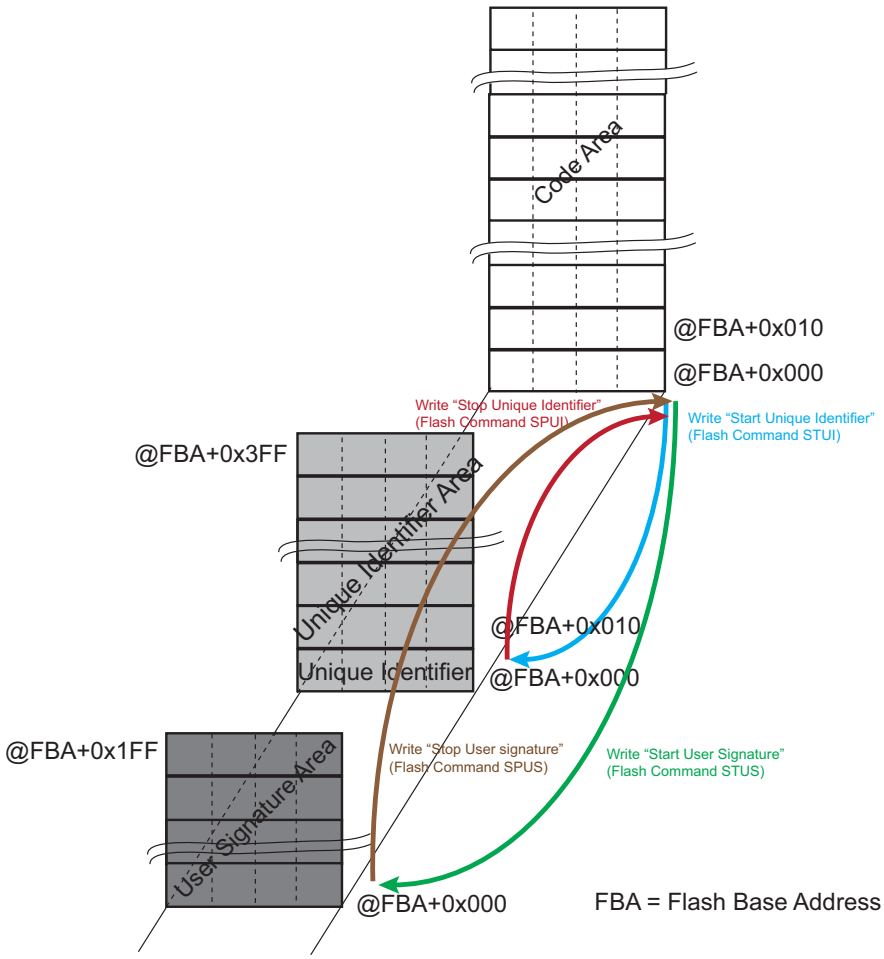
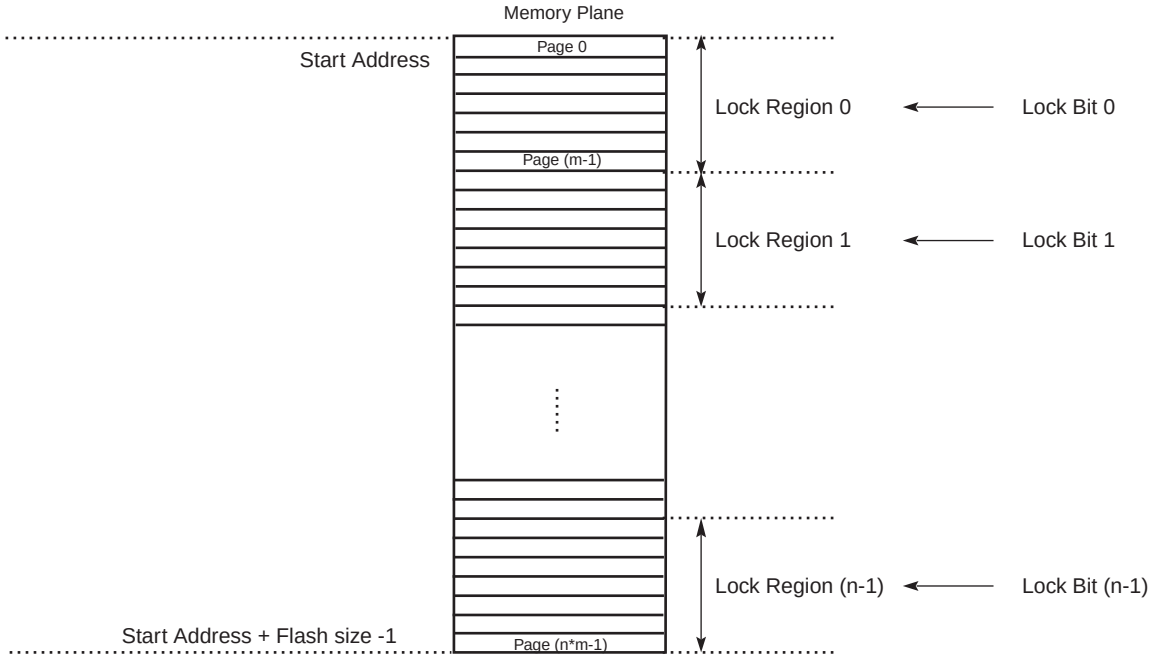


Figure 22-2. Organization of Embedded Flash for Code



22.4.2 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 (Flash Read Wait State) 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.

22.4.2.1 128-bit or 64-bit Access Mode

By default, the read accesses of the Flash are performed through a 128-bit wide memory interface. It 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.

22.4.2.2 Code Read Optimization

Code read optimization is enabled if the bit EEFC_FMR.SCOD is cleared.

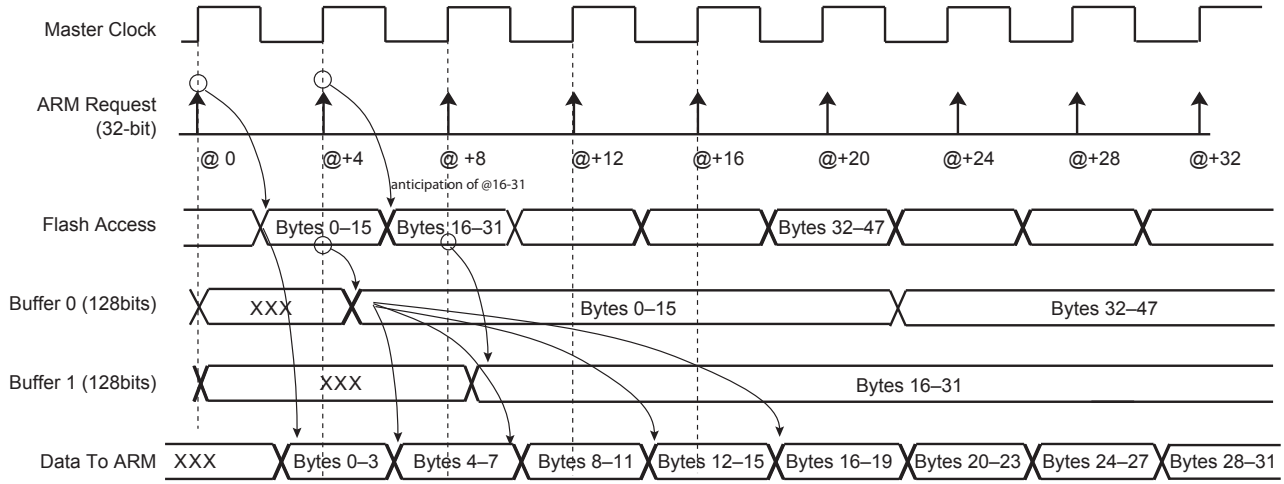
A system of 2 x 128-bit or 2 x 64-bit buffers is added in order to optimize sequential code fetch.

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.

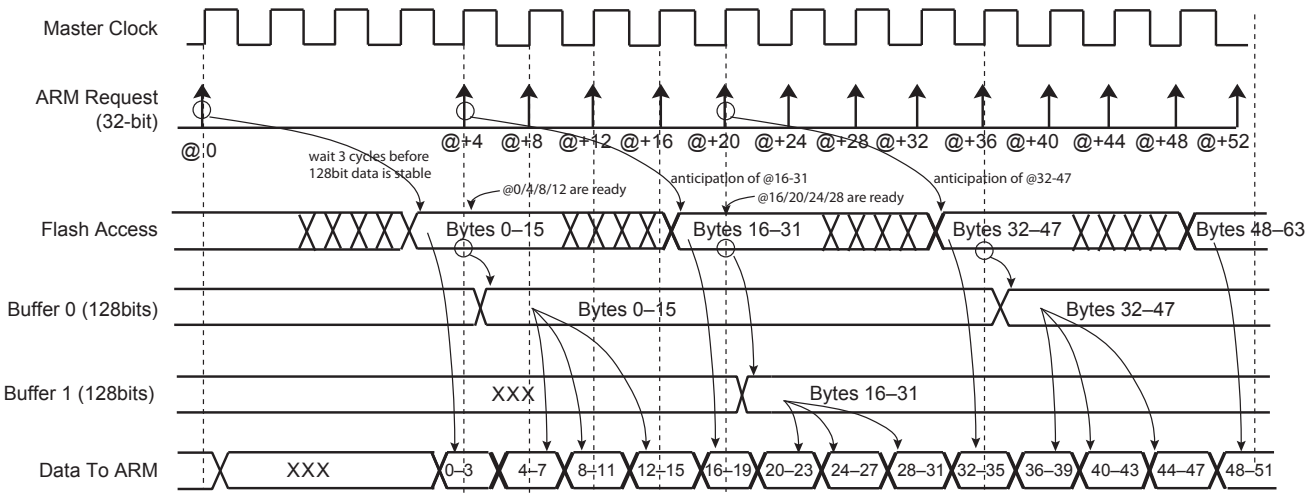
Another system of 2 x 128-bit or 2 x 64-bit buffers is added in order to optimize loop code fetch. Refer to Section 22.4.2.3 ”Code Loop Optimization” for more details.

Figure 22-3. Code Read Optimization for FWS = 0



Note: When FWS is equal to 0, all the accesses are performed in a single-cycle access.

Figure 22-4. 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.

22.4.2.3 Code Loop Optimization

Code loop optimization is enabled when the bit EEFC_FMR.CLOE is set.

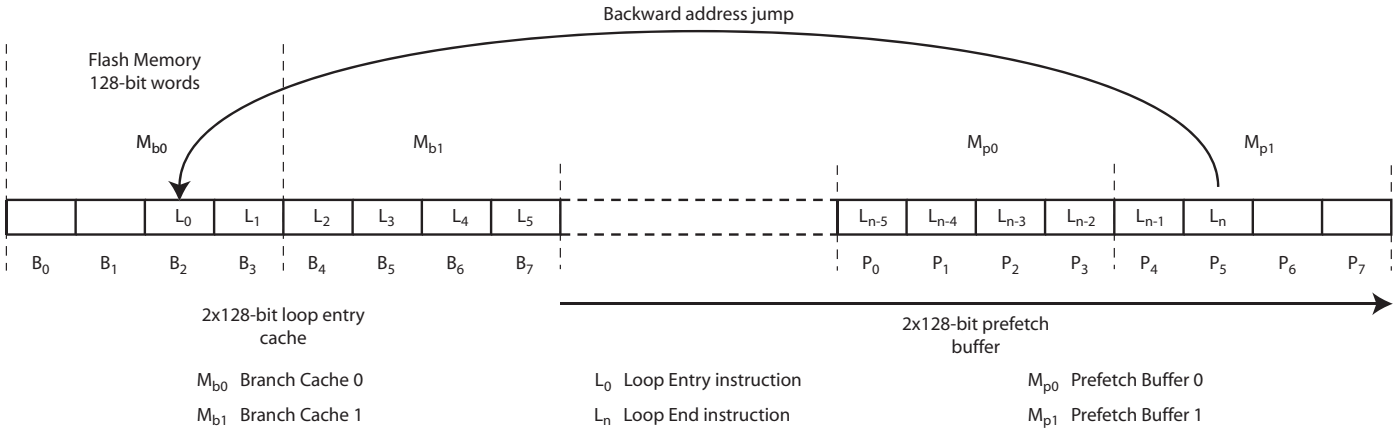
When a backward jump is inserted in the code, the pipeline of the sequential optimization is broken and becomes inefficient. In this case, the loop code read optimization takes over from the sequential code read optimization to prevent the insertion of wait states. The loop code read optimization is 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 22.4.2.2 ”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 22-5 illustrates code loop optimization.

Figure 22-5. Code Loop Optimization

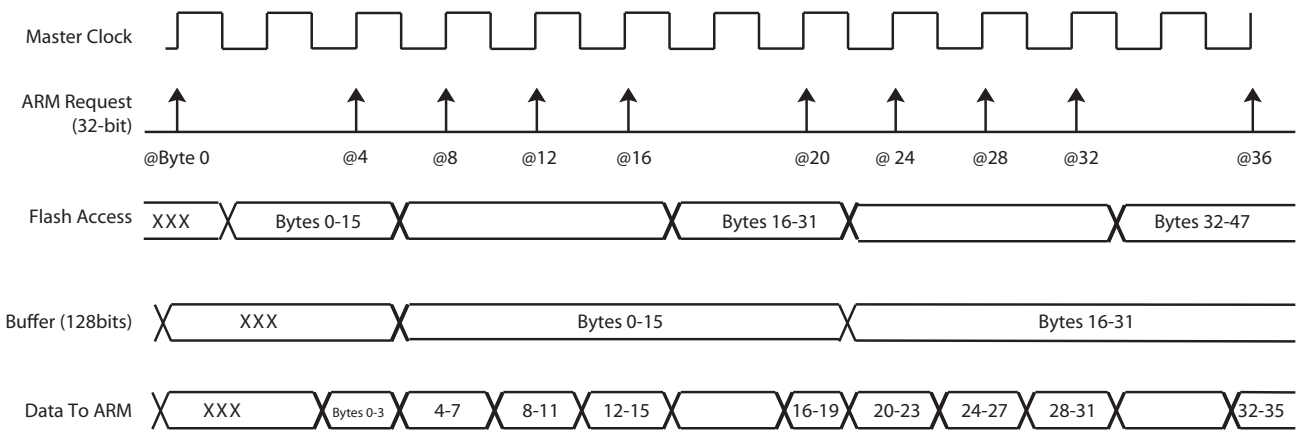


22.4.2.4 Data Read Optimization

The organization of the Flash in 128 bits or 64 bits is associated with two 128-bit or 64-bit prefetch buffers and one 128-bit or 64-bit data read buffer, thus providing maximum system performance. This buffer is added in order to store the requested data plus all the data contained in the 128-bit or 64-bit aligned data. This speeds up sequential data reads if, for example, FWS is equal to 1 (see Figure 22-6). 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 22-6. Data Read Optimization for FWS = 1



22.4.3 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 22-2. Set of Commands

Command	Value	Mnemonic
Get Flash descriptor	0x00	GETD
Write page	0x01	WP
Write page and lock	0x02	WPL
Erase page and write page	0x03	EWP
Erase page and write page then lock	0x04	EWPL
Erase all	0x05	EA
Erase pages	0x07	EPA
Set lock bit	0x08	SLB
Clear lock bit	0x09	CLB
Get lock bit	0x0A	GLB
Set GPNVM bit	0x0B	SGPB
Clear GPNVM bit	0x0C	CGPB
Get GPNVM bit	0x0D	GGPB
Start read unique identifier	0x0E	STUI
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

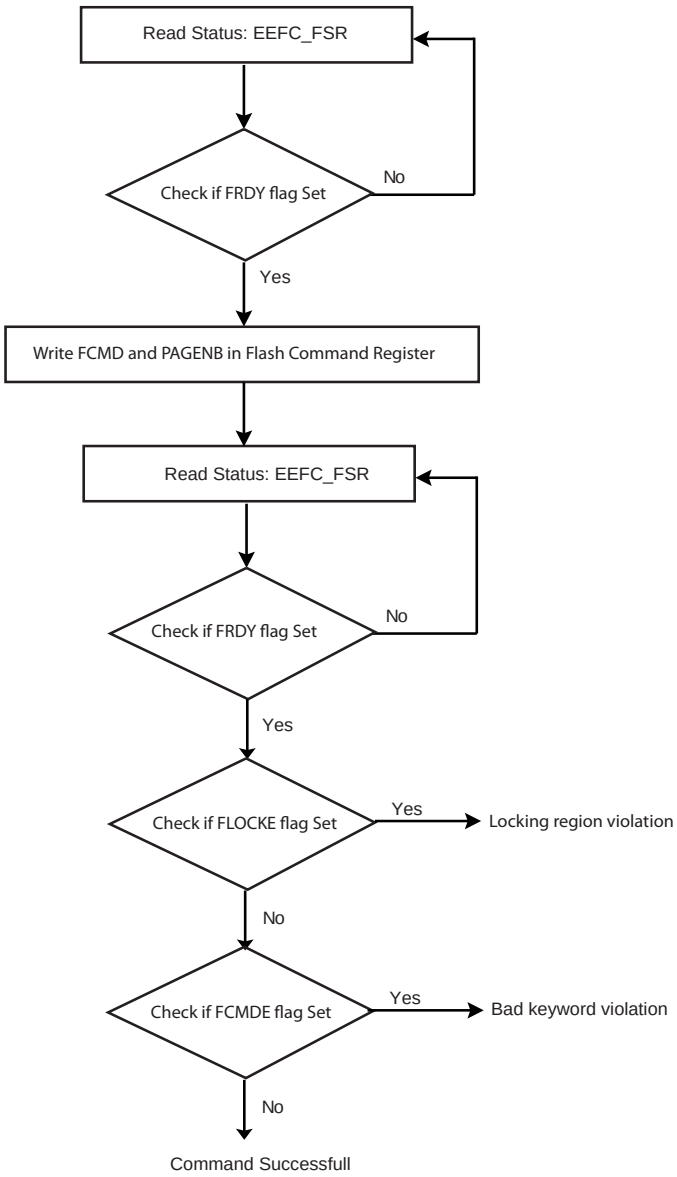
In order to execute one of these commands, select the required command using the FCMD field in the Flash Command register (EEFC_FCR). As soon as EEFC_FCR is written, the FRDY flag and the FVALUE field in the Flash Result register (EEFC_FRR) are automatically cleared. Once the current command has completed, 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 22-7. Command State Chart



22.4.3.1 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 22-3. Flash Descriptor Definition

Symbol	Word Index	Description
FL_ID	0	Flash Interface Description
FL_SIZE	1	Flash size in bytes
FL_PAGE_SIZE	2	Page size in bytes
FL_NB_PLANE	3	Number of planes
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

22.4.3.2 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.

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 22-8, "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 22-9, "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 22-10, "Programming Bytes in the Flash".

Figure 22-8. Full Page Programming

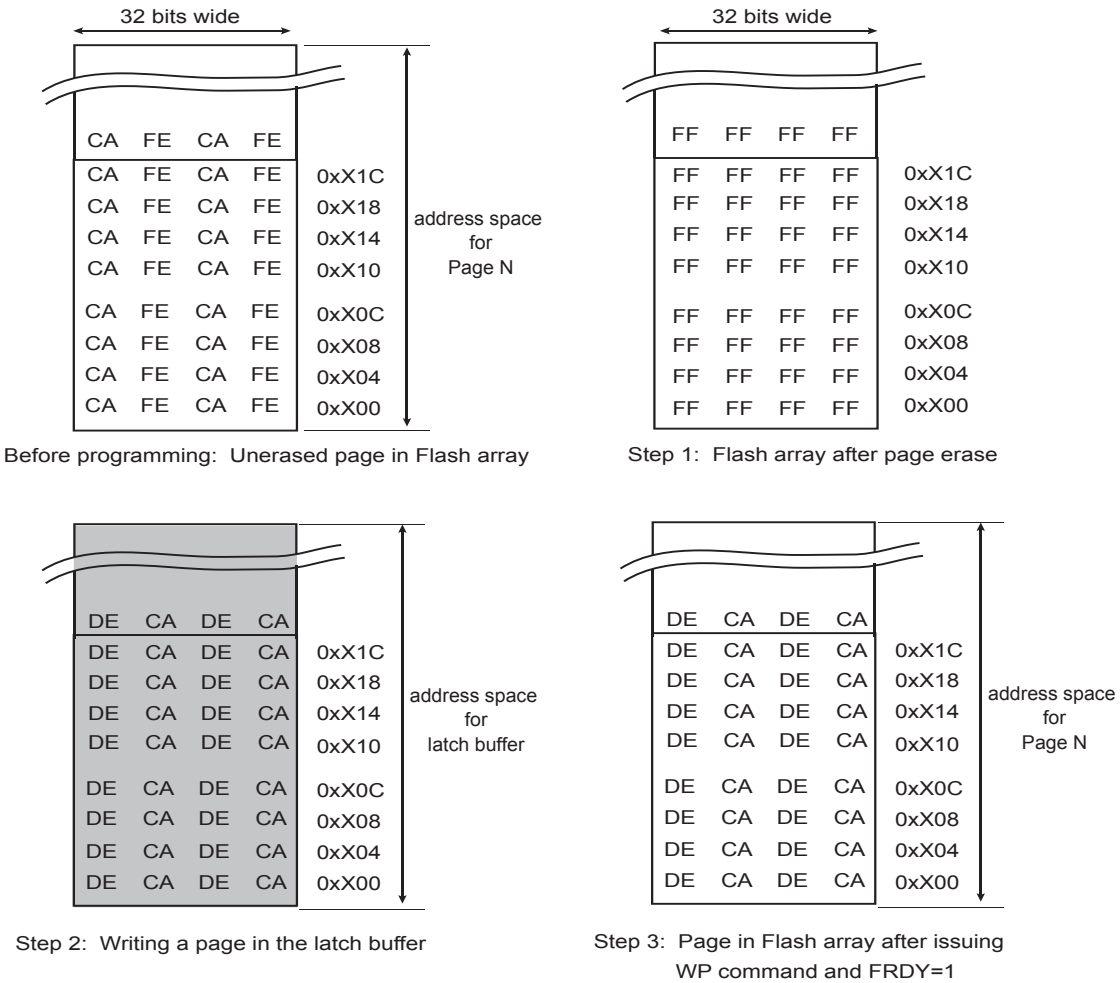
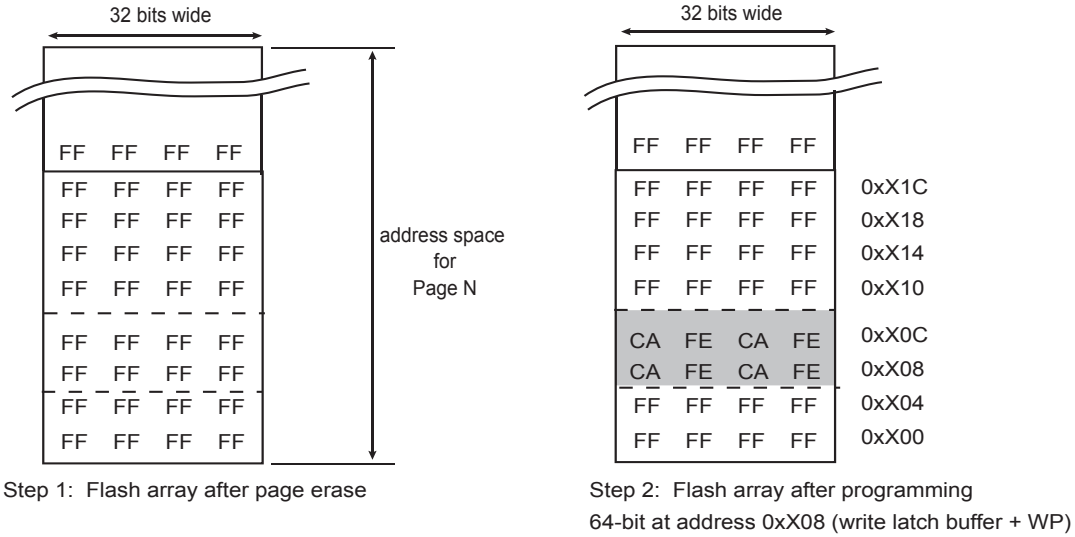


Figure 22-9. Partial Page Programming



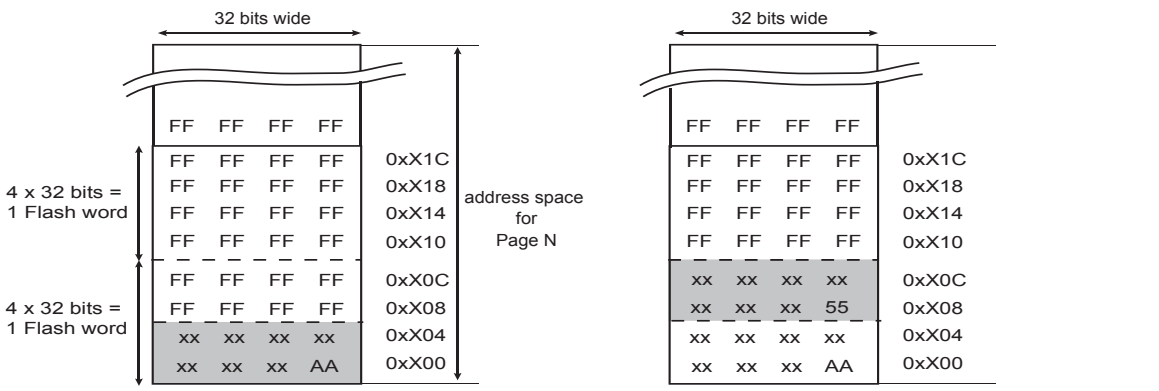
Step 1: Flash array after page erase

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 22-10. 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.

22.4.3.3 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 22-4. 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.

22.4.3.4 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.

22.4.3.5 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.

22.4.3.6 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 22-5 shows the bit implementation.

The RC calibration for the 4 MHz is set to ‘1000000’.

Table 22-5. Calibration Bit Indexes

RC Calibration Frequency	EEFC_FRR Bits
8 MHz RC calibration output	[28 - 22]
12 MHz RC calibration output	[38 - 32]

22.4.3.7 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.

22.4.3.8 Unique Identifier Area

Each device is programmed with a 2x512-bytes unique identifier area. See Figure 22-1, "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.

22.4.3.9 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 22-1, "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 SPUS 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.

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.

22.4.3.10 ECC Errors and Corrections

The Flash embeds an ECC module able to correct one unique error and able to detect two errors. The errors are detected while a read access is performed into memory array and stored in EEFC_FSR (see Section 22.5.3 ”EEFC Flash Status Register”). The error report is kept until EEFC_FSR is read.

There is one flag for a unique error on lower half part of the Flash word (64 LSB) and one flag for the upper half part (MSB). The multiple errors are reported in the same way.

Due to the anticipation technique to improve bandwidth throughput on instruction fetch, a reported error can be located in the next sequential Flash word compared to the location of the instruction being executed, which is located in the previously fetched Flash word.
If a software routine processes the error detection independently from the main software routine, the entire Flash located software must be rewritten because there is no storage of the error location.
If only a software routine is running to program and check pages by reading EEFC_FSR, the situation differs from the previous case. Performing a check for ECC unique errors just after page programming completion involves a read of the newly programmed page. This read sequence is viewed as data accesses and is not optimized by the Flash controller. Thus, in case of unique error, only the current page must be reprogrammed.

22.5 Enhanced Embedded Flash Controller (EEFC) User Interface

The User Interface of the Enhanced Embedded Flash Controller (EEFC) is integrated within the System Controller with base address 0x400E0A00.

Table 22-6. Register Mapping

Offset	Register	Name	Access	Reset State
0x00	EEFC Flash Mode Register	EEFC_FMR	Read/Write	0x0400_0000
0x04	EEFC Flash Command Register	EEFC_FCR	Write-only	–
0x08	EEFC Flash Status Register	EEFC_FSR	Read-only	0x0000_0001
0x0C	EEFC Flash Result Register	EEFC_FRR	Read-only	0x0
0x10 - 0x14	Reserved	–	–	–
0x18 - 0xE4	Reserved	–	–	–

22.5.1 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.

22.5.2 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 22-4, “EEFC_FCR.FARG Field for EPA command,” on page 364
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.

22.5.3 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
–	–	–	–	MECCEMSB	UECCEMSB	MECCELSB	UECCELSB

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).

- **UECCELSB: Unique ECC Error on LSB Part of the Memory Flash Data Bus (cleared on read)**

0: No unique error detected on 64 LSB data bus of the Flash memory since the last read of EEFC_FSR.

1: One unique error detected but corrected on 64 LSB data bus of the Flash memory since the last read of EEFC_FSR.

- **MECCELSB: Multiple ECC Error on LSB Part of the Memory Flash Data Bus (cleared on read)**

0: No multiple error detected on 64 LSB part of the Flash memory data bus since the last read of EEFC_FSR.

1: Multiple errors detected and NOT corrected on 64 LSB part of the Flash memory data bus since the last read of EEFC_FSR.

- **UECCEMSB: Unique ECC Error on MSB Part of the Memory Flash Data Bus (cleared on read)**

0: No unique error detected on 64 MSB data bus of the Flash memory since the last read of EEFC_FSR.

1: One unique error detected but corrected on 64 MSB data bus of the Flash memory since the last read of EEFC_FSR.

- **MECCEMSB: Multiple ECC Error on MSB Part of the Memory Flash Data Bus (cleared on read)**

0: No multiple error detected on 64 MSB part of the Flash memory data bus since the last read of EEFC_FSR.

1: Multiple errors detected and NOT corrected on 64 MSB part of the Flash memory data bus since the last read of EEFC_FSR.

22.5.4 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.

23. Fast Flash Programming Interface (FFPI)

23.1 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.

23.2 Embedded Characteristics

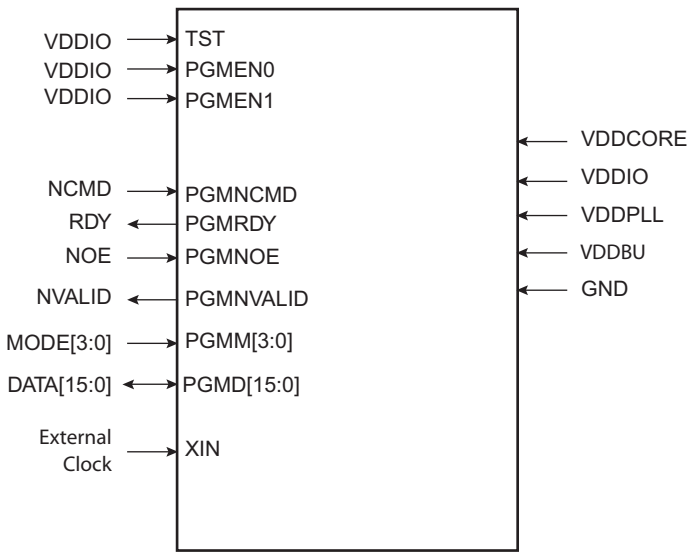
- Programming Mode for High-volume Flash Programming Using Gang Programmer
  - Offers Read and Write Access to the Flash Memory Plane
  - Enables Control of Lock Bits and General-purpose NVM Bits
  - Enables Security Bit Activation
  - Disabled Once Security Bit is Set
- Parallel Fast Flash Programming Interface
  - Provides an 16-bit Parallel Interface to Program the Embedded Flash
  - Full Handshake Protocol

23.3 Parallel Fast Flash Programming

23.3.1 Device Configuration

In Fast Flash Programming mode, the device is in a specific test mode. Only a certain set of pins is significant. The rest of the PIOs are used as inputs with a pull-up. The crystal oscillator is in bypass mode. Other pins must be left unconnected.

Figure 23-1. 16-bit Parallel Programming Interface



23.3.2 Signal Names

Table 23-1. Signal Description List

Signal Name	Function	Type	Active Level	Comments	Pin (LQFP176)
Power					
VDDIO	I/O Lines Power Supply	Power			
VDDCORE	Core Power Supply	Power			
VDDPLL	PLL Power Supply	Power			
GND	Ground	Power			
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	158
PGMEN1	Test Mode Select	Input	High	Must be connected to VDDIO	174
PIO					
PGMNCMD	Valid command available	Input	Low	Pulled-up input at reset	145
PGMRDY	0: Device is busy 1: Device is ready for a new command	Output	High	Pulled-up input at reset	144
PGMNOE	Output Enable (active high)	Input	Low	Pulled-up input at reset	111
PGMNVALID	0: DATA[15:0] is in input mode 1: DATA[15:0] is in output mode	Output	Low	Pulled-up input at reset	176
PGMM0	Specifies DATA type (see Table 23-2)	Input		Pulled-up input at reset	45
PGMM1					126
PGMM2					127
PGMM3					128
PGMD0	Bidirectional data bus	Input/Output		Pulled-up input at reset	147
PGMD1					130
PGMD2					131
PGMD3					133
PGMD4					134
PGMD5					135
PGMD6					43
PGMD7					42
PGMD8					40
PGMD9					115
PGMD10					116
PGMD11					124
PGMD12					38
PGMD13					36
PGMD14					34
PGMD15					110

Depending on the MODE settings, DATA is latched in different internal registers.

Table 23-2. Mode Coding

MODE[3:0]	Symbol	Data
0000	CMDE	Command Register
0001	ADDR0	Address Register LSBs
0010	ADDR1	-
0101	DATA	Data Register
Default	IDLE	No register

When MODE is equal to CMDE, then a new command (strobed on DATA[15:0] signals) is stored in the command register.

Table 23-3. Command Bit Coding

DATA[15:0]	Symbol	Command Executed
0x0011	READ	Read Flash
0x0012	WP	Write Page Flash
0x0022	WPL	Write Page and Lock Flash
0x0032	EWP	Erase Page and Write Page
0x0042	EWPL	Erase Page and Write Page then Lock
0x0013	EA	Erase All
0x0014	SLB	Set Lock Bit
0x0024	CLB	Clear Lock Bit
0x0015	GLB	Get Lock Bit
0x0034	SGPB	Set General Purpose NVM bit
0x0044	CGPB	Clear General Purpose NVM bit
0x0025	GGPB	Get General Purpose NVM bit
0x0054	SSE	Set Security Bit
0x0035	GSE	Get Security Bit
0x001F	WRAM	Write Memory
0x001E	GVE	Get Version

23.3.3 Entering Parallel Programming Mode

The following algorithm puts the device in Parallel Programming mode:

1. Apply the supplies as described in Table 23-1.
2. If an external clock is available, apply it to XIN within the VDDCORE POR reset timeout period, as defined in the section “Electrical Characteristics”.
3. Wait for the end of this reset period.
4. Start a read or write handshaking.

23.3.4 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.

23.3.4.1 Write Handshaking

For details on the write handshaking sequence, refer to Figure 23-2 and Table 23-4.

Figure 23-2. Parallel Programming Timing, Write Sequence

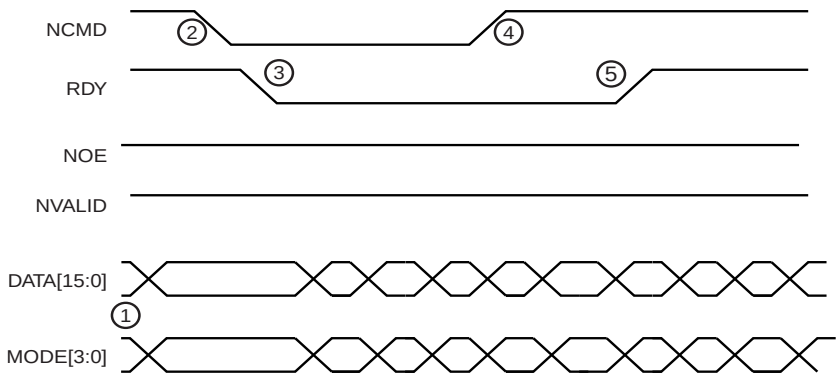


Table 23-4. Write Handshake

Step	Programmer Action	Device Action	Data I/O
1	Sets MODE and DATA signals	Waits for NCMD low	Input
2	Clears NCMD signal	Latches MODE and DATA	Input
3	Waits for RDY low	Clears RDY signal	Input
4	Releases MODE and DATA signals	Executes command and polls NCMD high	Input
5	Sets NCMD signal	Executes command and polls NCMD high	Input
6	Waits for RDY high	Sets RDY	Input

23.3.4.2 Read Handshaking

For details on the read handshaking sequence, refer to Figure 23-3 and Table 23-5.

Figure 23-3. Parallel Programming Timing, Read Sequence

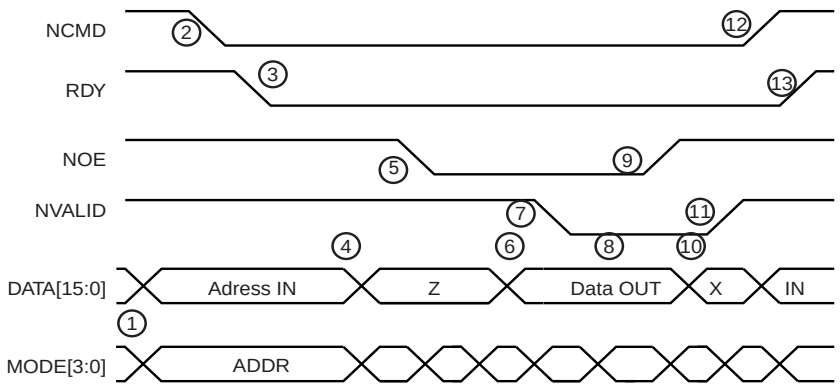


Table 23-5. Read Handshake

Step	Programmer Action	Device Action	DATA I/O
1	Sets MODE and DATA signals	Waits for NCMD low	Input
2	Clears NCMD signal	Latch MODE and DATA	Input
3	Waits for RDY low	Clears RDY signal	Input
4	Sets DATA signal in tristate	Waits for NOE Low	Input
5	Clears NOE signal	-	Tristate
6	Waits for NVALID low	Sets DATA bus in output mode and outputs the flash contents	Output
7	-	Clears NVALID signal	Output
8	Reads value on DATA Bus	Waits for NOE high	Output
9	Sets NOE signal	-	Output
10	Waits for NVALID high	Sets DATA bus in input mode	X
11	Sets DATA in output mode	Sets NVALID signal	Input
12	Sets NCMD signal	Waits for NCMD high	Input
13	Waits for RDY high	Sets RDY signal	Input

23.3.5 Device Operations

Several commands on the Flash memory are available. These commands are summarized in Table 23-3. Each command is driven by the programmer through the parallel interface running several read/write handshaking sequences.

When a new command is executed, the previous one is automatically achieved. Thus, chaining a read command after a write automatically flushes the load buffer in the Flash.

23.3.5.1 Flash Read Command

This command is used to read the contents of the Flash memory. The read command can start at any valid address in the memory plane and is optimized for consecutive reads. Read handshaking can be chained; an internal address buffer is automatically increased.

Table 23-6. Read Command

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	READ
2	Write handshaking	ADDR0	Memory Address LSB
3	Write handshaking	ADDR1	Memory Address
4	Read handshaking	DATA	*Memory Address++
5	Read handshaking	DATA	*Memory Address++
...	...	...	...
n	Write handshaking	ADDR0	Memory Address LSB
n+1	Write handshaking	ADDR1	Memory Address
n+2	Read handshaking	DATA	*Memory Address++
n+3	Read handshaking	DATA	*Memory Address++
...	...	...	...

23.3.5.2 Flash Write Command

This command is used to write the Flash contents.

The Flash memory plane is organized into several pages. Data to be written are stored in a load buffer that corresponds to a Flash memory page. The load buffer is automatically flushed to the Flash:

- before access to any page other than the current one
- when a new command is validated (MODE = CMDE)

The **Write Page** command **(WP)** is optimized for consecutive writes. Write handshaking can be chained; an internal address buffer is automatically increased.

Table 23-7. 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.

23.3.5.3 Flash Full Erase Command

This command is used to erase the Flash memory planes.

All lock regions must be unlocked before the Full Erase command by using the CLB command. Otherwise, the erase command is aborted and no page is erased.

Table 23-8. Full Erase Command

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	EA
2	Write handshaking	DATA	0

23.3.5.4 Flash Lock Commands

Lock bits can be set using WPL or EWPL commands. They can also be set by using the **Set Lock** command **(SLB)**. With this command, several lock bits can be activated. A Bit Mask is provided as argument to the command. When bit 0 of the bit mask is set, then the first lock bit is activated.

In the same way, the **Clear Lock** command **(CLB)** is used to clear lock bits.

Table 23-9. 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 nth lock bit is active when the bit n of the bit mask is set.

Table 23-10. 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

23.3.5.5 Flash General-purpose NVM Commands

General-purpose NVM bits (GP NVM bits) can be set using the **Set GPNVM** command **(SGPB)**. This command also activates GP NVM bits. A bit mask is provided as argument to the command. When bit 0 of the bit mask is set, then the first GP NVM bit is activated.

In the same way, the **Clear GPNVM** command **(CGPB)** is used to clear general-purpose NVM bits. The general-purpose NVM bit is deactivated when the corresponding bit in the pattern value is set to 1.

Table 23-11. 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 nth GP NVM bit is active when bit n of the bit mask is set.

Table 23-12. 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

23.3.5.6 Flash Security Bit Command

A security bit can be set using the **Set Security Bit** command **(SSE)**. Once the security bit is active, the Fast Flash programming is disabled. No other command can be run. An event on the Erase pin can erase the security bit once the contents of the Flash have been erased.

Table 23-13. 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:

- Power-off the chip.
- Power-on the chip with TST = 0.
- Assert the ERASE pin for at least the ERASE pin assertion time as defined in the section “Electrical Characteristics”.
- Power-off the chip.

Return to FFPI mode to check that the Flash is erased.

23.3.5.7 Memory Write Command

This command is used to perform a write access to any memory location.

The **Memory Write** command **(WRAM)** is optimized for consecutive writes. Write handshaking can be chained; an internal address buffer is automatically increased.

Table 23-14. 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++
...	...	...	...

23.3.5.8 Get Version Command

The **Get Version (GVE)** command retrieves the version of the FFPI interface.

Table 23-15. Get Version Command

Step	Handshake Sequence	MODE[3:0]	DATA[15:0]
1	Write handshaking	CMDE	GVE
2	Read handshaking	DATA	Version

24. Cortex-M Cache Controller (CMCC)

24.1 Description

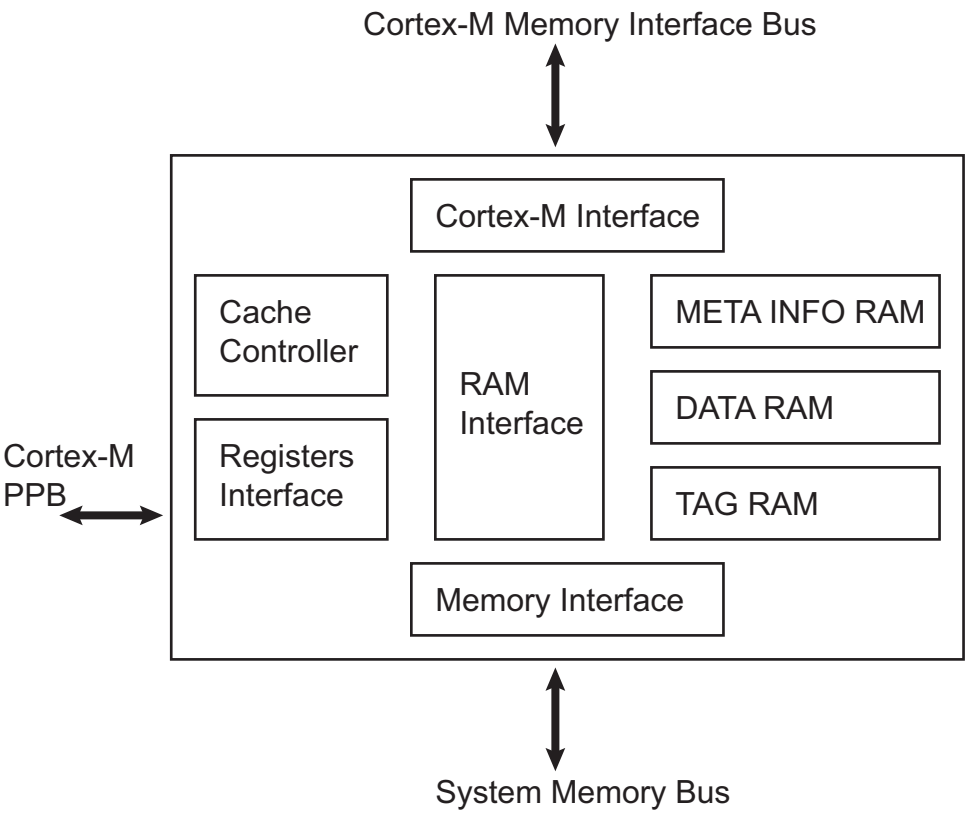
The Cortex-M Cache Controller (CMCC) is a 4-Way set associative unified cache controller. It integrates a controller, a tag directory, data memory, metadata memory and a configuration interface.

24.2 Embedded Characteristics

- Physically addressed and physically tagged.
- L1 data cache set to 2 Kbytes.
- L1 cache line size set to 16 Bytes.
- L1 cache integrates 32-bit 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.

24.3 Block Diagram

Figure 24-1. Block Diagram



24.4 Functional Description

24.4.1 Cache Operation

On reset, the cache controller data entries are all invalidated and the cache is enabled. 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.

The cache must always be enabled, even if the code is running out of a non-cached region.

When the cache is disabled, the accesses to the cache on its slave port are “forwarded” to the master port. In this case, there are two simultaneous accesses on the matrix: one on a non-cached region, and another “dummy” access on the cache master port. These two accesses can slow down the system due to the wait error introduction on the cache master port.

24.4.2 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.

24.4.2.1 Cache Invalidate-by-Line Operation

When an invalidate-by-line command is issued, the cache controller resets the valid bit information of the decoded cache line. As the line is no longer valid, the replacement counter points to that line.

Use the following sequence to invalidate one line of cache:

1. Disable the cache controller by clearing the CEN bit of the Control Register (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 to the CEN bit of the CMCC_CTRL.

24.4.2.2 Cache Invalidate All Operation

To invalidate all cache entries, write a one to the INVALL bit of the Maintenance Register 0 (CMCC_MAINT0).

24.4.3 Cache Performance Monitoring

The Cortex-M cache controller includes a programmable 32-bit monitor counter. The monitor can be configured to count the number of clock cycles, the number of data hits or the number of instruction hits.

Use the following sequence to activate the counter:

1. Configure the monitor counter 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.

24.5 Cortex-M Cache Controller (CMCC) User Interface

Table 24-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Cache Controller Type Register	CMCC_TYPE	Read-only	0x000011D7
0x04	Reserved	–	–	–
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	–	–	–

24.5.1 Cache Controller Type Register

Name: CMCC_TYPE

Address: 0x4007C000 (0), 0x48018000 (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
–	–	CLSIZE			CSIZE		

7	6	5	4	3	2	1	0
LCKDOWN	WAYNUM		RRP	LRUP	RANDP	–	–

- **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

24.5.2 Cache Controller Control Register

Name: CMCC_CTRL

Address: 0x4007C008 (0), 0x48018008 (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
–	–	–	–	–	–	–	CEN

- **CEN: Cache Controller Enable**

0: The cache controller is disabled.

1: The cache controller is enabled.

24.5.3 Cache Controller Status Register

Name: CMCC_SR

Address: 0x4007C00C (0), 0x4801800C (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
–	–	–	–	–	–	–	CSTS

- **CSTS: Cache Controller Status**

0: The cache controller is disabled.

1: The cache controller is enabled.

24.5.4 Cache Controller Maintenance Register 0

Name: CMCC_MAINT0

Address: 0x4007C020 (0), 0x48018020 (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
–	–	–	–	–	–	–	INVALL

- **INVALL: Cache Controller Invalidate All**

0: No effect.

1: All cache entries are invalidated.

24.5.5 Cache Controller Maintenance Register 1

Name: CMCC_MAINT1

Address: 0x4007C024 (0), 0x48018024 (1)

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

24.5.6 Cache Controller Monitor Configuration Register

Name: CMCC_MCFG

Address: 0x4007C028 (0), 0x48018028 (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
–	–	–	–	–	–	MODE	

- **MODE: Cache Controller Monitor Counter Mode**

Value	Name	Description
0	CYCLE_COUNT	Cycle counter
1	IHIT_COUNT	Instruction hit counter
2	DHIT_COUNT	Data hit counter

24.5.7 Cache Controller Monitor Enable Register

Name: CMCC_MEN

Address: 0x4007C02C (0), 0x4801802C (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
–	–	–	–	–	–	–	MENABLE

- **MENABLE: Cache Controller Monitor Enable**

0: The monitor counter is disabled.

1: The monitor counter is enabled.

24.5.8 Cache Controller Monitor Control Register

Name: CMCC_MCTRL

Address: 0x4007C030 (0), 0x48018030 (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

- **SWRST: Monitor**

0: No effect.

1: Resets the event counter register.

24.5.9 Cache Controller Monitor Status Register

Name: CMCC_MSR

Address: 0x4007C034 (0), 0x48018034 (1)

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**

25. Interprocessor Communication (IPC)

25.1 Description

The Interprocessor Communication (IPC) module has 32 interrupt sources. Each source has a set of enable, disable, clear, set, mask and status registers. The interrupt sources are ORed, and the IPC interrupt output line is connected to the Interrupt Controller input.

25.2 Block Diagram

Figure 25-1. IPC Block Diagram

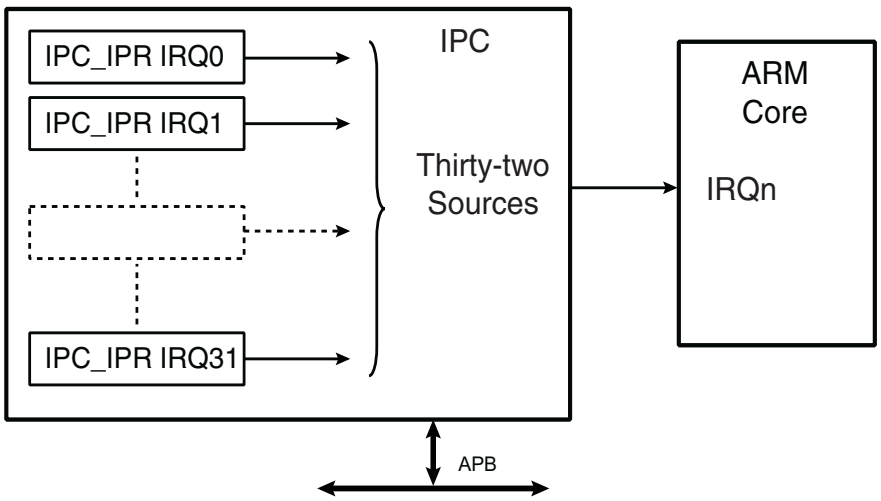
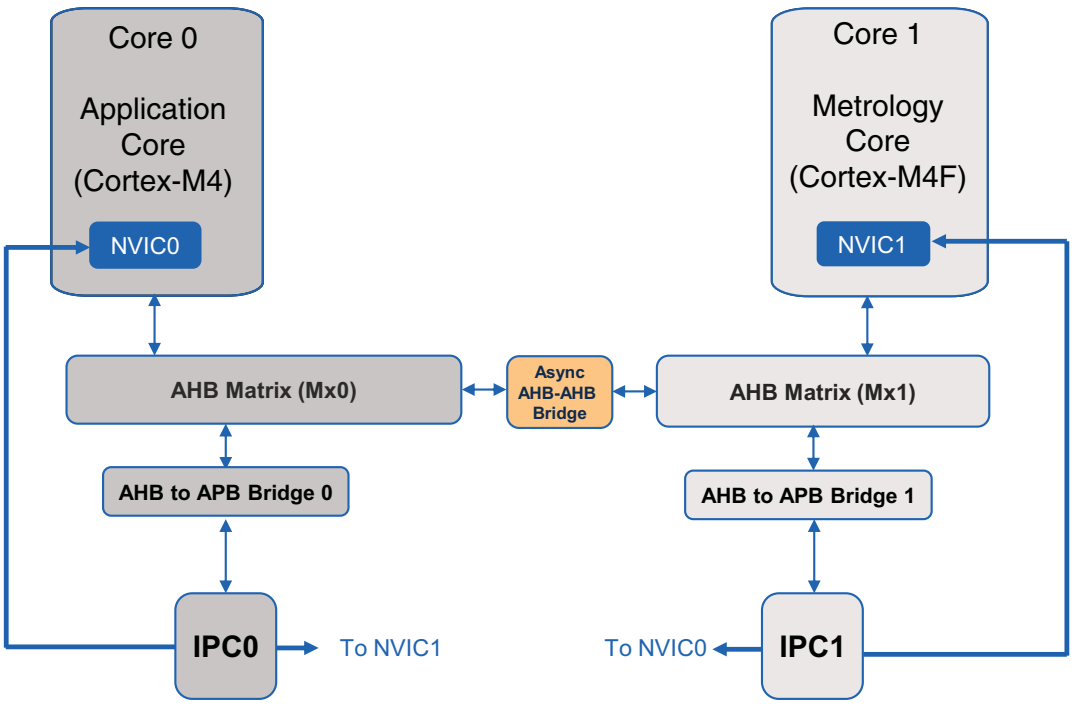


Figure 25-2. Dual Core IPC Implementation



25.3 Product Dependencies

25.3.1 Power Management

The Interprocessor Communication module is not continuously clocked. The IPC interface is clocked through the Power Management Controller (PMC), therefore the programmer must first configure the PMC to enable the IPC clock.

25.3.2 Interrupt Line

The IPC module has an interrupt line connected to the Interrupt Controller. Handling interrupts requires programming the Interrupt Controller before configuring the IPC.

25.4 Functional Description

25.4.1 Interrupt Sources

Table 25-1. Peripheral IDs

Instance	ID
IPC0	31
IPC1	39

25.4.1.1 Interrupt Generation

Interrupt sources can be individually generated by writing respectively the IPC_ISCR and IPC_ICCR registers.

25.4.1.2 Interrupt Source Control

Each interrupt source (IRQ0 to IRQ31) can be enabled or disabled by using the command registers: IPC_IECR (Interrupt Enable Command Register) and IPC_IDCR (Interrupt Disable Command Register). This set of registers conducts enabling or disabling of an instruction. The interrupt mask can be read in the IPC_IMR register. All IPC interrupts can be enabled/disabled, thus configuring the IPC Interrupt mask register. Each pending and unmasked IPC interrupt asserts the IPC output interrupt line.

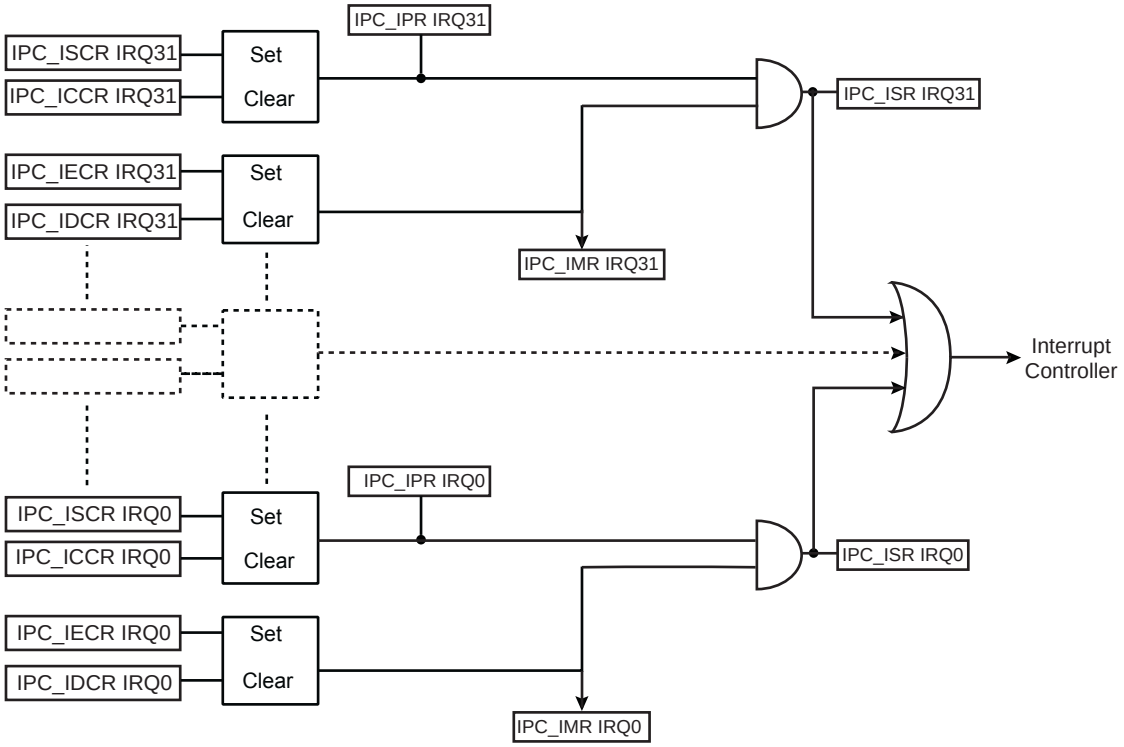
A disabled interrupt does not affect servicing of other interrupts.

25.4.1.3 Interrupt Status

The IPC_IECR and IPC_IDCR registers are used to determine which interrupt sources are active/inhibited to generate an interrupt output. The IPC_IMR register is a status of the interrupt source selection (a result from write into the IPC_IECR and IPC_IDCR registers). The IPC_ISCR and IPC_ICCR registers are used to activate/inhibit interrupt sources. The IPC_IPR register is a status register giving active interrupt sources.

The IPC_ISR register reports which interrupt source(s) is(are) currently asserting an interrupt output. IPC_ISR is basically equivalent to an AND between the IPC_IPR and IPC_IMR registers.

Figure 25-3. Interrupt Input Stage



25.5 Interprocessor Communication (IPC) User Interface

Table 25-2. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	Interrupt Set Command Register	IPC_ISCR	Write-only	–
0x0004	Interrupt Clear Command Register	IPC_ICCR	Write-only	–
0x0008	Interrupt Pending Register	IPC_IPR	Read-only	0x0
0x000C	Interrupt Enable Command Register	IPC_IECR	Write-only	–
0x0010	Interrupt Disable Command Register	IPC_IDCR	Write-only	–
0x0014	Interrupt Mask Register	IPC_IMR	Read-only	0x0
0x0018	Interrupt Status Register	IPC_ISR	Read-only	0x0

25.5.1 IPC Interrupt Set Command Register

Name: IPC_ISCR

Address: 0x4004C000 (0), 0x48014000 (1)

Access: Write-only

31	30	29	28	27	26	25	24
IRQ31	IRQ30	IRQ29	IRQ28	IRQ27	IRQ26	IRQ25	IRQ24
23	22	21	20	19	18	17	16
IRQ23	IRQ22	IRQ21	IRQ20	IRQ19	IRQ18	IRQ17	IRQ16
15	14	13	12	11	10	9	8
IRQ15	IRQ14	IRQ13	IRQ12	IRQ11	IRQ10	IRQ9	IRQ8
7	6	5	4	3	2	1	0
IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	IRQ0

- **IRQ0-IRQ31: Interrupt Set**

0: No effect.

1: Sets the corresponding interrupt.

25.5.2 IPC Interrupt Clear Command Register

Name: IPC_ICCR

Address: 0x4004C004 (0), 0x48014004 (1)

Access: Write-only

31	30	29	28	27	26	25	24
IRQ31	IRQ30	IRQ29	IRQ28	IRQ27	IRQ26	IRQ25	IRQ24
23	22	21	20	19	18	17	16
IRQ23	IRQ22	IRQ21	IRQ20	IRQ19	IRQ18	IRQ17	IRQ16
15	14	13	12	11	10	9	8
IRQ15	IRQ14	IRQ13	IRQ12	IRQ11	IRQ10	IRQ9	IRQ8
7	6	5	4	3	2	1	0
IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	IRQ0

- **IRQ0-IRQ31: Interrupt Clear**

0: No effect.

1: Clears the corresponding interrupt.

25.5.3 IPC Interrupt Pending Register

Name: IPC_IPR

Address: 0x4004C008 (0), 0x48014008 (1)

Access: Read-only

Reset: 0x0

31	30	29	28	27	26	25	24
IRQ31	IRQ30	IRQ29	IRQ28	IRQ27	IRQ26	IRQ25	IRQ24
23	22	21	20	19	18	17	16
IRQ23	IRQ22	IRQ21	IRQ20	IRQ19	IRQ18	IRQ17	IRQ16
15	14	13	12	11	10	9	8
IRQ15	IRQ14	IRQ13	IRQ12	IRQ11	IRQ10	IRQ9	IRQ8
7	6	5	4	3	2	1	0
IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	IRQ0

- **IRQ0-IRQ31: Interrupt Pending**

0: The corresponding interrupt is not pending.

1: The corresponding interrupt is pending.

25.5.4 IPC Interrupt Enable Command Register

Name: IPC_IECR

Address: 0x4004C00C (0), 0x4801400C (1)

Access: Write-only

31	30	29	28	27	26	25	24
IRQ31	IRQ30	IRQ29	IRQ28	IRQ27	IRQ26	IRQ25	IRQ24
23	22	21	20	19	18	17	16
IRQ23	IRQ22	IRQ21	IRQ20	IRQ19	IRQ18	IRQ17	IRQ16
15	14	13	12	11	10	9	8
IRQ15	IRQ14	IRQ13	IRQ12	IRQ11	IRQ10	IRQ9	IRQ8
7	6	5	4	3	2	1	0
IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	IRQ0

- **IRQ0-IRQ31: Interrupt Enable**

0: No effect.

1: Enables the corresponding interrupt.

25.5.5 IPC Interrupt Disable Command Register

Name: IPC_IDCR

Address: 0x4004C010 (0), 0x48014010 (1)

Access: Write-only

31	30	29	28	27	26	25	24
IRQ31	IRQ30	IRQ29	IRQ28	IRQ27	IRQ26	IRQ25	IRQ24
23	22	21	20	19	18	17	16
IRQ23	IRQ22	IRQ21	IRQ20	IRQ19	IRQ18	IRQ17	IRQ16
15	14	13	12	11	10	9	8
IRQ15	IRQ14	IRQ13	IRQ12	IRQ11	IRQ10	IRQ9	IRQ8
7	6	5	4	3	2	1	0
IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	IRQ0

- **IRQ0-IRQ31: Interrupt Disable**

0: No effect.

1: Disables the corresponding interrupt.

25.5.6 IPC Interrupt Mask Register

Name: IPC_IMR

Address: 0x4004C014 (0), 0x48014014 (1)

Access: Read-only

Reset: 0x0

31	30	29	28	27	26	25	24
IRQ31	IRQ30	IRQ29	IRQ28	IRQ27	IRQ26	IRQ25	IRQ24
23	22	21	20	19	18	17	16
IRQ23	IRQ22	IRQ21	IRQ20	IRQ19	IRQ18	IRQ17	IRQ16
15	14	13	12	11	10	9	8
IRQ15	IRQ14	IRQ13	IRQ12	IRQ11	IRQ10	IRQ9	IRQ8
7	6	5	4	3	2	1	0
IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	IRQ0

- **IRQ0-IRQ31: Interrupt Mask**

0: The corresponding interrupt is disabled.

1: The corresponding interrupt is enabled.

25.5.7 IPC Interrupt Status Register

Name: IPC_ISR

Address: 0x4004C018 (0), 0x48014018 (1)

Access: Read-only

Reset: 0x0

31	30	29	28	27	26	25	24
IRQ31	IRQ30	IRQ29	IRQ28	IRQ27	IRQ26	IRQ25	IRQ24
23	22	21	20	19	18	17	16
IRQ23	IRQ22	IRQ21	IRQ20	IRQ19	IRQ18	IRQ17	IRQ16
15	14	13	12	11	10	9	8
IRQ15	IRQ14	IRQ13	IRQ12	IRQ11	IRQ10	IRQ9	IRQ8
7	6	5	4	3	2	1	0
IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	IRQ0

- **IRQ0-IRQ31: Current Interrupt Identifier**

0: The corresponding interrupt source is not currently asserting the interrupt output.

1: The corresponding interrupt source is currently asserting the interrupt output.

26. Bus Matrix (MATRIX)

26.1 Description

The Bus 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 AHB masters to 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).

26.2 Embedded Characteristics

- One Decoder for Each Master.
- Support for Long Bursts of 32, 64 and 128 Beats and Up to the 256-beat Word Burst AHB Limit.
- Enhanced Programmable Mixed Arbitration for Each Slave.
  - Round-robin.
  - Fixed Priority.
  - Latency Quality of Service.
- 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.
- Write Protection of User Interface Registers.

26.2.1 Matrix 0

26.2.1.1 Matrix 0 Masters

The Bus Matrix 0, which corresponds to the sub-system 0 (Core 0 - CM4P0), manages the masters listed in Table 26-1. Each master can perform an access to an available slave concurrently with other masters.

Each master has its own specifically-defined decoder. In order to simplify the addressing, all the masters have the same decodings.

Table 26-1. List of Bus Matrix Masters

Master 0	Cortex-M4 Instruction/Data (CM4P0 I/D Bus)
Master 1	Cortex-M4 System (CM4P0 S Bus)
Master 2	Peripheral DMA Controller 0 (PDC0)
Master 3	Integrity Check Module (ICM)
Master 4	Matrix1
Master 5	Reserved
Master 6	CMCC0

26.2.1.2 Matrix 0 Slaves

The Bus Matrix manages the slaves listed in Table 26-2. Each slave has its own arbiter providing a dedicated arbitration per slave.

Table 26-2. List of Bus Matrix Slaves

Slave 0	Internal SRAM0
Slave 1	Internal ROM
Slave 2	Internal Flash
Slave 3	Reserved
Slave 4	Peripheral Bridge 0
Slave 5	CPKCC RAM and ROM
Slave 6	Matrix1
Slave 7	CMCC0

26.2.1.3 Master to Slave Access (Matrix 0)

Table 26-3 gives valid paths for master to slave access on Matrix 0. The paths shown as "-" are forbidden or not wired, i.e., access from the Cortex-M4 S Bus to the Internal ROM.

Table 26-3. Matrix 0 Master to Slave Access

	Masters	0	1	2	3	4	5	6
Slaves		**Cortex-M4 I/D Bus**	**Cortex-M4 S Bus**	**PDC0**	**ICM**	**Matrix1**	**Reserved**	**CMCC0**
0	Internal SRAM0	-	X	X	X	X	-	-
1	Internal ROM	X	-	X	X	-	-	-
2	Internal Flash	X	-	-	X	X	-	X
3	Reserved	-	-	-	-	-	-	-
4	Peripheral Bridge 0	-	X	X	-	X	-	-
5	CPKCC SRAM, ROM	-	X	-	X	-	-	-
6	Matrix1	-	X	-	X	-	-	-
7	CMCC0	X	-	-	-	-	-	-

26.2.1.4 Accesses through Matrix 0

- CM4P0 I/D Bus access to:
  - Flash, ROM
  - Flash through Cache Controller CMCC0 (respectively through 0x11000000 to 0x11FFFFFF and 0x13000000 to 0x16FFFFFF)
- CMP4P0 S Bus access to:
  - SRAM0, SRAM1 through Matrix1, SRAM2 through Matrix1
  - CPKCC

- PDC0 access to:
  - SRAM0, ROM
  - HBRIDGE0
- ICM access to:
  - Flash, ROM, SRAM0, SRAM1 through Matrix1, SRAM2 through Matrix1
  - CPKCC
  - HBRIDGE1 through Matrix1
- Matrix1 access to
  - Flash (through 0x01000000 to 0x01FFFFFF and 0x11000000 to 0x11FFFFFF)
  - SRAM0
  - HBRIDGE0
- Cache Controller CMCC0 access to:
  - Flash (through 0x11000000 to 0x11FFFFFF)

26.2.2 Matrix 1

26.2.2.1 Matrix 1 Masters

The Bus Matrix 1, which corresponds to the sub-system 1 (Core 1 - CM4P1), manages the masters listed in Table 26-4. Each master can perform an access to an available slave concurrently with other masters.

Each master has its own specifically-defined decoder. In order to simplify the addressing, all the masters have the same decodings.

Table 26-4. List of Bus Matrix Masters

Master 0	Cortex-M4 Instruction/Data (CM4P1 I/D Bus)
Master 1	Cortex-M4 System (CM4P1 S Bus)
Master 2	Peripheral DMA Controller 1 (PDC1)
Master 3	Matrix0
Master 4	Reserved
Master 5	CMCC1

26.2.2.2 Matrix 1 Slaves

The Bus Matrix manages the slaves listed in Table 26-5. Each slave has its own arbiter providing a dedicated arbitration per slave.

Table 26-5. List of Bus Matrix Slaves

Slave 0	Internal SRAM1
Slave 1	Internal SRAM2
Slave 2	Reserved
Slave 3	Peripheral Bridge 1
Slave 4	Matrix0
Slave 5	CMCC1

26.2.2.3 Master to Slave Access (Matrix 1)

Table 26-6 gives valid paths for master to slave access on Matrix 1. The paths shown as "-" are forbidden or not wired, i.e., access from the Cortex-M4 S Bus to the Internal ROM.

Table 26-6. Matrix 1 Master to Slave Access

Slaves	Masters	0	1	2	3	4	5
		Cortex-M4 I/D Bus	Cortex-M4 S Bus	PDC1	Matrix0	Reserved	CMCC1
0	Internal SRAM1	X	X	X	X	-	-
1	Internal SRAM2	-	X	X	X	-	-
2	Reserved	-	-	-	-	-	-
3	Peripheral Bridge 1	-	X	X	X	-	-
4	Matrix0	X	X	-	-	-	X
5	CMCC1	X	-	-	-	-	-

26.2.2.4 Accesses through Matrix 1

- CM4P1 I/D Bus access to:
  - Flash (through 0x01000000 to 0x01FFFFFF)
  - Flash through Cache CMCC1
- CM4P1 S-Bus access to:
  - SRAM1, SRAM2, SRAM0 through Matrix0 (0x20000000)
  - HBRIDGE1, HBRIDGE0 through Matrix0 (0x40000000)
- PDC1 access to:
  - SRAM1, SRAM2
  - HBRIDGE1
- Matrix0 access to:
  - SRAM1, SRAM2
  - HBRIDGE1
- Cache CMCC1 access to:
  - Flash through 0x11000000

26.3 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. However, the technique does not provide any benefits 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 Slave Configuration registers, one for every slave which set a default master for each slave. The Slave Configuration register contains two fields to manage master selection: DEFMSTR_TYPE and FIXED_DEFMSTR. The 2-bit DEFMSTR_TYPE field selects the default master type (no default, last access master, fixed default master), whereas the 4-bit FIXED_DEFMSTR field selects a fixed default master provided that DEFMSTR_TYPE is set to fixed default master. Refer to Section 26.9.2 “Bus Matrix Slave Configuration Registers”.

26.4 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 the 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 regardless of the number of requesting masters.

26.5 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 one latency cycle for the last master that accessed the slave. Other non-privileged masters still get one latency clock cycle if they need to access the same slave. This technique is used for masters that perform single accesses or short bursts with some idle cycles in between.

This configuration provides no benefit on access latency or bandwidth when reaching maximum slave bus throughput whatever is the number of requesting masters.

26.6 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 used for a master that mainly performs single accesses or short bursts with idle cycles in between.

This configuration provides no benefit on access latency or bandwidth when reaching maximum slave bus throughput, regardless of the number of requesting masters.

26.7 Arbitration

The Bus Matrix provides an arbitration mechanism that reduces latency when a conflict occurs, 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 two arbitration types or mixing them for each slave:

1. Round-robin arbitration (default)
2. Fixed priority arbitration

The resulting algorithm may be complemented by selecting a default master configuration for each slave.

When re-arbitration must be done, specific conditions apply. See Section 26.7.1 “Arbitration Scheduling”.

26.7.1 Arbitration Scheduling

Each arbiter has the ability to arbitrate between two or more 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 burst length, predicted end of burst matches the size of the transfer but is managed differently for undefined burst length. See Section 26.7.1.1 “Undefined Length Burst Arbitration”.
4. Slot cycle limit: When the slot cycle counter has reached the limit value, indicating that the current master access is too long and must be broken. See Section 26.7.1.2 “Slot Cycle Limit Arbitration”.

26.7.1.1 Undefined Length Burst Arbitration

In order to prevent long AHB burst lengths that can lock the access to the slave for an excessive period of time, the user can trigger the re-arbitration before the end of the incremental bursts. The re-arbitration period can be selected from the following Undefined Length Burst Type (ULBT) possibilities:

1. Unlimited: no predetermined end of burst is generated. This value enables 1-kbyte burst lengths.
2. 1-beat bursts: predetermined end of burst is generated at each single transfer during the INCR transfer.
3. 4-beat bursts: predetermined end of burst is generated at the end of each 4-beat boundary during INCR transfer.
4. 8-beat bursts: predetermined end of burst is generated at the end of each 8-beat boundary during INCR transfer.
5. 16-beat bursts: predetermined end of burst is generated at the end of each 16-beat boundary during INCR transfer.
6. 32-beat bursts: predetermined end of burst is generated at the end of each 32-beat boundary during INCR transfer.
7. 64-beat bursts: predetermined end of burst is generated at the end of each 64-beat boundary during INCR transfer.
8. 128-beat bursts: predetermined end of burst is generated at the end of each 128-beat boundary during INCR transfer.

The use of undefined length 8-beat bursts or less is discouraged since this may decrease the overall bus bandwidth due to arbitration and slave latencies at each first access of a burst.

However, if the usual length of undefined length bursts is known for a master, it is recommended to configure the ULBT according to this length.

This selection can be done through the ULBT field of the Master Configuration registers (MATRIX_MCFG).

26.7.1.2 Slot Cycle Limit Arbitration

The Bus Matrix contains specific logic to break long accesses, such as very long bursts on a very slow slave (i.e., an external low speed memory). At each arbitration time, a counter is loaded with the value previously written in the SLOT_CYCLE field of the related Slave Configuration register (MATRIX_SCFG) and decreased at each clock cycle. When the counter elapses, the arbiter has the ability to re-arbitrate at the end of the current AHB bus access cycle.

Unless a master has a very tight access latency constraint, which could lead to data overflow or underflow due to a badly undersized internal FIFO with respect to its throughput, the Slot Cycle Limit should be disabled (SLOT_CYCLE = 0) or set to its default maximum value in order not to inefficiently break long bursts performed by some Atmel masters.

In most cases, this feature is not needed and should be disabled for power saving.

Warning: This feature cannot prevent any slave from locking its access indefinitely.

26.7.2 Arbitration Priority Scheme

The Bus Matrix arbitration scheme is organized in priority pools. The corresponding access criticality class is assigned to each priority pool as shown in the "Latency Quality of Service" column in Table 26-7. Latency Quality of Service is determined through the Bus Matrix user interface. See Section 26.9.3 "Bus Matrix Priority Registers A For Slaves" for details.

Table 26-7. Arbitration Priority Pools

Priority Pool	Latency Quality of Service
3	Latency Critical
2	Latency Sensitive
1	Bandwidth Sensitive
0	Background Transfers

Round-robin priority is used in the highest and lowest priority pools 3 and 0, whereas fixed level priority is used between priority pools and in the intermediate priority pools 2 and 1. See Section 26.7.2.2 "Round-robin Arbitration".

For each slave, each master is assigned to one of the slave priority pools through the Latency Quality of Service inputs or through the priority registers for slaves (MxPR fields of MATRIX_PRAS and MATRIX_PRBS). When evaluating master requests, this priority pool level always takes precedence.

After reset, most of the masters belong to the lowest priority pool (MxPR = 0, Background Transfer) and, therefore, are 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 latency-sensitive master or a bandwidth-sensitive master will use such a priority level. The higher the priority level (MxPR value), the higher the master priority.

To ensure a good level of CPU performance, it is recommended to configure the CPU priority with the default reset value 2 (Latency Sensitive).

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.

26.7.2.1 Fixed Priority Arbitration

Fixed priority arbitration algorithm is the first and only arbitration algorithm applied between masters from distinct priority pools. It is also used in priority pools other than the highest and lowest priority pools (intermediate priority pools).

Fixed priority arbitration allows the Bus Matrix arbiters to dispatch the requests from different masters to the same slave by using the fixed priority defined by the user in the MxPR field for each master in the Priority registers, MATRIX_PRAS and MATRIX_PRBS. If two or more master requests are active at the same time, the master with the highest priority MxPR number is serviced first.

In intermediate priority pools, if two or more master requests with the same priority are active at the same time, the master with the highest number is serviced first.

26.7.2.2 Round-robin Arbitration

This algorithm is only used in the highest and lowest priority pools. It allows the Bus Matrix arbiters to properly dispatch requests from different masters to the same slave. If two or more master requests are active at the same time in the priority pool, they are serviced in a round-robin increasing master number order.

26.8 Register Write Protection

To prevent any single software error from corrupting the Bus Matrix behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the "Write Protection Mode Register" (MATRIX_WPMR).

If a write access to a write-protected register is detected, the WPVS flag in the "Write Protection Status Register" (MATRIX_WPSR) is set and the field WPVSRC indicates the register in which 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 following registers can be write-protected:

- "Bus Matrix Master Configuration Registers"
- "Bus Matrix Slave Configuration Registers"
- "Bus Matrix Priority Registers A For Slaves"
- "System I/O Configuration Register"

26.9 AHB Bus Matrix (MATRIX) User Interface

Table 26-8. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	Master Configuration Register 0	MATRIX_MCFG0	Read/Write	0x00000004
0x0004	Master Configuration Register 1	MATRIX_MCFG1	Read/Write	0x00000004
0x0008	Master Configuration Register 2	MATRIX_MCFG2	Read/Write	0x00000004
0x000C	Master Configuration Register 3	MATRIX_MCFG3	Read/Write	0x00000004
0x0010	Master Configuration Register 4	MATRIX_MCFG4	Read/Write	0x00000004
0x0014	Master Configuration Register 5	MATRIX_MCFG5	Read/Write	0x00000004
0x0018	Master Configuration Register 6	MATRIX_MCFG6	Read/Write	0x00000004
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	Slave Configuration Register 6	MATRIX_SCFG6	Read/Write	0x000001FF
0x005C	Slave Configuration Register 7	MATRIX_SCFG7	Read/Write	0x000001FF
0x0060 - 0x007C	Reserved	–	–	–
0x0080	Priority Register A for Slave 0	MATRIX_PRAS0	Read/Write	0x00000000(1)
0x0084	Reserved	–	–	–
0x0088	Priority Register A for Slave 1	MATRIX_PRAS1	Read/Write	0x00000000(1)
0x008C	Reserved	–	–	–
0x0090	Priority Register A for Slave 2	MATRIX_PRAS2	Read/Write	0x00000000(1)
0x0094	Reserved	–	–	–
0x0098	Priority Register A for Slave 3	MATRIX_PRAS3	Read/Write	0x00000000(1)
0x009C	Reserved	–	–	–
0x00A0	Priority Register A for Slave 4	MATRIX_PRAS4	Read/Write	0x00000000(1)
0x00A4	Reserved	–	–	–
0x00A8	Priority Register A for Slave 5	MATRIX_PRAS5	Read/Write	0x00000000(1)
0x00AC	Reserved	–	–	–
0x00B0	Priority Register A for Slave 6	MATRIX_PRAS6	Read/Write	0x00000000(1)
0x00B4	Reserved	–	–	–
0x00B8	Priority Register A for Slave 7	MATRIX_PRAS7	Read/Write	0x00000000(1)
0x00BC - 0x0110	Reserved	–	–	–
0x0114	System I/O Configuration Register	MATRIX_SYSIO	Read/Write	0x00000000
0x0118	Reserved	–	–	–

Table 26-8. Register Mapping (Continued)

Offset	Register	Name	Access	Reset
0x0120	Reserved	–	–	–
0x0124	Reserved	–	–	–
0x0128	Core Debug Configuration Register	MATRIX_CORE_DEBUG	Read/Write	0x00000000
0x012C - 0x01E0	Reserved	–	–	–
0x01E4	Write Protection Mode Register	MATRIX_WPMR	Read/Write	0x00000000
0x01E8	Write Protection Status Register	MATRIX_WPSR	Read-only	0x00000000

Note: 1. Values in the Bus Matrix Priority Registers are product dependent.

26.9.1 Bus Matrix Master Configuration Registers

Name: MATRIX_MCFGx [x=0..6]

Address: 0x400E0200 (0), 0x48010000 (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
–	–	–	–	–	ULBT		

This register can only be written if the WPEN bit is cleared in the "Write Protection Mode Register" .

- **ULBT: Undefined Length Burst Type**

0: Unlimited Length Burst

No predicted end of burst is generated, therefore INCR bursts coming from this master can only be broken if the Slave Slot Cycle Limit is reached. If the Slot Cycle Limit is not reached, the burst is normally completed by the master, at the latest, on the next AHB 1 KByte address boundary, allowing up to 256-beat word bursts or 128-beat double-word bursts.

This value should not be used in the very particular case of a master capable of performing back-to-back undefined length bursts on a single slave, since this could indefinitely freeze the slave arbitration and thus prevent another master from accessing this slave.

1: Single Access

The undefined length burst is treated as a succession of single accesses, allowing re-arbitration at each beat of the INCR burst or bursts sequence.

2: 4-beat Burst

The undefined length burst or bursts sequence is split into 4-beat bursts or less, allowing re-arbitration every 4 beats.

3: 8-beat Burst

The undefined length burst or bursts sequence is split into 8-beat bursts or less, allowing re-arbitration every 8 beats.

4: 16-beat Burst

The undefined length burst or bursts sequence is split into 16-beat bursts or less, allowing re-arbitration every 16 beats.

5: 32-beat Burst

The undefined length burst or bursts sequence is split into 32-beat bursts or less, allowing re-arbitration every 32 beats.

6: 64-beat Burst

The undefined length burst or bursts sequence is split into 64-beat bursts or less, allowing re-arbitration every 64 beats.

7: 128-beat Burst

The undefined length burst or bursts sequence is split into 128-beat bursts or less, allowing re-arbitration every 128 beats.

Unless duly needed, the ULBT should be left at its default 0 value for power saving.

26.9.2 Bus Matrix Slave Configuration Registers

Name: MATRIX_SCFGx [x=0..7]

Address: 0x400E0240 (0), 0x48010040 (1)

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 Protection 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 Section 26.7.1.2 "Slot Cycle Limit Arbitration" for details.

- **DEFMSTR_TYPE: Default Master Type**

0: No Default Master

At the end of the current slave access, if no other master request is pending, the slave is disconnected from all masters.

This results in a one clock cycle latency for the first access of a burst transfer or for a single access.

1: Last Default Master

At the end of the current slave access, if no other master request is pending, the slave stays connected to the last master having accessed it.

This results in not having one clock cycle latency when the last master tries to access the slave again.

2: Fixed Default Master

At the end of the current slave access, if no other master request is pending, the slave connects to the fixed master the number that has been written in the FIXED_DEFMSTR field.

This results in not having one clock cycle latency when the fixed master tries to access the slave again.

- **FIXED_DEFMSTR: Fixed Default Master**

This is the number of the Default Master for this slave. Only used if DEFMSTR_TYPE is 2. Specifying the number of a master which is not connected to the selected slave is equivalent to setting DEFMSTR_TYPE to 0.

26.9.3 Bus Matrix Priority Registers A For Slaves

Name: MATRIX_PRASx [x=0..7]

Address: 0x400E0280 (0)[0], 0x400E0288 (0)[1], 0x400E0290 (0)[2], 0x400E0298 (0)[3], 0x400E02A0 (0)[4], 0x400E02A8 (0)[5], 0x400E02B0 (0)[6], 0x400E02B8 (0)[7], 0x48010080 (1)[0], 0x48010088 (1)[1], 0x48010090 (1)[2], 0x48010098 (1)[3], 0x480100A0 (1)[4], 0x480100A8 (1)[5], 0x480100B0 (1)[6], 0x480100B8 (1)[7]

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	M7PR		–	–	M6PR	

23	22	21	20	19	18	17	16
–	–	M5PR		–	–	M4PR	

15	14	13	12	11	10	9	8
–	–	M3PR		–	–	M2PR	

7	6	5	4	3	2	1	0
–	–	M1PR		–	–	M0PR	

This register can only be written if the WPE bit is cleared in the “Write Protection 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 Section 26.7.2 “Arbitration Priority Scheme” for details.

26.9.4 System I/O Configuration Register

Name: MATRIX_SYSIO

Address: 0x400E0314 (0), 0x48010114 (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
–	–	–	–	–	–	SYSIO9	–
7	6	5	4	3	2	1	0
–	–	–	–	SYSIO3	SYSIO2	SYSIO1	SYSIO0

This register can only be written if the WPEN bit is cleared in the "Write Protection Mode Register" .

- **SYSIO0: PB0 or TDI Assignment**

0 = TDI function selected.
1 = PB0 function selected.

- **SYSIO1: PB1 or TDO/TRACESWO Assignment**

0 = TDO/TRACESWO function selected.
1 = PB1 function selected.

- **SYSIO2: PB2 or TMS/SWDIO Assignment**

0 = TMS/SWDIO function selected.
1 = PB2 function selected.

- **SYSIO3: PB3 or TCK/SWCLK Assignment**

0 = TCK/SWCLK function selected.
1 = PB3 function selected.

- **SYSIO9: PC9 or ERASE Assignment**

0 = ERASE function selected.
1 = PC9 function selected.

26.9.5 Core Debug Configuration Register

Name: MATRIX_CORE_DEBUG

Address: 0x400E0328 (0), 0x48010128 (1)

Access: Read/Write

Reset: See Table 26-8

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	–	CROSS_TRG0	CROSS_TRG1	–

- **CROSS_TRG1: Core 1 --> Core 0 Cross Triggering**

0 = Core 1 is not able to trigger an event on core 0.
1 = Core 1 is able to trigger an event on core 0.

- **CROSS_TRG0: Core 0 --> Core 1 Cross Triggering**

0 = Core 0 is not able to trigger an event on core 1.
1 = Core 0 is able to trigger an event on core 1.

26.9.6 Write Protection Mode Register

Name: MATRIX_WPMR

Address: 0x400E03E4 (0), 0x480101E4 (1)

Access: Read/Write

Reset: See Table 26-8

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

For more information on Write Protection registers, refer to Section 26.8 “Register Write Protection”.

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x4D4154 (“MAT” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x4D4154 (“MAT” in ASCII).

See Section 26.8 “Register Write Protection” for the list of registers that can be protected.

- **WPKEY: Write Protection Key**

Value	Name	Description
0x4D4154	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

26.9.7 Write Protection Status Register

Name: MATRIX_WPSR

Address: 0x400E03E8 (0), 0x480101E8 (1)

Access: Read-only

Reset: See Table 26-8

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 information on Write Protection registers, refer to Section 26.8 “Register Write Protection”.

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of the MATRIX_WPMR register.

1: A write protection violation has occurred since the last write of the MATRIX_WPMR 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.

27. G3 Power Line Communications (GPLC)

27.1 Description

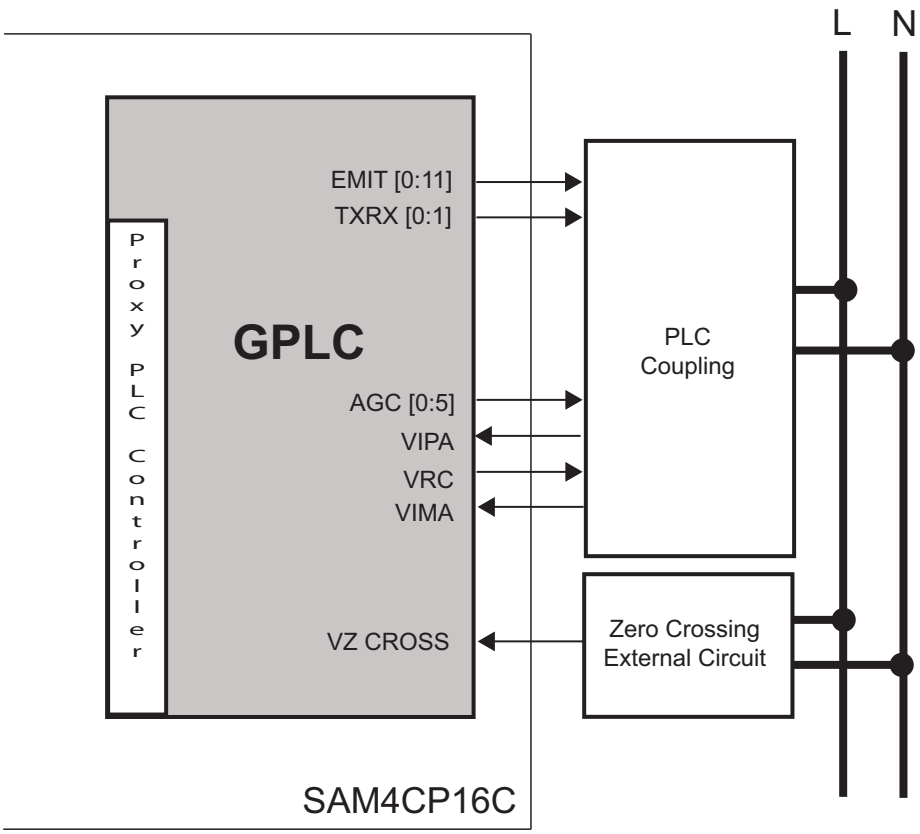
GPLC is an embedded G3-PLC modem for Power Line Communication. Its flexible architecture, composed of hardware accelerators and coprocessors, achieves a very efficient G3 PHY layer implementation.

27.2 Embedded Characteristics

- G3-PLC modem
  - Implements G3 CENELEC-A, FCC and ARIB profiles (ITU-T G.9903, June ’14)
  - Power Line Carrier Modem for 50 Hz and 60 Hz mains
  - G3-PLC coherent and differential modulation schemes available
- Automatic Gain Control and continuous amplitude tracking in signal reception
- Zero cross detection
- Embedded PLC Analog Front End (AFE), requires only external discrete high efficient Class D Line Driver for signal injection

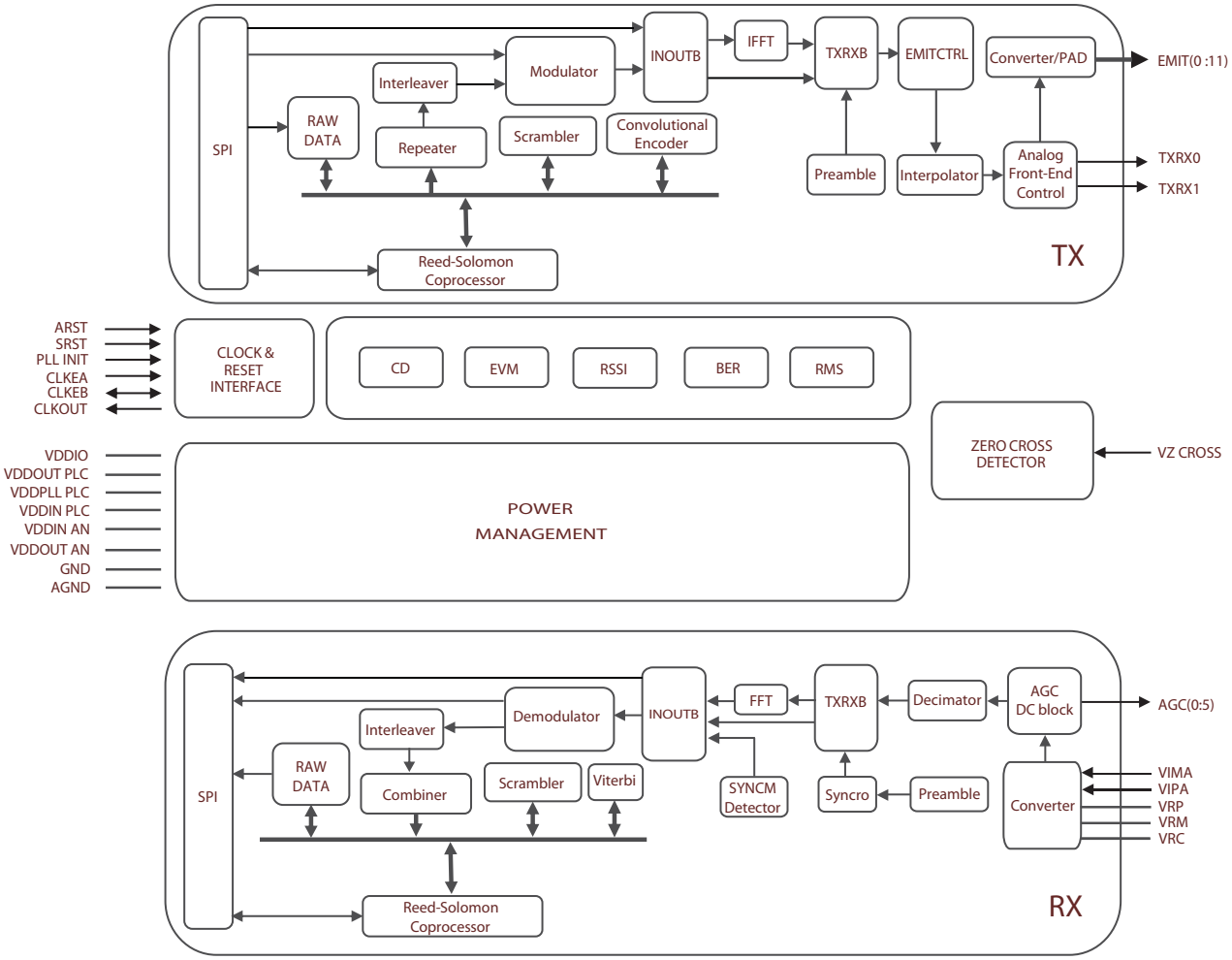
27.2.1 GPLC Application Block Diagram

Figure 27-1. GPLC application example



27.3 Block Diagram

Figure 27-2. GPLC Functional Block Diagram



27.4 Signal Description

Table 27-1. Signal Description

Signal Name	Function	Type
VIMA	Negative Differential Voltage Input	Input
VIPA	Positive Differential Voltage Input	Input
VRP	Internal Reference “Plus” Voltage. Connect an external decoupling capacitor between VRP and VRM (1nF - 100nF)	Output
VRM	Internal Reference “Minus” Voltage. Connect an external decoupling capacitor between VRP and VRM (1nF - 100nF)	Output
VRC	Common-mode Voltage. Bypass to analog ground with an external decoupling capacitor (100pF - 1nF)	Output
EMIT0 - EMIT11	PLC Transmission ports	Output
AGC0 - AGC5	PLC Automatic Gain Control: • These digital tri-state outputs are managed by ACG hardware logic to drive external circuitry when input signal attenuation is needed	Output
TXRX0 - TXRX1	PLC Ext. coupling TxRx control	Output
VZ CROSS	Mains Zero-Cross Detection Signal: • This input detects the zero-crossing of the mains voltage	Input
ARST	PLC Asynchronous Reset: • ARST is active low • Internal configuration: 33 kΩ typ. pull up resistor	Input
SRST	PLC Synchronous Reset • SRST is active low • Internal configuration: 33 kΩ typ. pull up resistor	Input
PLL INIT	PLL Initialization Signal • PLL INIT is active low • Internal configuration: 33 kΩ typ. pull up resistor	Input
CLKEA	PLC External Clock Input • CLKEA must be connected to one terminal of a crystal (when a crystal is being used) or used as input for external clock signal	Input
CLKEB	PLC External Clock Input/Output • CLKEB must be connected to one terminal of a crystal (when a crystal is being used) or must be floating when an external clock signal is connected thru CLKEA	I/O
CLKOUT	12 MHz External CLK Output	Output
VDDIO	PLC Digital pads 3.3V Power Supply	Power
VDDPLL PLC	PLC PLL Power Supply	Power
VDDIN PLC	PLC Digital Regulator input	Power
VDDOUT PLC	PLC Digital Regulator output	Power
VDDIN AN	PLC Analog Regulator input	Power
VDDOUT AN	PLC Analog Regulator output	Power
GND	Digital Ground	Power
AGND	Analog Ground	Power

27.5 Analog Front-End

27.5.1 PLC coupling circuitry description

Atmel PLC coupling reference designs have been designed to achieve high performance, low cost and simplicity.

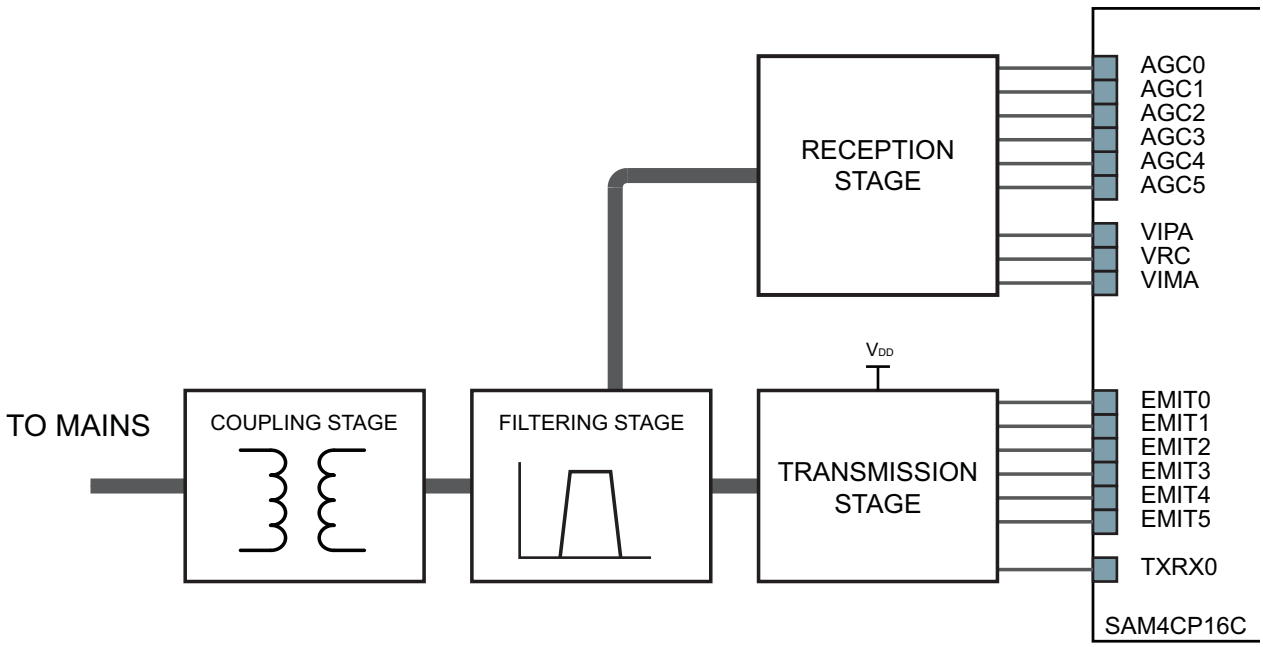
With these values on mind, Atmel has developed a set of PLC couplings covering frequencies below 500 kHz compliant with different applicable regulations.

Atmel PLC technology is purely digital and does not require external DAC/ADC, thus simplifying the external required circuitry. Generally Atmel PLC coupling reference designs make use of few passive components plus a Class D amplification stage for transmission.

All PLC coupling reference designs are generally composed by the same sub-circuits:

- Transmission Stage
- Reception Stage
- Filtering Stage
- Coupling Stage

Figure 27-3. PLC coupling block diagram



A particular reference design can contain more than one sub-circuit of the same kind (i.e., two transmission stages).

27.5.1.1 Transmission Stage

The transmission stage adapts the EMIT signals and amplifies them if required. It can be composed by:

- Driver: A group of resistors which adapt the EMIT signals to either control the Class-D amplifier or to be filtered by the next stage.
- Amplifier: If required, a Class-D amplifier which generates a square waveform from 0 to V_{DD} is included.
- Bias and protection: A couple of resistors and a couple of Schottky barrier diodes provide a DC component and provide protection from received disturbances.

Transmission stage shall be always followed by a filtering stage.

27.5.1.2 Filtering Stage

The filtering stage is composed by band-pass filters which have been designed to achieve high performance in field deployments complying at the same time with the proper normative and standards.

The in-band flat response filtering stage does not distort the injected signal, reduces spurious emission to the limits set by the corresponding regulation and blocks potential interferences from other transmission channels.

The Filtering stage has three aims:

- Band-pass filtering of high frequency components of the square waveform generated by the Transmission Stage.
- Adapt Input/Output impedances for optimal reception/transmission. This is controlled by TXRX signal.
- In some cases, Band-pass filtering for received signals.

When the system is intended to be connected to a physical channel with high voltage or which is not electrically referenced to the same point then the filtering stage must be always followed by a coupling stage.

27.5.1.3 Coupling Stage

The coupling stage blocks the DC component of the line to/from which the signal is injected/received (i.e., 50/60 Hz of the mains). This is carried out by a high voltage capacitor.

Coupling stage could also electrically isolate the coupling circuitry from the external world by means of a 1:1 transformer.

27.5.1.4 Reception Stage

The reception stage adapts the received analog signal to be properly captured by the GPLC internal reception chain. Reception circuit is independent of the PLC channel which is being used. It basically consists of:

- Anti aliasing filter (RC Filter)
- Automatic Gain Control (AGC) circuit
- Driver of the internal ADC

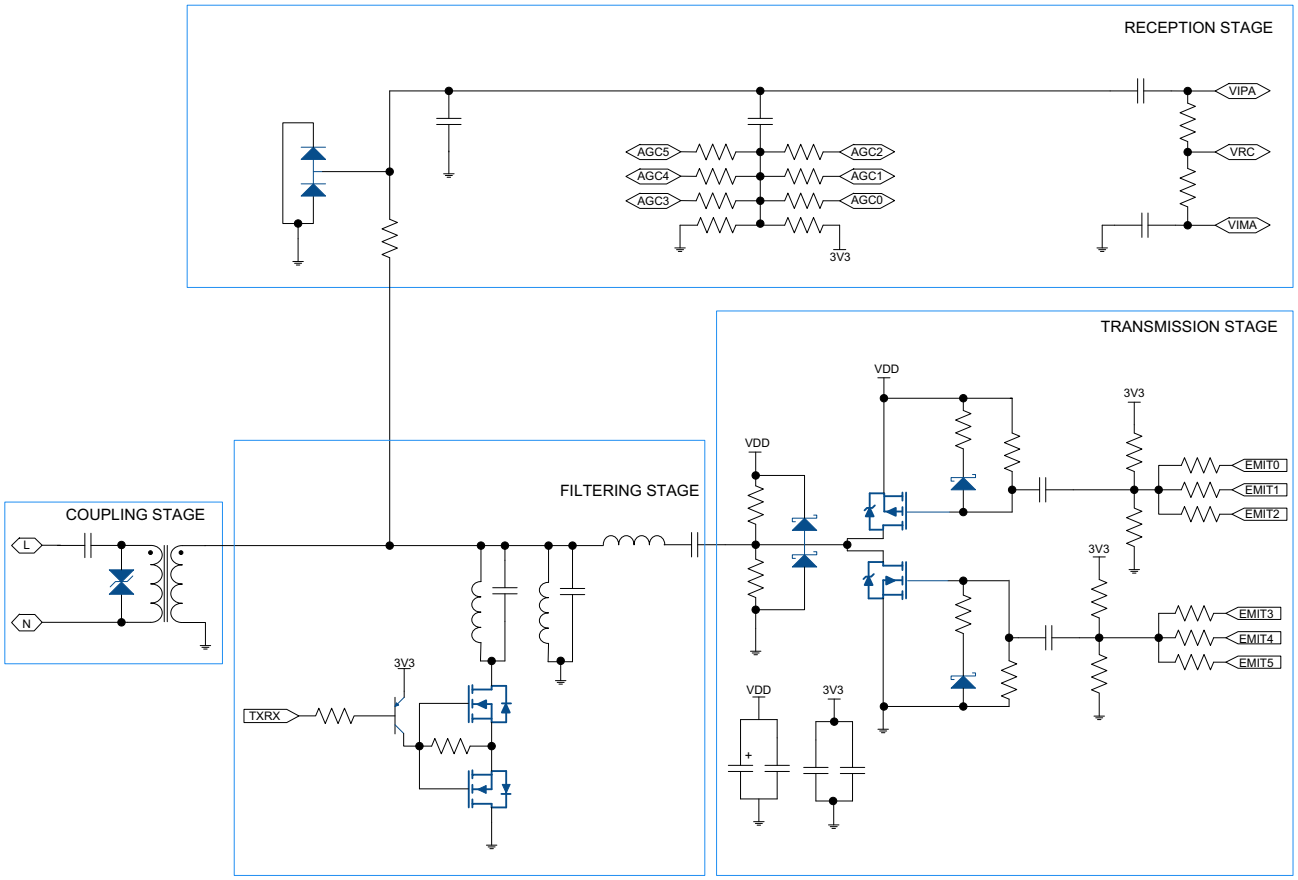
The AGC circuit avoids distortion on the received signal that may arise when the input signal is high enough to polarize the protective diodes in direct region.

The driver to the internal ADC comprises a couple of resistors and a couple of capacitors. This driver provides a DC component and adapts the received signal to be properly converted by the internal reception chain.

27.5.1.5 Generic PLC Coupling

Please consider that this is a generic PLC Coupling design for a particular application please refer to Atmel doc43052 “PLC Coupling Reference Designs”.

Figure 27-4. PLC Coupling block diagram detailed



27.5.2 ATPLCOUP reference designs

Atmel provides PLC coupling reference designs for different applications and frequency bands up to 500 kHz. Please refer to Atmel doc43052 “PLC Coupling Reference Designs” for a detailed description.

27.5.3 Zero-crossing detector

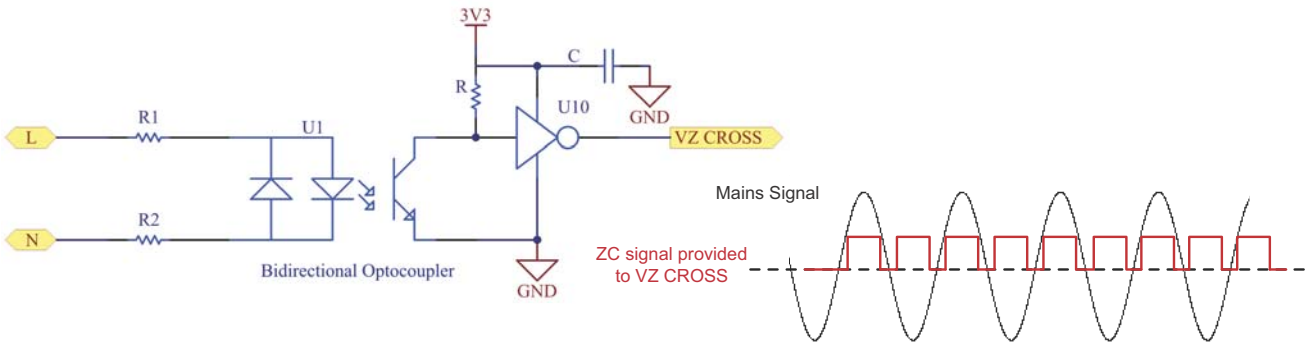
27.5.3.1 Overview

Zero Crossing Detector block works predicting future zero crossing in function of the past zero crossings. To achieve this, the system embeds a configurable Input Signal Management (ISM) block and a PLL, both of which manage Zero Crossing Detector Input Signal to calculate Zero Crossing Output Flag. The zero-cross detection of waves of 50 Hz and 60 Hz with ±10% of error is supported.

The PLL block interprets its input signal such a way that it indicates a zero cross in the middle of a positive pulse. It is important to note that depending on the external circuit which implements the Zero Crossing Detector Input Signal this interpretation is not always correct. So for these cases it is required to transform the Input Signal in a signal where the middle of a positive pulse corresponds to a truly zero cross. This transformation is implemented through the Input Signal Management (ISM) configured by MODE_INV and MODE_REP fields in ZC_CONFIG register.

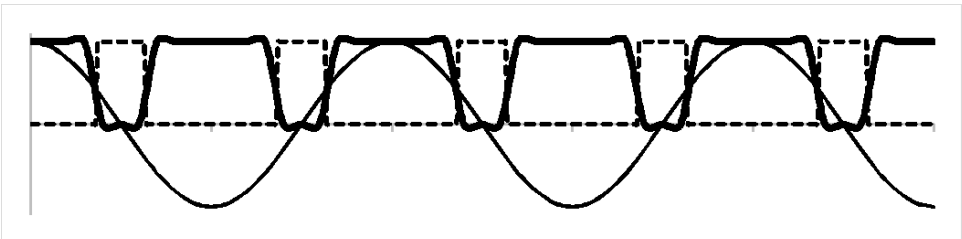
Zero Crossing Detector Input Signal (VZ CROSS) must fulfil some requirements. The first requirement is that VZ CROSS signal must be a pulse train which its duty cycle must be >60% or <40% (polarity is configurable). In addition, if we have to detect Ascent or Descent zero-crossing, Zero Crossing Detector Input Signal period must be equal than period of the wave we need to obtain zero-crossing. Ascent and Descent Zero Crossing Detection are configured by setting MODE_MUX and MODE_ASC fields in ZC_CONFIG register.

Figure 27-5. Typical circuit, using a bidirectional optocoupler and a Schmitt trigger



The input signal “VZ CROSS”(wider line) generated by this circuit for Zero Cross Detection of the wave “L”-“N” (finer line) is plotted in next figure. The digital signal at output of Input Signal Management (ISM) is plotted in Figure 27-6.

Figure 27-6. Digital signal (dashed line) at output of Input Signal Management (ISM) internal block

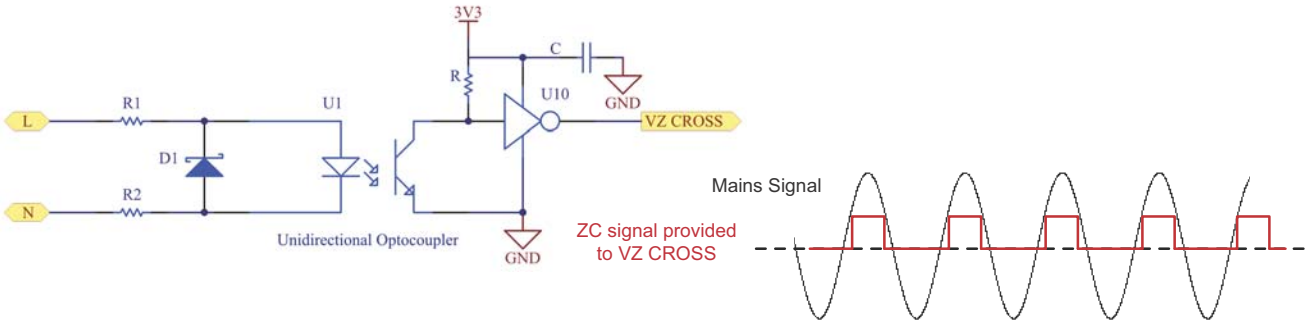


For this circuit, Zero Cross Internal registers should be configured this way:

ZC_CONFIG.MODE_MUX = ‘0’
ZC_CONFIG.MODE_ASC = ‘0’
ZC_CONFIG.MODE_INV = ‘1’
ZC_CONFIG.MODE_REP = ‘0’
ZC_CONFIG.FILTER_BP = ‘0’

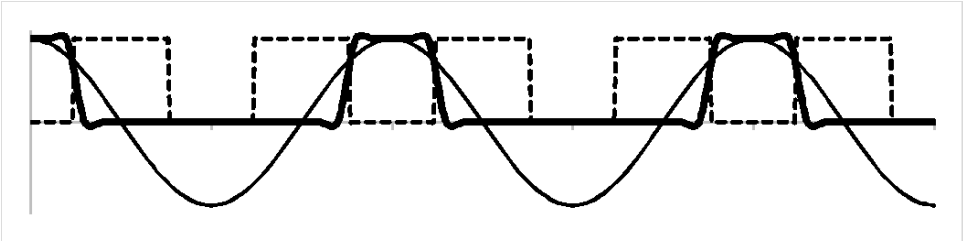
Some situations (for example in some protocols like G3) could require only ascent (or descent) mains signal zero-crossings to be detected. When we have to detect Ascent or Descent Zero Cross of the wave (finer line), the circuit should generate an input signal “VZ CROSS” (wider line) with the same period, as specified in next figure. This could be easily implemented by using an unidirectional optocoupler or a Zener diode topology in the external circuitry.

Figure 27-7. Typical circuit, using a unidirectional optocoupler and a Schmitt trigger



The digital signal at output of Input Signal Management (ISM) is plotted in Figure 27-8:

Figure 27-8. Digital signal (dashed line) at output of Input Signal Management (ISM) internal block



For this case, Zero Cross Internal registers should be configured this way:

ZC_CONFIG.MODE_MUX = ‘1’
ZC_CONFIG.MODE_ASC = ‘0’(ascent) or ‘1’(descent)
ZC_CONFIG.MODE_INV = ‘1’
ZC_CONFIG.MODE_REP = ‘1’
ZC_CONFIG.FILTER_BP = ‘0’

27.5.3.2 Zero Crossing Config register

Name: ZC_CONFIG

Address: 0x4A0

Access: Read/Write

Reset: 0x00023210

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	Reserved	PEAK2_ZC_EN	PEAK1_ZC_EN
15	14	13	12	11	10	9	8
-	FILTER_NUM [6:0]						
7	6	5	4	3	2	1	0
-	-	-	FILTER_BP	MODE_REP	MODE_INV	MODE_ASC	MODE_MUX

- **MODE_MUX:** Zero Crossing Mode
  '0': Selection of both ascent and descent zero-crossing
  '1': Selection of ascent or descent zero-crossing
- **MODE_ASC:** Ascent-Descent Mode
  '0': If MODE_MUX is 1, Ascent Zero Crossing
  '1': If MODE_MUX is 1, Descent Zero Crossing
- **MODE_INV:** Inversion Mode
  '0': No effect.
  '1': Zero Crossing Detector Input Signal is inverted.
- **MODE_REP:** Repetition Mode
  '0': No effect.
  '1': Zero Crossing Detector Input Signal period is down by half.
- **FILTER_BP:** Zero Crossing Input Signal Filter Enable
  '0': Filter enabled.
  '1': Filter not enabled.
- **FILTER_NUM[6:0]:** Zero Crossing Input Signal Filter Parameter
  Time (counted in number of clock cycles) that the Zero Crossing Input Signal (1-bit) must be constant to set that value as the input signal for Zero Crossing Detection. Used to refuse fast transitions in Zero Crossing Input Signal.
- **PEAK1_ZC_EN:** indicates if PEAK_ZC_TIME updates its value with the last ZC_TIME when a PEAK1 is detected. It is active high.
- **PEAK2_ZC_EN:** indicates if PEAK_ZC_TIME updates its value with the last ZC_TIME when a PEAK2 is detected. It is active high.

28. Peripheral DMA Controller (PDC)

28.1 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.

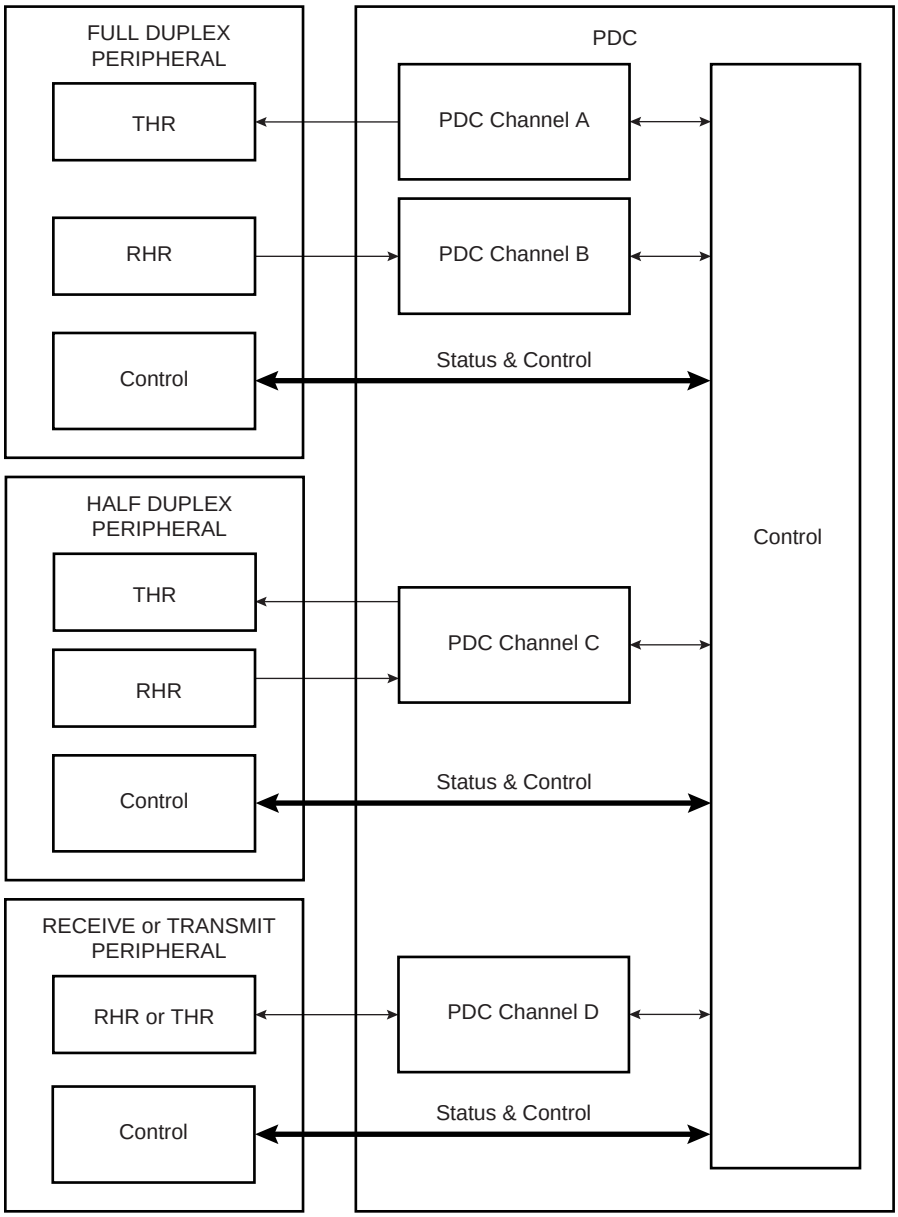
To launch a transfer, the peripheral triggers its associated PDC channels by using transmit and receive signals. When the programmed data is transferred, an end of transfer interrupt is generated by the peripheral itself.

28.2 Embedded Characteristics

- Performs Transfers to/from APB Communication Serial Peripherals.
- Supports Half-duplex and Full-duplex Peripherals.

28.3 Block Diagram

Figure 28-1. Block Diagram



28.4 Functional Description

28.4.1 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 28.4.3 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.

28.4.2 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.

28.4.3 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).

28.4.4 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.

28.4.5 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.

28.4.5.1 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.

28.4.5.2 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.

28.4.5.3 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.

28.4.5.4 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.

28.5 Peripheral DMA Controller (PDC) User Interface

Table 28-1. Register Mapping

Offset	Register	Name[1]	Access	Reset
0x00	Receive Pointer Register	PERIPH_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 configured by the user depending on the function and the desired peripheral.

28.5.1 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.

28.5.2 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.

28.5.3 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.

28.5.4 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.

28.5.5 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.

28.5.6 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.

28.5.7 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.

28.5.8 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.

28.5.9 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.

28.5.10 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.

29. Clock Generator

29.1 Description

The Clock Generator User Interface is embedded within the Power Management Controller and is described in Section 30.18 ”Power Management Controller (PMC) User Interface”. However, the Clock Generator registers are named CKGR_.

29.2 Embedded Characteristics

The Clock Generator is made up of:

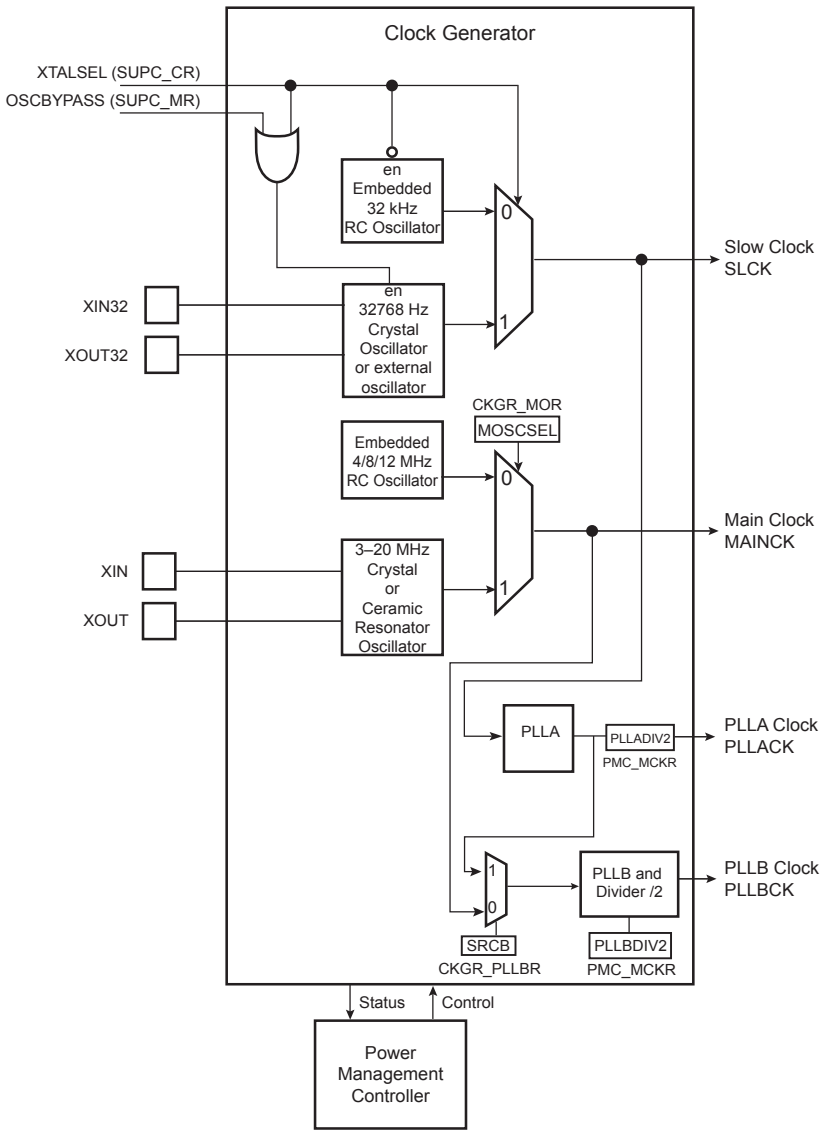
- A low-power 32.768 kHz crystal oscillator with Bypass mode.
- A low-power embedded 32 kHz (typical) RC oscillator.
- A 3 to 20 MHz crystal or ceramic resonator-based oscillator with Bypass mode.
- A factory-trimmed embedded RC oscillator. Three output frequencies can be selected: 4/8/12 MHz. By default 4 MHz is selected.
- Two programmable PLLs, (PLLA input from 32 kHz, output clock range 8 MHz and PLLB input from 3 to 32 MHz, output clock range 80 to 240 MHz), capable of providing the clock MCK to the processor and to the peripherals.

It provides the following clocks:

- SLCK: Slow clock. The only permanent clock within the system.
- MAINCK: output of the Main clock oscillator selection: either the crystal or ceramic resonator-based oscillator or 4/8/12 MHz RC oscillator.
- PLLACK: output of the 8 MHz programmable PLL (PLLA).
- PLLBCK: output of the divider and 80 to 240 MHz programmable PLL (PLLB).

29.3 Block Diagram

Figure 29-1. Clock Generator Block Diagram



29.4 Slow Clock

The Supply Controller embeds a Slow clock generator that is supplied with the VDDBU power supply. As soon as VDDBU is supplied, both the 32.768 kHz crystal oscillator and the embedded 32 kHz (typical) RC oscillator are powered up, but only the RC oscillator is enabled.

The Slow clock is generated either by the 32.768 kHz 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.

29.4.1 Embedded 32 kHz (typical) RC Oscillator

By default, the embedded 32 kHz (typical) RC oscillator is enabled and selected as a source of Slow clock. The user has to take into account the possible drifts of this oscillator. Refer to the section “DC Characteristics”.

This oscillator is disabled by clearing SUPC_CR.XTALSEL.

29.4.2 32.768 kHz Crystal Oscillator

The Clock Generator integrates a low-power 32.768 kHz crystal oscillator. To use this oscillator, the XIN32 and XOUT32 pins must be connected to a 32.768 kHz crystal. Refer to the section “Electrical Characteristics” for appropriate loading capacitor selection on XIN32 and XOUT32.

Note that the user is not obliged to use the 32.768 kHz crystal oscillator and can use the 32 kHz (typical) RC oscillator instead.

The 32.768 kHz crystal oscillator provides a more accurate frequency than the 32 kHz (typical) RC oscillator.

To select the 32.768 kHz crystal oscillator as the source of the Slow clock, the bit SUPC_CR.XTALSEL must be set. This results in a sequence which enables the 32.768 kHz 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 VDDBU power supply. If the user does not need the 32.768 kHz crystal oscillator, the XIN32 and XOUT32 pins can be left unconnected.

The user can also set the 32.768 kHz crystal oscillator in Bypass mode instead of connecting a crystal. In this case, the user must provide the external clock signal on XIN32. For input characteristics of the XIN32 pin, refer to 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.

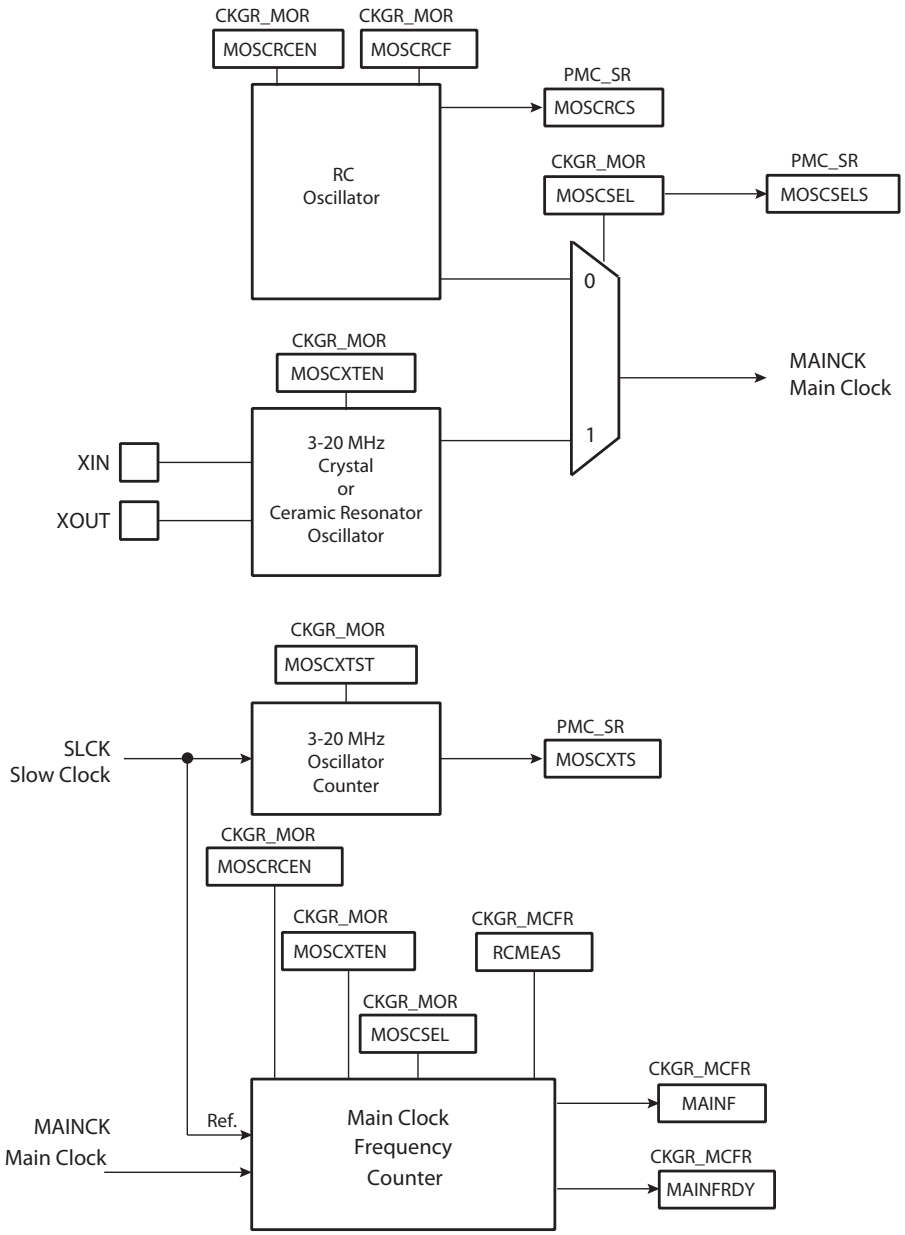
29.5 Main Clock

The Main clock has two sources:

- A 4/8/12 MHz RC oscillator with a fast startup time and that is selected by default to start the system.
- A 3 to 20 MHz crystal or ceramic resonator-based oscillator with Bypass mode.

Figure 29-2 shows the Main clock block diagram.

Figure 29-2. Main Clock Block Diagram



29.5.1 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, PMC_SR.MOSCRCS 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) triggers an interrupt to the processor.

When Main clock (MAINCK) is not used to drive the processor and frequency monitor (SLCK or PLLACK is 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.

29.5.2 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, SEL4 bit must be set to 1 and a frequency value must be configured in the field CAL4. Likewise, SEL8/12 bits must be set to 1 and a trim value must be configured in the fields CAL8/12 in order to adjust the other frequencies of the RC oscillator.

It is possible to adjust the RC oscillator frequency while operating from this clock. For example, when running on lowest frequency it is possible to change the CAL4 value if PMC_OCR.SEL4 bit is set.

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. The software can calculate the error with an expected frequency and correct the CAL4 (or CAL8/CAL12) field accordingly. This may be used to compensate frequency drift due to derating factors such as temperature and/or voltage.

29.5.3 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 startup time counter. The startup time depends on the characteristics of the external device connected to this oscillator. Refer to the section “Electrical Characteristics” for the startup time.

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 startup time is about 62 ms.

When the startup 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) triggers an interrupt to the processor.

29.5.4 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 startup 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 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 or PLLBCK 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.

29.5.5 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. Refer to 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 CKGR_MOR.MOSCXTEN.

29.5.6 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 CKGR_MCFR.RCMEAS 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, CKGR_MCFR.MAINFRDY is set and the counter stops counting. Its value can be read in CKGR_MCFR.MAINF 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.

29.5.7 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.

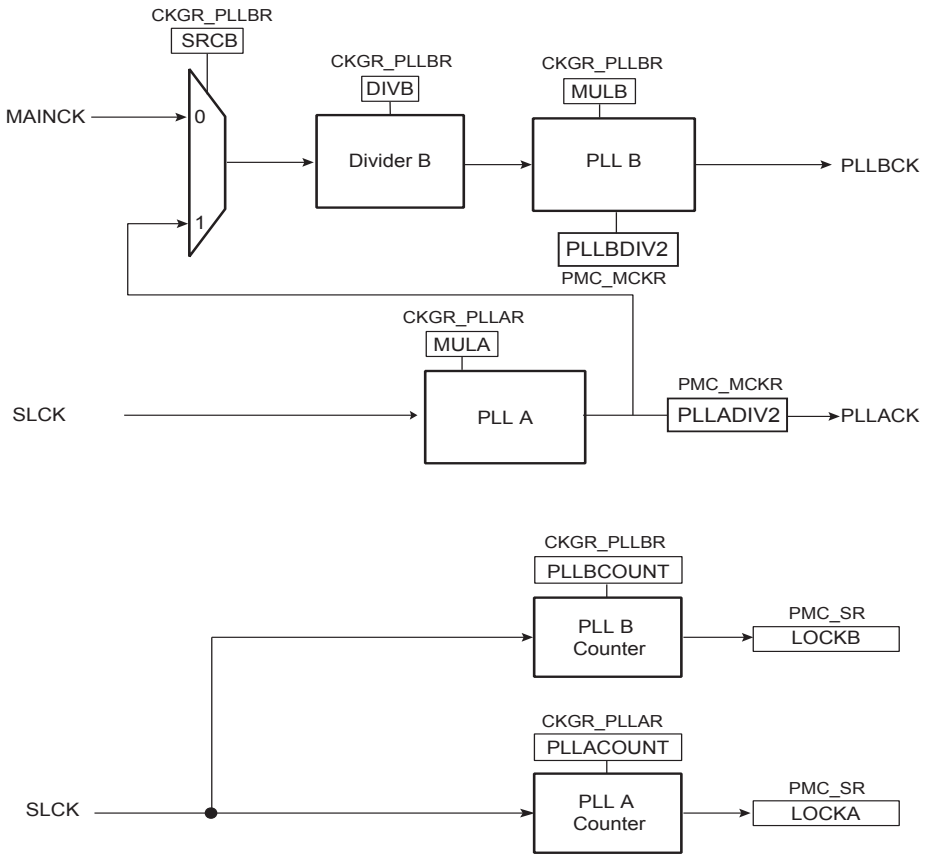
If switching to the crystal oscillator, a check must be carried out to ensure that the oscillator is present and that its frequency is valid. Follow the sequence below:

1. Select the Slow clock as MCK by configuring 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 startup time as defined in the section “Electrical Characteristics”.
4. Wait for PMC_SR.MOSCXTS flag to rise, indicating the end of a startup 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.

29.6 Divider and PLL Blocks

The device features one divider block and two PLL blocks that permit a wide range of frequencies to be selected on either the Master clock, the Processor clock or the Programmable clock outputs. Figure 29-3 shows the block diagram of the divider and PLL blocks.

Figure 29-3. Divider and PLL Blocks Diagram



29.6.1 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 PLLs (PLLA, PLLB) allow multiplication of the SLCK clock source for PLLA or divided MAINCK or PLLA output clock for PLLB. The PLL clock signal has a frequency that depends on the respective source signal frequency and on the parameters DIV (DIVB) and MUL (MULA, MULB) and PLLEN (PLLAEN). The factor applied to the source signal frequency is (MUL + 1)/DIV. When MUL is written to 0 or PLLEN=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 PLLA(B)EN higher than 0.

To change the frequency of the PLLA, the PLLA must be first disabled by writing 0 in the MULA field and 0 in PLLACOUNT field. Then, wait for two SLCK clock cycles before configuring the PLLA to generate the new frequency by programming a new multiplier in MULA and the PLLACOUNT field in the same register access. See the “Electrical Characteristics” to get the PLLACOUNT values covering the PLL transient time.

Whenever the PLL is re-enabled or one of its parameters is changed, PMC_SR.LOCK (LOCKA, LOCKB) is automatically cleared. The values written in the PLLCOUNT field (PLLACOUNT, PLLBCOUNT) in CKGR_PLLR (CKGR_PLLAR, CKGR_PLLBR) 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, PMC_SR.LOCK is set and triggers 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 PMC_MCKR.PLLDIV2 (PLLADIV2, PLLBDIV2).

The PLLADIV2 has no effect on PLLB clock input because the output of the PLLA is directly routed to PLLB input selection.

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 PMC_MCKR.CSS.
2. Change the frequency (MOSCRCF) or oscillator selection (MOSCSEL) in CKGR_MOR.
3. Wait for PMC_SR.MOSCRCS (if frequency changes) or PMC_SR.MOSCSELS (if oscillator selection changes).
4. Disable and then enable the PLL.
5. Wait for PMC_SR.LOCK flag.
6. Switch back to the PLL by writing the appropriate value to PMC_MCKR.CSS.

30. Power Management Controller (PMC)

30.1 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 32.768 kHz 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.

30.2 Embedded Characteristics

The PMC provides the following clocks:

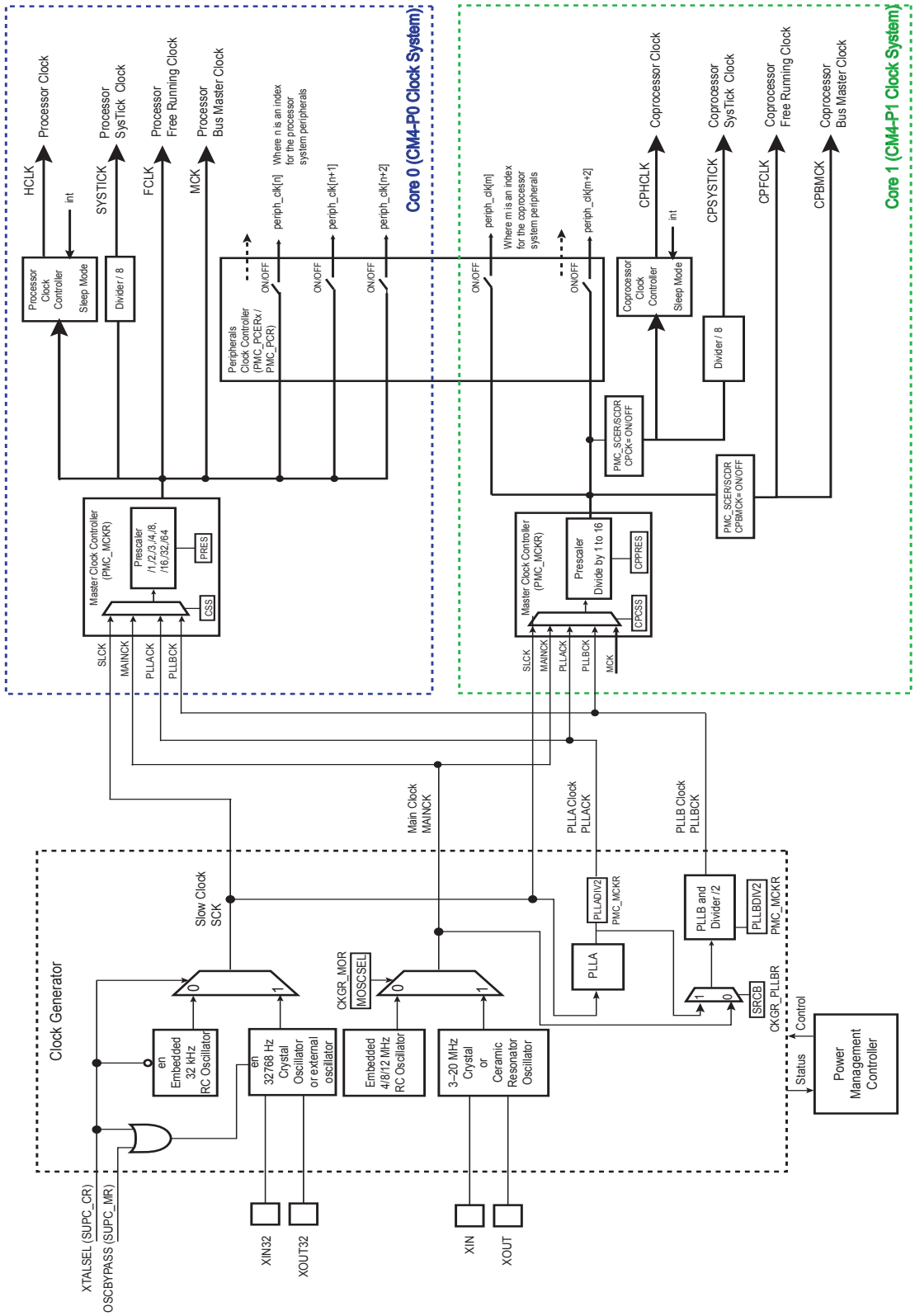
- Master Clock (MCK): programmable from a few hundred Hz to the maximum operating frequency of the device. It is available to the modules running permanently, such as the Enhanced Embedded Flash Controller.
- Processor Clock (HCLK) and Coprocessor (second processor) Clock (CPHCLK): automatically switched off when entering the processor in Sleep Mode.
- Free running Processor Clock (FCLK) and Free running Coprocessor Clock (CPFCLK).
- One SysTick external clock for each Cortex-M4 core.
- 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 32.768 kHz crystal oscillator frequency monitor.
- A frequency counter on Main clock.
- An on-the-fly adjustable 4/8/12 MHz RC oscillator frequency.

30.3 Block Diagram

Figure 30-1. General Clock Block Diagram



30.4 Master Clock Controller

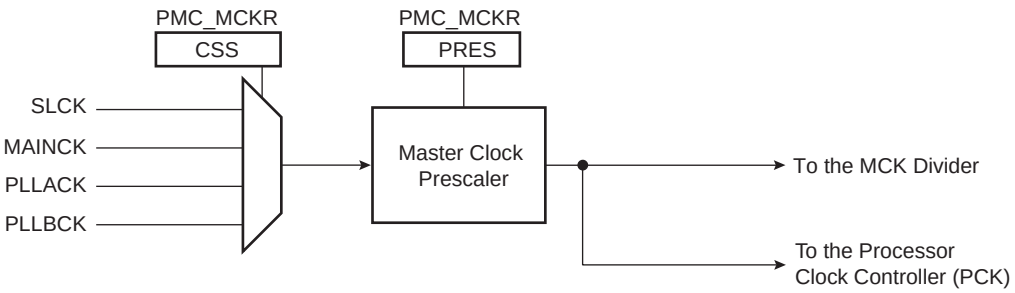
The Master Clock Controller provides selection and division of the Master clock (MCK) and coprocessor Master clock (CPMCK). MCK is the source clock of the peripheral clocks in the subsystem 0 and CPMCK is the source of the peripheral clocks in the subsystem 1. The Master Clock is selected from one of the clocks provided by the Clock Generator.

Selecting the Slow clock provides a Slow clock signal to the whole device. Selecting the Main clock saves power consumption of the PLLs. The Master Clock Controller is made up of a clock selector and a prescaler.

The Master clock selection is made by writing PMC_MCKR.CSS/CPCSS (Clock Source Selection/Coprocessor Clock Source Selection). The prescaler supports the division by a power of 2 of the selected clock between 1 and 64, and the division by 3. The PMC_MCKR.PRES/CPPRES programs the prescaler.

Each time PMC_MCKR is written to define a new Master clock, PMC_SR.MCKRDY is cleared. It reads 0 until the Master clock is established. Then, MCKRDY 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 30-2. Master Clock Controller



30.5 Processor Clock Controller

The PMC features a Processor Clock Controller (HCLK) and a Coprocessor Clock Controller (CPHCLK) that implements the Processor Sleep mode. The processor clocks 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 is enabled after a reset and is automatically re-enabled by any enabled interrupt. The Coprocessor Clock Controller is disabled after reset. It is up to the Master application to enable the CPHCLK. Similar to HCLK, CPHCLK is automatically re-enabled by any enabled instruction after having executed a WFI instruction. 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.

30.6 SysTick Clock

The SysTick calibration value is fixed to 8000 which allows the generation of a time base of 1 ms with SysTick clock to the maximum frequency on MCK divided by 8.

30.7 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).

If the peripherals located on the coprocessor system bus require data exchange with the co-processor or the main processor, the CPBMCK clock must be enabled prior to enable any co-processor peripheral clock.

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.

30.8 Free Running Processor Clock

The Free Running Processor Clock (FCLK) together with the Free Running Coprocessor Master clock (CPFCLK) used for sampling interrupts and clocking debug blocks ensures that interrupts can be sampled, and sleep events can be traced, while the processors are sleeping. It is connected to Master clock (MCK)/Coprocessor Master clock (CPMCK).

30.9 Programmable Clock Output Controller

The PMC controls 3 signals to be outputs 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), the PLLB clock (PLLBCK), and the Master clock (MCK) by writing PMC_PCKx.CSS. Each output signal can also be divided by a power of 2 between 1 and 64 by writing PMC_PCKx.PRES.

Each output signal can be enabled and disabled by writing a 1 to the corresponding PMC_SCER.PCKx and PMC_SCDR.PCKx, respectively. Status of the active programmable output clocks are given in PMC_SCSR.PCKx.

PMC_SR.PCKRDYx status flag indicates that the Programmable clock is actually what has been programmed in the Programmable clock registers.

As the Programmable Clock Controller does not manage with glitch prevention when switching clocks, it is strongly recommended to disable the Programmable clock before any configuration change and to re-enable it after the change is actually performed.

30.10 Main Processor Fast Startup

At exit from Wait mode, the device allows the main 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 30.11 ”Main Processor 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 (PMC_MCKR.CSS 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 CKGR_MOR.WAITMODE, or by executing the WaitForEvent (WFE) instruction of the processor while the LPM bit is at 1 in the Fast Startup Mode register (PMC_FSMR). Immediately after setting the WAITMODE bit or using the WFE instruction, wait for PMC_SR.MCKRDY to be set.

In case of dual core activity, it is recommended to check the coprocessor state before instructing the main processor to enter Wait mode.

A Fast Startup occurs upon the detection of a programmed level on one of the wake-up input pins WKUPx (For the number of inputs, refer to section “Peripheral Signal Multiplexing on I/O Lines”) or upon an active alarm from the RTC and RTT. 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.

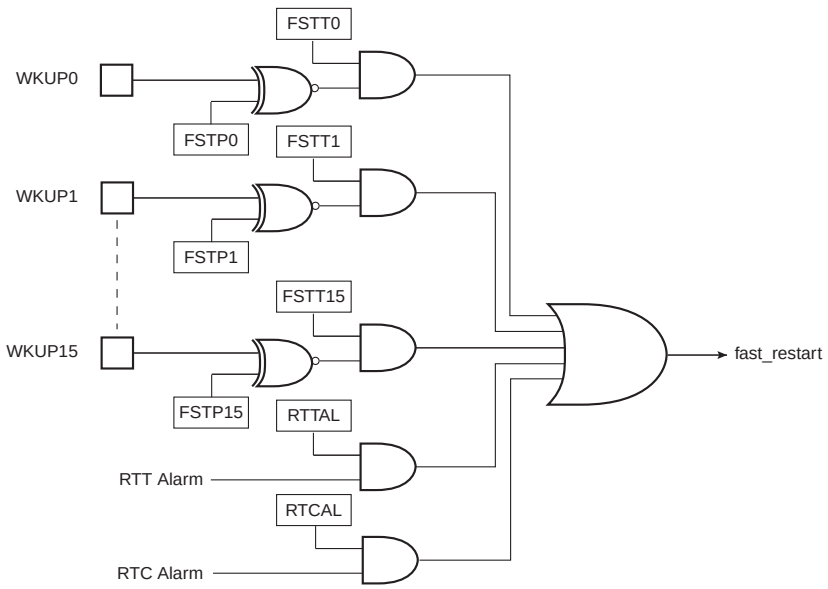
The Fast Startup circuitry, as shown in Figure 30-3, is fully asynchronous and provides a fast startup signal to the PMC. As soon as the fast startup signal is asserted, the embedded 4/8/12MHz RC oscillator restarts automatically.

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 PMC_FSMR.FLPM. FLPM 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 PMC_FSMR.FLPM. If 0 is programmed (Standby mode), the power consumption is slightly higher than in Deep-power-down mode.

When programming 2 in PMC_FSMR.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 30-3. 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. The status can be read in the PIO Controller and the status registers of the RTC and RTT.

30.11 Main Processor Startup from Embedded Flash

The inherent startup time of the embedded Flash cannot provide a fast startup of the system.

If system fast startup 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 22. ”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 30.10 ”Main Processor Fast Startup”).

30.12 Coprocessor Sleep Mode

The coprocessor enters Sleep mode by executing the WaitForInterrupt (WFI) instruction of the coprocessor. Any enabled interrupt can wake the processor up.

30.13 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 configuring CKGR_MOR.CFDEN. It is also disabled in either of the following cases:

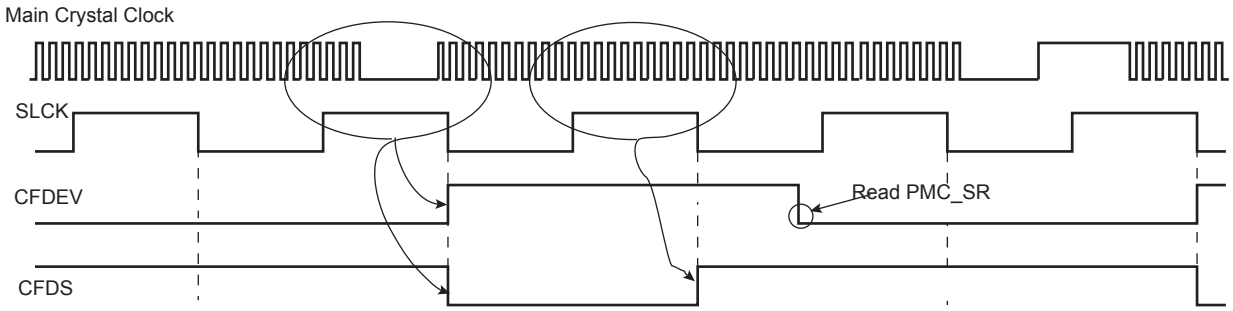
- after a VDDCORE reset
- when the 3 to 20 MHz crystal oscillator is disabled (MOSCXTEN = 0)

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, PMC_SR.CFDEV 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 current status of the clock failure detection can be read at any time from PMC_SR.CFDS.

Figure 30-4. 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 (CKGR_MOR.MOSCSEL = 1), and if MCK source is PLLACK or PLLBCK (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.

Two 32 kHz (typical) RC oscillator clock cycles are necessary 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 or PLLBCK.

The current status of the clock failure detection can be read at any time from PMC_SR.FOS.

This fault output remains active until the defect is detected and until it is cleared by the bit FOCLR in the PMC Fault Output Clear register (PMC_FOCR).

30.14 32.768 kHz Crystal Oscillator Frequency Monitor

The frequency of the 32.768 kHz 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 CKGR_MOR.XT32KFME. The PMC_OCR.SEL4/SEL8/SEL12 must be cleared.

An error flag (PMC_SR.XT32KERR) is asserted when the 32.768 kHz crystal oscillator frequency is out of the +/-10% nominal frequency value (i.e., 32.768 kHz). 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 4 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 32.768 kHz crystal oscillator frequency deviation over +/-10% of the nominal frequency is systematically reported as an error by the PMC_SR.XT32KERR. Between -1% and -10% and +1% and +10%, the error is not systematically reported.

Thus only a crystal running at 32.768 kHz 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 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 PMC_SR.MOSCRCS is set.

The error flag can be defined as an interrupt source of the PMC by setting PMC_IER.XT32KERR.

30.15 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 CKGR_MOR.MOSCXTEN:

   The user can define a startup time. This is done by writing a value in CKGR_MOR.MOSCXTST. Once this register has been correctly configured, the user must wait for PMC_SR.MOSCXTS to be set. This is done either by polling PMC_SR.MOSCXTS, or by waiting for the interrupt line to be raised if the associated interrupt source (PMC_IER.MOSCXTS) has been enabled.
3. Switch the MAINCK to the 3 to 20 MHz crystal oscillator by setting CKGR_MOR.MOSCSEL.
4. Wait for PMC_SR.MOSCSELS to be set 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 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 CKGR_MOR.MOSCSEL. 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 or B.

   All parameters needed to configure PLLx and the divider are located in CKGR_PLLxR.

   The DIVx field is used to control the divider. 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 254. 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 PMC_SR.LOCKx to be set. This can be done either by polling PMC_SR.LOCKx or by waiting for the interrupt line to be raised if the associated interrupt source (PMC_IER.LOCKx) has been enabled. 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 PMC_SR.MCKRDY to be set. This can be done either by polling PMC_SR.MCKRDY or by waiting for the interrupt line to be raised if the associated interrupt source (PMC_IER.MCKRDY) has been enabled. 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 PMC_MCKR.PRES.
  - Wait for PMC_SR.MCKRDY to be set.
  - Program PMC_MCKR.CSS.
  - Wait for PMC_SR.MCKRDY to be set.
- If a new value for CSS field corresponds to Main clock or Slow clock,
  - Program PMC_MCKR.CSS.
  - Wait for PMC_SR.MCKRDY to be set.
  - Program PMC_MCKR.PRES.
  - Wait for PMC_SR.MCKRDY to be set.

If CSS or PRES are modified at any stage, MCKRDY 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 and Main clock for PLLB. For further information, see Section 30.16.2 ”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, PLLBCK. 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 has been configured, the corresponding Programmable clock must be enabled and the user is constrained to wait for PMC_SR.PCKRDYx to be set. This can be done either by polling PMC_SR.PCKRDYx or by waiting for the interrupt line to be raised if the associated interrupt source (PMC_IER.PCKRDYx) has been enabled. 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.

30.16 Clock Switching Details

30.16.1 Master Clock Switching Timings

Table 30-1 and Table 30-2 give the worst case timings required for the Master clock to switch from one selected clock to another one. This is in the event that the prescaler is de-activated. When the prescaler is activated, an additional time of 64 clock cycles of the newly selected clock has to be added.

Table 30-1. Clock Switching Timings (Worst Case)

From To	Main Clock	SLCK	PLL Clock
Main Clock	–	4 x SLCK + 2.5 x Main Clock	3 x PLL Clock + 4 x SLCK + 1 x Main Clock
SLCK	0.5 x Main Clock + 4.5 x SLCK	–	3 x PLL Clock + 5 x SLCK
PLL Clock	0.5 x Main Clock + 4 x SLCK + PLLCOUNT x SLCK + 2.5 x PLLx Clock	2.5 x PLL Clock + 5 x SLCK + PLLCOUNT x SLCK	2.5 x PLL Clock + 4 x SLCK + PLLCOUNT x SLCK

Notes: 1. PLL designates either the PLLA or the PLLB Clock.
2. PLLCOUNT designates either PLLACOUNT or PLLBCOUNT.

Table 30-2. Clock Switching Timings between Two PLLs (Worst Case)

From To	PLLA Clock	PLLB Clock
PLLA Clock	2.5 x PLLA Clock + 4 x SLCK + PLLACOUNT x SLCK	3 x PLLA Clock + 4 x SLCK + 1.5 x PLLA Clock
PLLB Clock	3 x PLLB Clock + 4 x SLCK + 1.5 x PLLB Clock	2.5 x PLLB Clock + 4 x SLCK+ PLLBCOUNT x SLCK

30.16.2 Clock Switching Waveforms

Figure 30-5. Switch Master Clock from Slow Clock to PLLx Clock

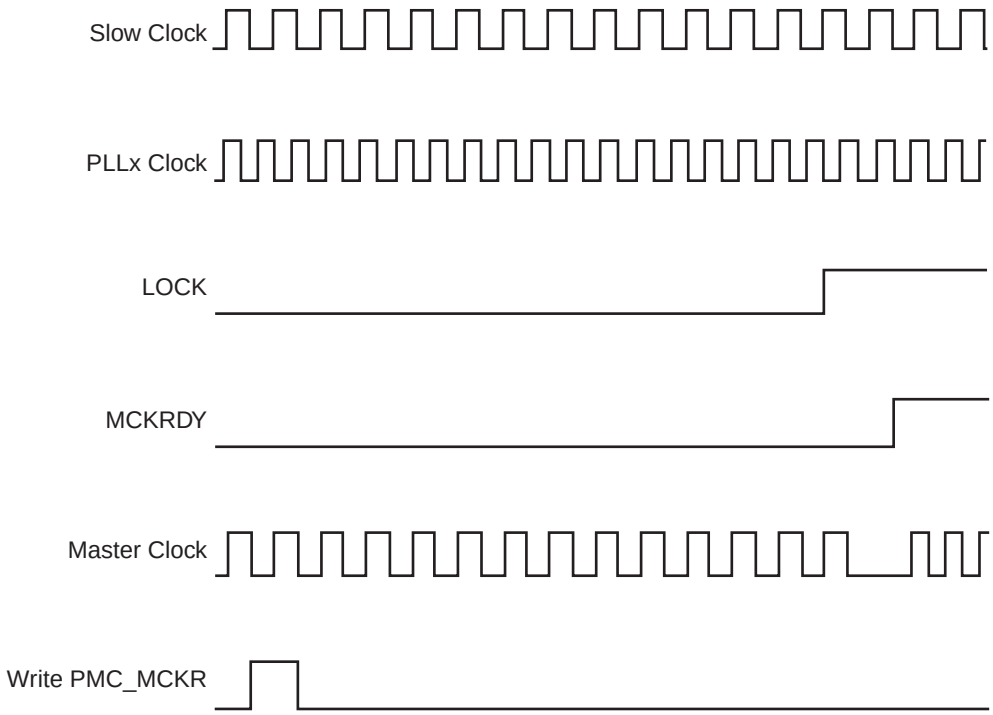


Figure 30-6. Switch Master Clock from Main Clock to Slow Clock

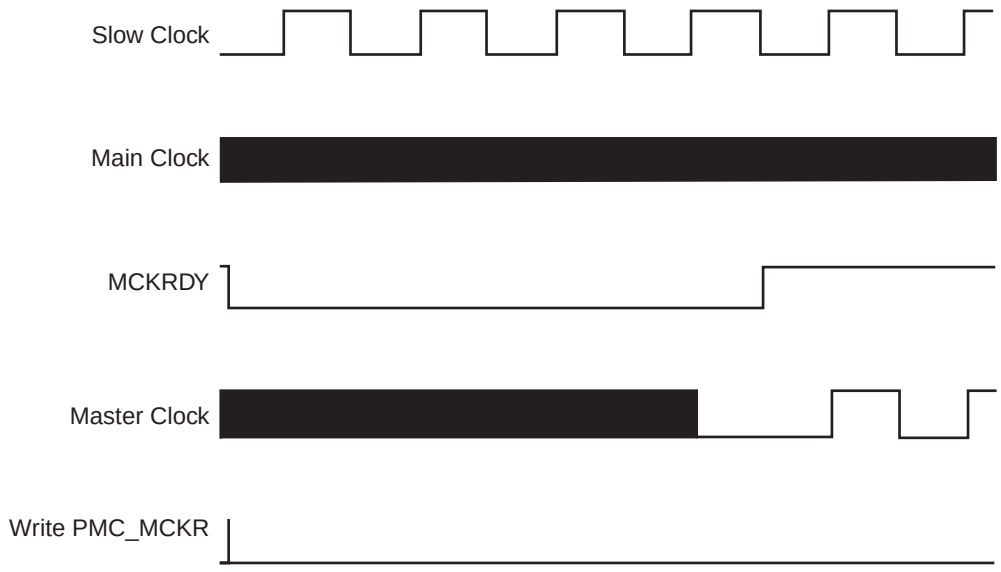


Figure 30-7. Change PLLx Programming

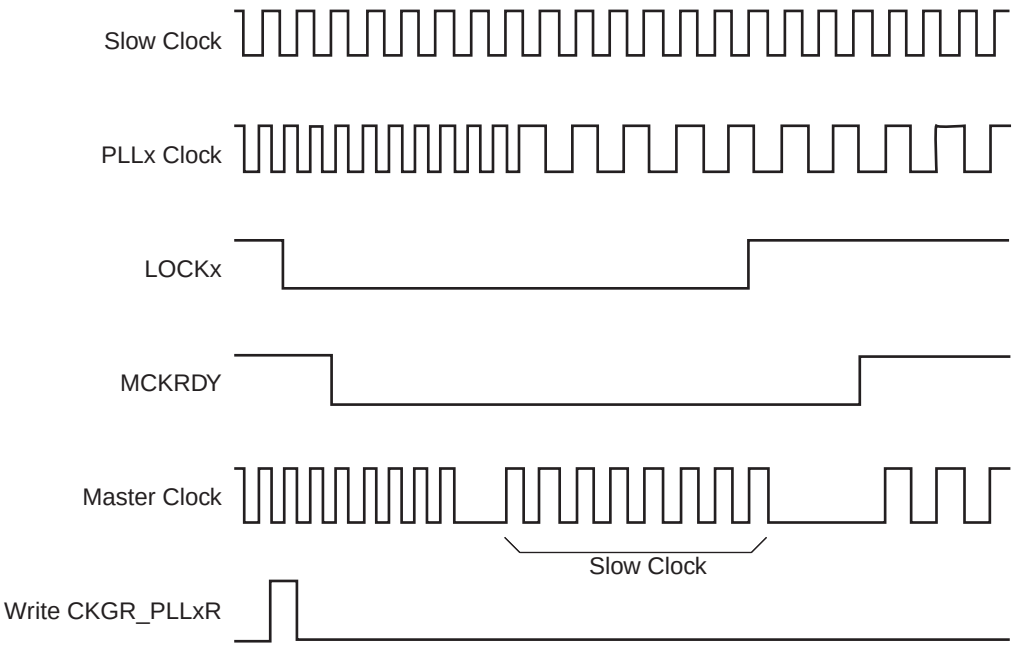
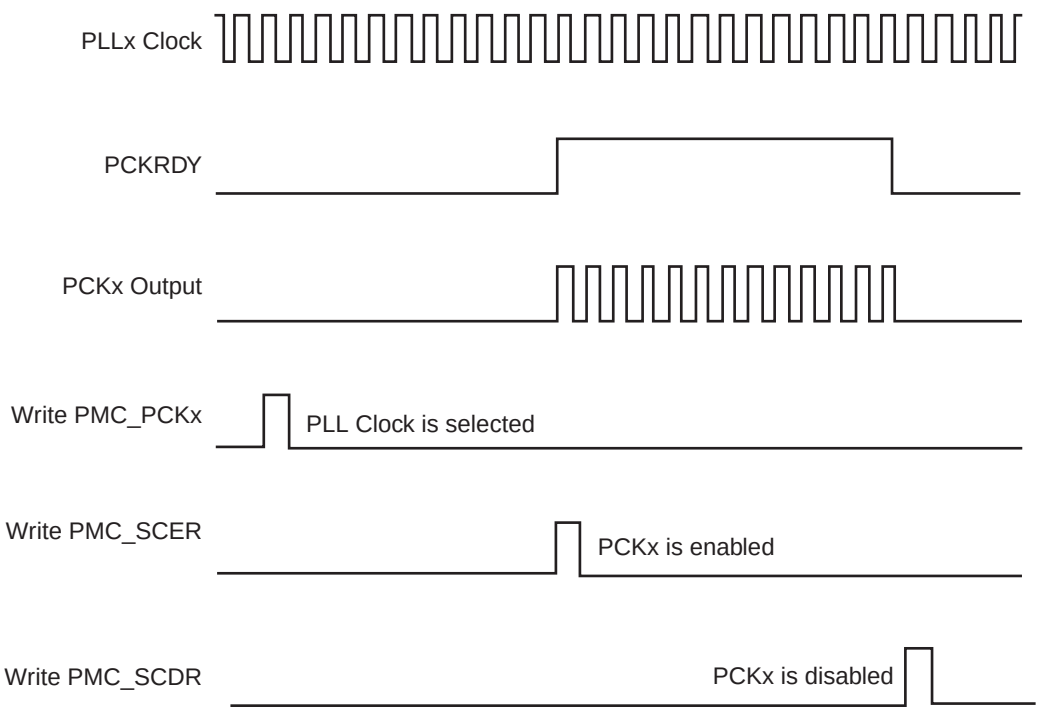


Figure 30-8. Programmable Clock Output Programming



30.17 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 Clock Generator PLLB Register"
- "PMC Master Clock Register"
- "PMC Programmable Clock Register"
- "PMC Fast Startup Mode Register"
- "PMC Fast Startup Polarity Register"
- "PMC Coprocessor Fast Startup Mode Register"
- "PMC Peripheral Clock Enable Register 1"
- "PMC Peripheral Clock Disable Register 1"
- "PMC Oscillator Calibration Register"

30.18 Power Management Controller (PMC) User Interface

Table 30-3. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	System Clock Enable Register	PMC_SCER	Write-only	–
0x0004	System Clock Disable Register	PMC_SCDR	Write-only	–
0x0008	System Clock Status Register	PMC_SCSR	Read-only	0x0000_0001
0x000C	Reserved	–	–	–
0x0010	Peripheral Clock Enable Register 0	PMC_PCER0	Write-only	–
0x0014	Peripheral Clock Disable Register 0	PMC_PCDR0	Write-only	–
0x0018	Peripheral Clock Status Register 0	PMC_PCSR0	Read-only	0x0000_0000
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	PLLB Register	CKGR_PLLBR	Read/Write	0x0000_3F00
0x0030	Master Clock Register	PMC_MCKR	Read/Write	0x0000_0001
0x0034 - 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	Coprocessor Fast Startup Mode Register	PMC_CPFSMR	Read/Write	0x0000_0000
0x0080 - 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	–	–	–
0x0110	Oscillator Calibration Register	PMC_OCR	Read/Write	0x0040_4040
0x0114 - 0x0120	Reserved	–	–	–
0x0134 - 0x0144	Reserved	–	–	–

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

30.18.1 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
CPKEY				–	–	CPBMCK	CPCK
15	14	13	12	11	10	9	8
–	–	–	–	–	PCK2	PCK1	PCK0
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” .

- **PCKx: Programmable Clock x Output Enable**

0 = No effect.

1 = Enables the corresponding Programmable Clock output.

- **CPCK: Coprocessor (Second Processor) Clocks Enable**

0 = No effect.

1 = Enables the corresponding Coprocessor Clocks (CPHCLK, CPSYSTICK) if CPKEY = 0xA.

- **CPBMCK: Coprocessor Bus Master Clocks Enable**

0 = No effect.

1 = Enables the corresponding Coprocessor Bus Master Clock (CPBMCK, CPFCLK) if CPKEY = 0xA.

Note: Enabling CPBMCK must be performed prior to or at the same time as programming CPCK to 1 in PMC_SCER or prior communication with one of the peripherals of the coprocessor system bus.

- **CPKEY: Coprocessor Clocks Enable Key**

Value	Name	Description
0xA	PASSWD	This field must be written to 0xA in order to validate CPCK field.

30.18.2 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
CPKEY				–	–	CPBMCK	CPCK
15	14	13	12	11	10	9	8
–	–	–	–	–	PCK2	PCK1	PCK0
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” .

- **PCKx: Programmable Clock x Output Disable**

0 = No effect.

1 = Disables the corresponding Programmable Clock output.

- **CPCK: Coprocessor Clocks Disable**

0 = No effect.

1 = Enables the corresponding Coprocessor Clocks (CPHCLK, CPFCLK, CPSYSTICK) if CPKEY = 0xA.

- **CPBMCK: Coprocessor Bus Master Clocks Disable**

0 = No effect.

1 = Disables the corresponding Coprocessor Bus Master Clock (CPBMCK, CPFCLK) if CPKEY = 0xA.

Note: Disabling CPBMCK must not be performed if CPCK is 1 in PMC_SCSR.

- **CPKEY: Coprocessor Clocks Disable Key**

Value	Name	Description
0xA	PASSWD	This field must be written to 0xA in order to validate CPCK field.

30.18.3 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
–	–	–	–	–	–	CPBMCK	CPCK
15	14	13	12	11	10	9	8
–	–	–	–	–	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

- **PCKx: Programmable Clock x Output Status**

0 = The corresponding Programmable Clock output is disabled.

1 = The corresponding Programmable Clock output is enabled.

- **CPCK: Coprocessor (Second Processor) Clocks Status**

0 = Coprocessor Clocks (CPHCLK, CPSYSTICK) are disabled (value after reset).

1 = Coprocessor Clocks (CPHCLK, CPSYSTICK) are enabled.

- **CPBMCK: Coprocessor Bus Master Clock Status**

0 = Coprocessor Clocks (CPBMCK, CPFCLK) are disabled (value after reset).

1 = Coprocessor Clocks (CPBMCK, CPFCLK) are enabled.

30.18.4 PMC Peripheral Clock Enable Register 0

Name: PMC_PCER0

Address: 0x400E0410

Access: Write-only

31	30	29	28	27	26	25	24
PID31	–	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
–	–	–	–	–	–	–	–

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 30.18.23 ”PMC Peripheral Clock Enable Register 1”).

Note: Programming the control bits of the Peripheral ID that are not implemented has no effect on the behavior of the PMC.

30.18.5 PMC Peripheral Clock Disable Register 0

Name: PMC_PCDR0

Address: 0x400E0414

Access: Write-only

31	30	29	28	27	26	25	24
PID31	–	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
–	–	–	–	–	–	–	–

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 30.18.24 ”PMC Peripheral Clock Disable Register 1”).

30.18.6 PMC Peripheral Clock Status Register 0

Name: PMC_PCSR0

Address: 0x400E0418

Access: Read-only

31	30	29	28	27	26	25	24
PID31	–	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
–	–	–	–	–	–	–	–

- **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 30.18.25 ”PMC Peripheral Clock Status Register 1”).

30.18.7 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 startup 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 startup 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 Startup Time**

Specifies the number of Slow clock cycles multiplied by 8 for the crystal oscillator startup 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.

- **XT32KFME: 32.768 kHz Crystal Oscillator Frequency Monitoring Enable**

0 = The 32.768 kHz Crystal oscillator frequency monitoring is disabled.

1 = The 32.768 kHz Crystal oscillator frequency monitoring is enabled.

30.18.8 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 x 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.

30.18.9 PMC Clock Generator PLLA Register

Name: CKGR_PLLAR

Address: 0x400E0428

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	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
PLLAEN							

Possible limitations on PLLA input frequencies and multiplier factors should be checked before using the PMC.

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **PLLAEN: PLLA Control**

0 = PLLA is disabled.

1 = PLLA is enabled.

2 - 255 = forbidden.

- **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 disabled (PLLA also disabled if DIVA = 0).

200 up to 254 = The PLLA Clock frequency is the PLLA input frequency multiplied by MULA + 1.

Unlisted values are forbidden.

To change the PLLA frequency, please read Section 29.6.1 ”Divider and Phase Lock Loop Programming”.

30.18.10 PMC Clock Generator PLLB Register

Name: CKGR_PLLBR

Address: 0x400E042C

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	SRCB	–	–	MULB		

23	22	21	20	19	18	17	16
MULB							

15	14	13	12	11	10	9	8
–	–	PLLBCOUNT					

7	6	5	4	3	2	1	0
DIVB							

Possible limitations on PLLB input frequencies and multiplier factors should be checked before using the PMC.

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **DIVB: PLLB Front-End Divider**

Value	Name	Description
0	0	PLLB is disabled.
1	BYPASS	Divider is bypassed.
2 - 255	-	Divider output is the selected clock divided by DIVB.

- **PLLBCOUNT: PLLB Counter**

Specifies the number of Slow clock cycles before the LOCKB bit is set in PMC_SR after CKGR_PLLBR is written.

- **MULB: PLLB Multiplier**

0 = The PLLB is disabled (PLLB also disabled if DIVB = 0).

1 up to 62 = The PLLB Clock frequency is the PLLB input frequency multiplied by MULB + 1.

Unlisted values are forbidden.

- **SRCB: Source for PLLB**

Value	Name	Description
0	MAINCK_IN_PLLB	The PLLB input clock is Main Clock.
1	PLLA_IN_PLLB	The PLLB input clock is PLLA output.

30.18.11 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
CPPRES				–	CPCSS		

15	14	13	12	11	10	9	8
–	–	PLLBDIV2	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
3	PLLB_CLK	PLLB 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

- **PLLBDIV2: PLLB Divisor by 2**

PLLBDIV2	PLLB Clock Division
0	PLLB clock frequency is divided by 1
1	PLLB clock frequency is divided by 2

- **CPCSS: Coprocessor Master Clock Source Selection**

Value	Name	Description
0	SLOW_CLK	Slow Clock is selected
1	MAIN_CLK	Main Clock is selected
2	PLLA_CLK	PLLA Clock is selected
3	PLLB_CLK	PLLB Clock is selected
4	MCK	Master Clock is selected

- **CPPRES: Coprocessor Programmable Clock Prescaler**

0 - 15 = The selected clock is divided by CPPRES+1.

30.18.12 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
3	PLLB_CLK	PLLB Clock is selected
4	MCK	Master Clock is selected

- **PRES: Programmable Clock Prescaler**

Value	Name	Description
0	CLK_1	Selected clock
1	CLK_2	Selected clock divided by 2
2	CLK_4	Selected clock divided by 4
3	CLK_8	Selected clock divided by 8
4	CLK_16	Selected clock divided by 16
5	CLK_32	Selected clock divided by 32
6	CLK_64	Selected clock divided by 64

30.18.13 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	LOCKB	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**
- **LOCKB: PLLB 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: 32.768 kHz Crystal Oscillator Error Interrupt Enable**

30.18.14 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	LOCKB	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**
- **LOCKB: PLLB 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: 32.768 kHz Crystal Oscillator Error Interrupt Disable**

30.18.15 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	LOCKB	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.

- **LOCKB: PLLB Lock Status**

0 = PLLB is not locked.

1 = PLLB 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 = 32.768 kHz 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: 32.768 kHz Crystal Oscillator Error**

0 = The frequency of the 32.768 kHz crystal oscillator is correct (32.768 kHz +/- 1%) or the monitoring is disabled.

1 = The frequency of the 32.768 kHz crystal oscillator is incorrect or has been incorrect for an elapsed period of time since the monitoring has been enabled.

30.18.16 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	LOCKB	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**
- **LOCKB: PLLB 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: 32.768 kHz Crystal Oscillator Error Interrupt Mask**

30.18.17 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	–	–	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.

- **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

30.18.18 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.

30.18.19 PMC Coprocessor Fast Startup Mode Register

Name: PMC_CPFSMR

Address: 0x400E047C

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	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.

30.18.20 PMC Fault Output Clear Register

Name: PMC_FOCR

Address: 0x400E0478

Access: Write-only

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

- **FOCLR: Fault Output Clear**

Clears the clock failure detector fault output.

30.18.21 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 30.17 ”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.

30.18.22 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.

30.18.23 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
–	–	–	–	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.

30.18.24 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
–	–	–	–	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”.

30.18.25 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
–	–	–	–	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”.

30.18.26 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.

31. Chip Identifier (CHIPID)

31.1 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.

31.2 Embedded Characteristics

- Chip ID Registers
  - Identification of the Device Revision, Sizes of the Embedded Memories, Set of Peripherals, Embedded Processor.

Table 31-1. SAM4CP Chip IDs Registers

Chip Name	CHIPID_CIDR	CHIPID_EXID
SAM4CP16C	0xA64C_0CE0	0x4

31.3 Chip Identifier (CHIPID) User Interface

Table 31-2. Register Mapping

Offset	Register	Name	Access	Reset
0x0	Chip ID Register	CHIPID_CIDR	Read-only	–
0x4	Chip ID Extension Register	CHIPID_EXID	Read-only	–

31.3.1 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
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
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
12	128K	128 Kbytes
13	256K	256 Kbytes
14	96K	96 Kbytes
15	512K	512 Kbytes

- **ARCH: Architecture Identifier**

Value	Device
0x64	SAM4CP16C

- **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.

31.3.2 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.

Value	Name	Description
0x4	SAM4CP	SAM4C embedding GPLC peripheral

32. Parallel Input/Output Controller (PIO)

32.1 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 the following:

- 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 I/O line pull-up and pull-down.
- Input visibility and output control.

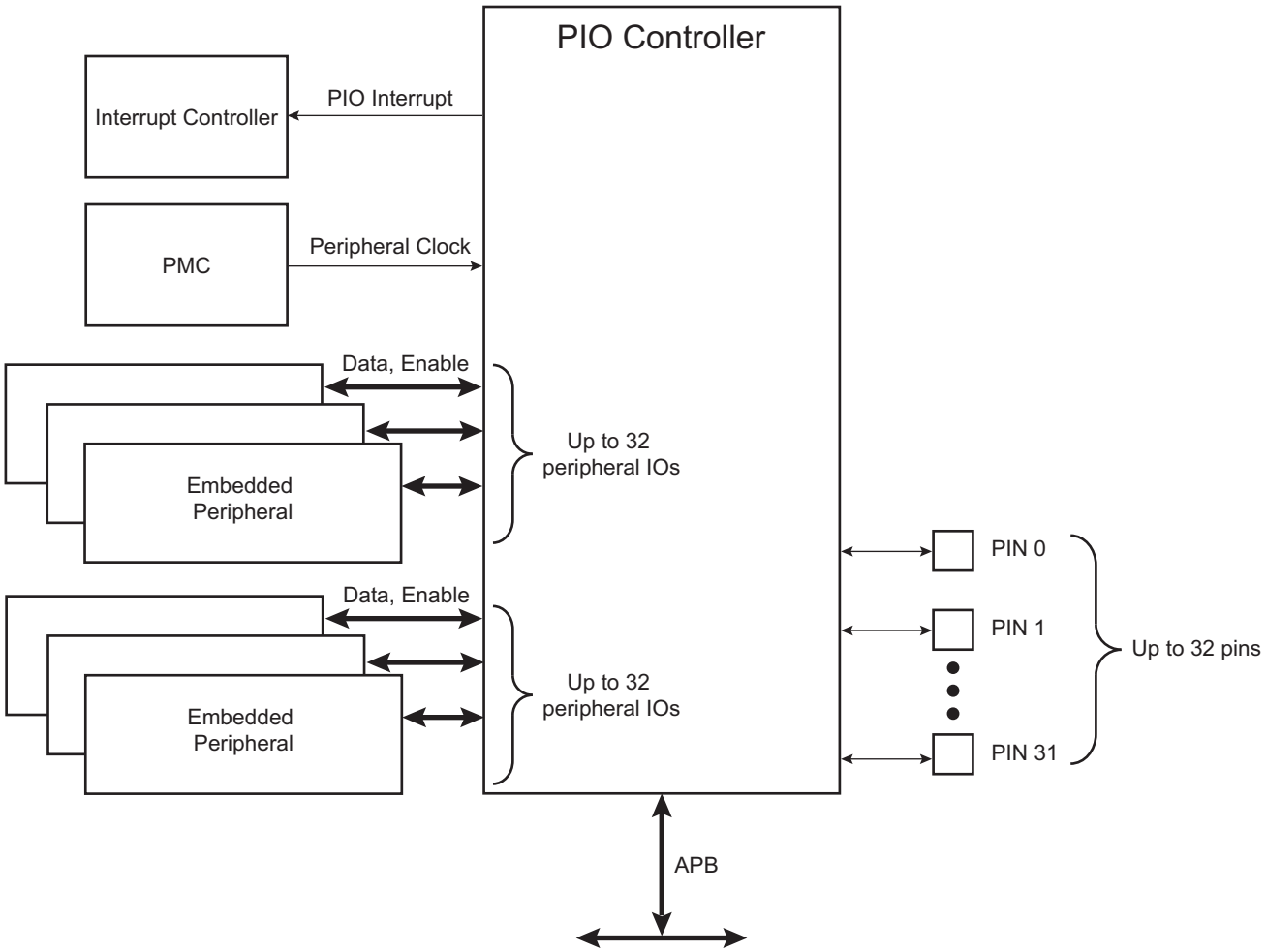
The PIO Controller also features a synchronous output providing up to 32 bits of data output in a single write operation.

32.2 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.
- 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 Drive.

32.3 Block Diagram

Figure 32-1. Block Diagram



32.4 Product Dependencies

32.4.1 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.

32.4.2 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.

32.4.3 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.

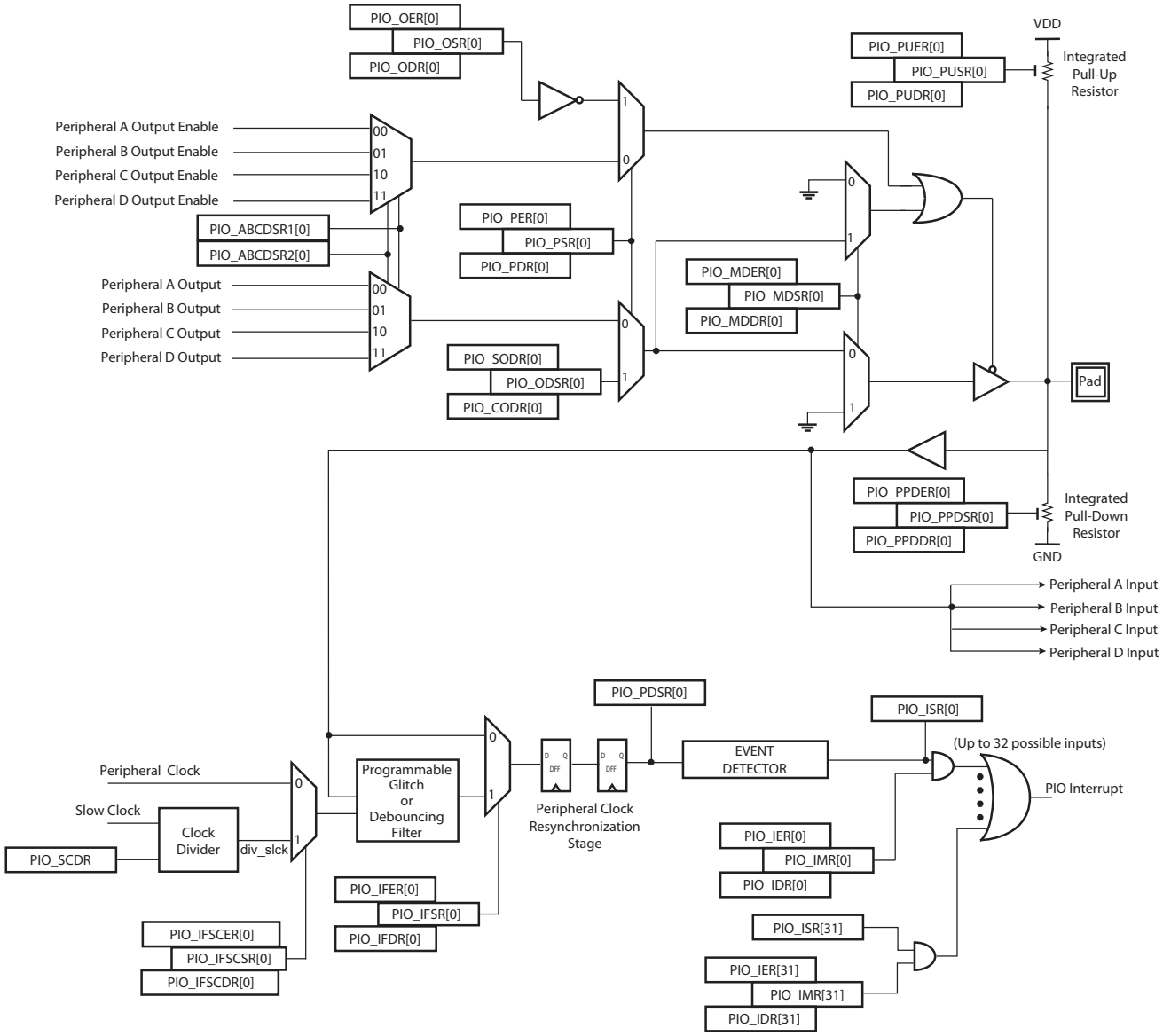
Table 32-1. Peripheral IDs

Instance	ID
PIOA	11
PIOB	12
PIOC	37

32.5 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 32-2. In this description each signal shown represents one of up to 32 possible indexes.

Figure 32-2. I/O Line Control Logic



32.5.1 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.

32.5.2 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 Peripheral 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.

32.5.3 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 (refer to Figure 32-2).

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.

32.5.4 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.

32.5.5 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.

32.5.6 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.

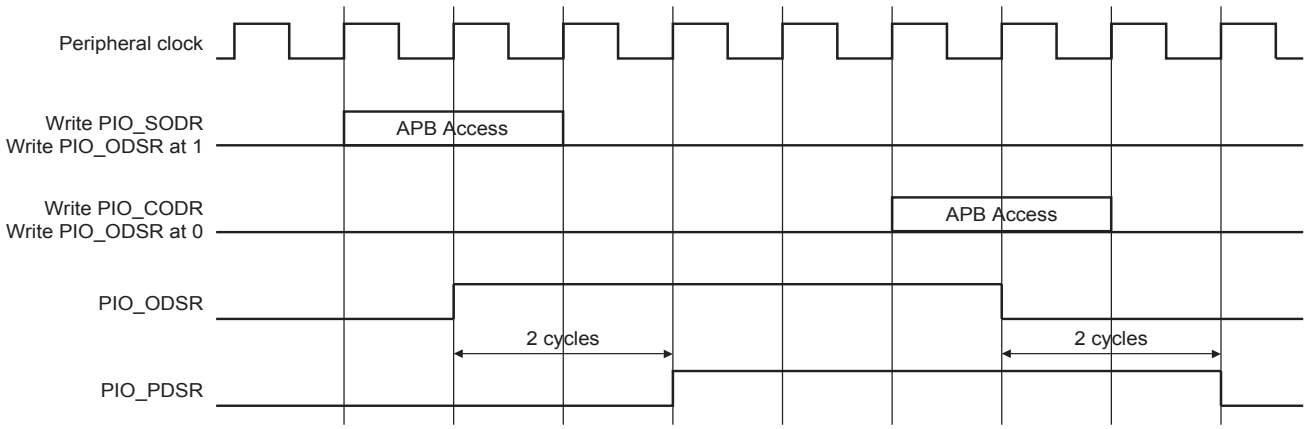
The multi-drive feature is controlled by the Multi-driver Enable register (PIO_MDER) and the Multi-driver Disable register (PIO_MDDR). The multi-drive can be selected whether the I/O line is controlled by the PIO Controller or assigned to a peripheral function. The Multi-driver Status register (PIO_MDSR) 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.

32.5.7 Output Line Timings

Figure 32-3 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 32-3 also shows when the feedback in the Pin Data Status register (PIO_PDSR) is available.

Figure 32-3. Output Line Timings



32.5.8 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.

32.5.9 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 32-4 and Figure 32-5.

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 32-4. Input Glitch Filter Timing

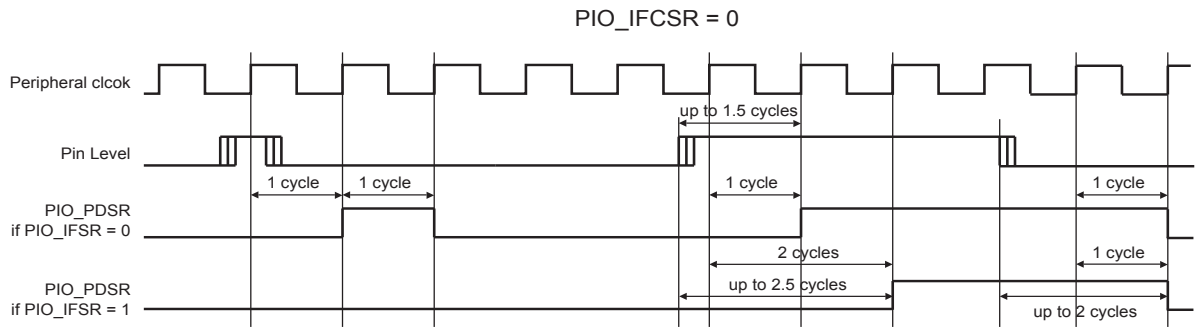
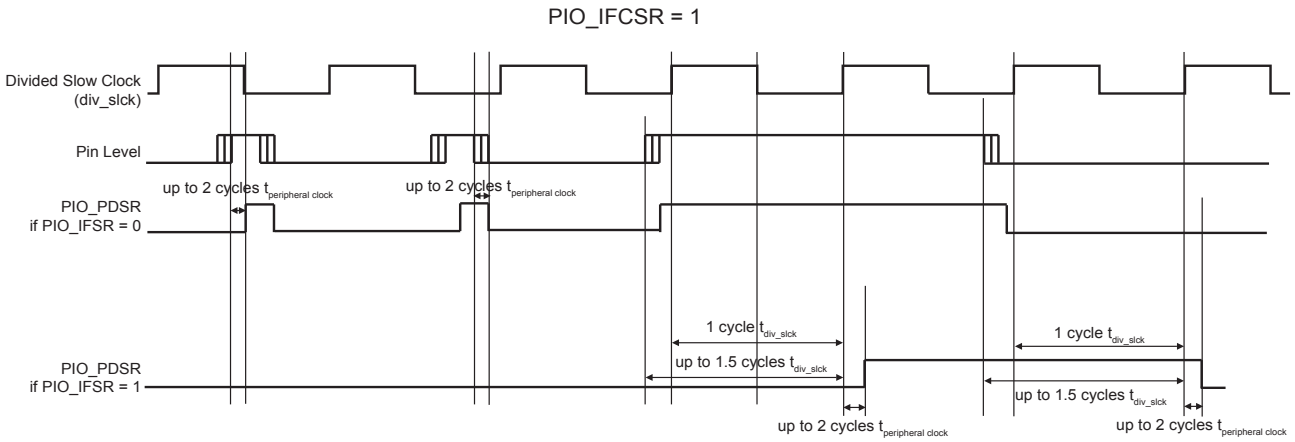


Figure 32-5. Input Debouncing Filter Timing



32.5.10 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.

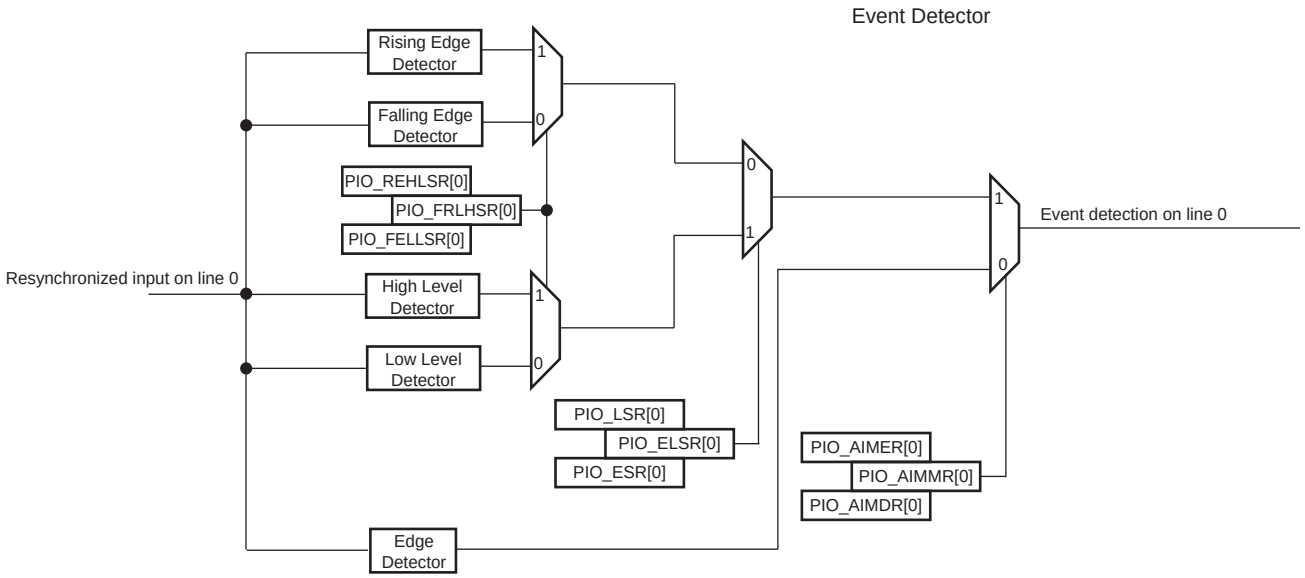
In order to select an additional interrupt mode:

- 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 32-6. Event Detector on Input Lines (Figure Represents Line 0)



Example of interrupt generation on following lines:

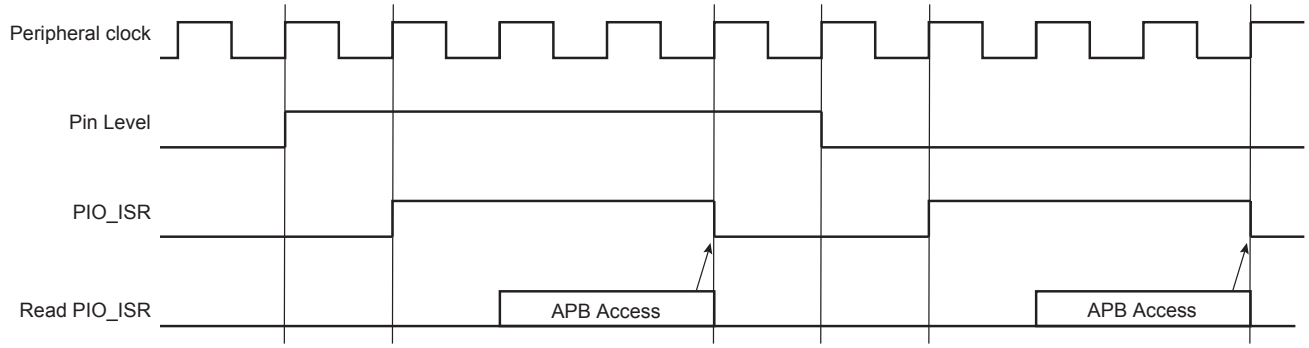
- Rising edge on PIO line 0
- Falling edge on PIO line 1
- Rising edge on PIO line 2
- Low-level on PIO line 3
- High-level on PIO line 4
- High-level on PIO line 5
- Falling edge on PIO line 6
- Rising edge on PIO line 7
- Any edge on the other lines

Table 32-2 provides the required configuration for this example.

Table 32-2. 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 32-7. Input Change Interrupt Timings When No Additional Interrupt Modes



32.5.11 Programmable I/O Drive

It is possible to configure the I/O drive for pads PA0 to PA4 and PA9 to PA31. Refer to the section “Electrical Characteristics”.

32.5.12 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.

32.5.13 I/O Lines Programming Example

The programming example shown in Table 32-3 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 32-3. 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

Table 32-3. Programming Example (Continued)

PIO_CODR	0x0FFF_FFFF
PIO_IER	0x0F00_0F00
PIO_IDR	0xF0FF_F0FF
PIO_MDER	0x0000_000F
PIO_MDDR	0xFFFF_FFF0
PIO_PUDR	0xFFF0_00F0
PIO_PUER	0x000F_FF0F
PIO_PPDDR	0xFF0F_FFFF
PIO_PPDER	0x00F0_0000
PIO_ABCDSR1	0xF0F0_0000
PIO_ABCDSR2	0xFF00_0000
PIO_OWER	0x0000_000F
PIO_OWDR	0x0FFF_FFF0

32.5.14 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"

32.6 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 32-4. Register Mapping[1]

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	[2]
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[3] Read/Write	–
0x003C	Pin Data Status Register	PIO_PDSR	Read-only	[4]
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[5]	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	[2]
0x006C	Reserved	–	–	–

Table 32-4. Register Mapping[1] (Continued)

Offset	Register	Name	Access	Reset
0x0070	Peripheral ABCD Select Register 1	PIO_ABCDSR1	Read/Write	0x00000000
0x0074	Peripheral ABCD 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	[2]
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	Reserved	–	–	–
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	Reserved	–	–	–
0x0114	Reserved	–	–	–

Table 32-4. Register Mapping[1] (Continued)

Offset	Register	Name	Access	Reset
0x0118	I/O Drive Register	PIO_DRIVER	Read/Write	0x00000000
0x011C	Reserved	–	–	–
0x0120 - 0x014C	Reserved	–	–	–

Notes:
1. If an offset is not listed in the table it must be considered as reserved.
2. Reset value depends on the product implementation.
3. PIO_ODSR is Read-only or Read/Write depending on PIO_OWSR I/O lines.
4. 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.
5. PIO_ISR is reset at 0x0. However, the first read of the register may read a different value as input changes may have occurred.

32.6.1 PIO Enable Register

Name: PIO_PER

Address: 0x400E0E00 (PIOA), 0x400E1000 (PIOB), 0x4800C000 (PIOC)

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).

32.6.2 PIO Disable Register

Name: PIO_PDR

Address: 0x400E0E04 (PIOA), 0x400E1004 (PIOB), 0x4800C004 (PIOC)

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).

32.6.3 PIO Status Register

Name: PIO_PSR

Address: 0x400E0E08 (PIOA), 0x400E1008 (PIOB), 0x4800C008 (PIOC)

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).

32.6.4 PIO Output Enable Register

Name: PIO_OER

Address: 0x400E0E10 (PIOA), 0x400E1010 (PIOB), 0x4800C010 (PIOC)

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.

32.6.5 PIO Output Disable Register

Name: PIO_ODR

Address: 0x400E0E14 (PIOA), 0x400E1014 (PIOB), 0x4800C014 (PIOC)

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.

32.6.6 PIO Output Status Register

Name: PIO_OSR

Address: 0x400E0E18 (PIOA), 0x400E1018 (PIOB), 0x4800C018 (PIOC)

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.

32.6.7 PIO Input Filter Enable Register

Name: PIO_IFER

Address: 0x400E0E20 (PIOA), 0x400E1020 (PIOB), 0x4800C020 (PIOC)

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.

32.6.8 PIO Input Filter Disable Register

Name: PIO_IFDR

Address: 0x400E0E24 (PIOA), 0x400E1024 (PIOB), 0x4800C024 (PIOC)

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.

32.6.9 PIO Input Filter Status Register

Name: PIO_IFSR

Address: 0x400E0E28 (PIOA), 0x400E1028 (PIOB), 0x4800C028 (PIOC)

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.

32.6.10 PIO Set Output Data Register

Name: PIO_SODR

Address: 0x400E0E30 (PIOA), 0x400E1030 (PIOB), 0x4800C030 (PIOC)

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.

32.6.11 PIO Clear Output Data Register

Name: PIO_CODR

Address: 0x400E0E34 (PIOA), 0x400E1034 (PIOB), 0x4800C034 (PIOC)

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.

32.6.12 PIO Output Data Status Register

Name: PIO_ODSR

Address: 0x400E0E38 (PIOA), 0x400E1038 (PIOB), 0x4800C038 (PIOC)

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.

32.6.13 PIO Pin Data Status Register

Name: PIO_PDSR

Address: 0x400E0E3C (PIOA), 0x400E103C (PIOB), 0x4800C03C (PIOC)

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.

32.6.14 PIO Interrupt Enable Register

Name: PIO_IER

Address: 0x400E0E40 (PIOA), 0x400E1040 (PIOB), 0x4800C040 (PIOC)

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.

32.6.15 PIO Interrupt Disable Register

Name: PIO_IDR

Address: 0x400E0E44 (PIOA), 0x400E1044 (PIOB), 0x4800C044 (PIOC)

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.

32.6.16 PIO Interrupt Mask Register

Name: PIO_IMR

Address: 0x400E0E48 (PIOA), 0x400E1048 (PIOB), 0x4800C048 (PIOC)

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.

32.6.17 PIO Interrupt Status Register

Name: PIO_ISR

Address: 0x400E0E4C (PIOA), 0x400E104C (PIOB), 0x4800C04C (PIOC)

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.

32.6.18 PIO Multi-driver Enable Register

Name: PIO_MDER

Address: 0x400E0E50 (PIOA), 0x400E1050 (PIOB), 0x4800C050 (PIOC)

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.

32.6.19 PIO Multi-driver Disable Register

Name: PIO_MDDR

Address: 0x400E0E54 (PIOA), 0x400E1054 (PIOB), 0x4800C054 (PIOC)

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.

32.6.20 PIO Multi-driver Status Register

Name: PIO_MDSR

Address: 0x400E0E58 (PIOA), 0x400E1058 (PIOB), 0x4800C058 (PIOC)

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.

32.6.21 PIO Pull-Up Disable Register

Name: PIO_PUDR

Address: 0x400E0E60 (PIOA), 0x400E1060 (PIOB), 0x4800C060 (PIOC)

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.

32.6.22 PIO Pull-Up Enable Register

Name: PIO_PUER

Address: 0x400E0E64 (PIOA), 0x400E1064 (PIOB), 0x4800C064 (PIOC)

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.

32.6.23 PIO Pull-Up Status Register

Name: PIO_PUSR

Address: 0x400E0E68 (PIOA), 0x400E1068 (PIOB), 0x4800C068 (PIOC)

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.

32.6.24 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.

32.6.25 PIO Peripheral ABCD Select Register 2

Name: PIO_ABCDSR2

Access: Read/Write

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24

23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16

15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8

7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in 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.

32.6.26 PIO Input Filter Slow Clock Disable Register

Name: PIO_IFSCDR

Address: 0x400E0E80 (PIOA), 0x400E1080 (PIOB), 0x4800C080 (PIOC)

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 < $t_{peripheral\ clock}/2$.

32.6.27 PIO Input Filter Slow Clock Enable Register

Name: PIO_IFSCER

Address: 0x400E0E84 (PIOA), 0x400E1084 (PIOB), 0x4800C084 (PIOC)

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 < $t_{div\_slck}/2$.

32.6.28 PIO Input Filter Slow Clock Status Register

Name: PIO_IFSCSR

Address: 0x400E0E88 (PIOA), 0x400E1088 (PIOB), 0x4800C088 (PIOC)

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 < $t_{div\_slck}/2$.

32.6.29 PIO Slow Clock Divider Debouncing Register

Name: PIO_SCDR

Address: 0x400E0E8C (PIOA), 0x400E108C (PIOB), 0x4800C08C (PIOC)

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**

$t_{div\_slck} = ((DIV + 1) \times 2) \times t_{slck}$

32.6.30 PIO Pad Pull-Down Disable Register

Name: PIO_PPDDR

Address: 0x400E0E90 (PIOA), 0x400E1090 (PIOB), 0x4800C090 (PIOC)

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.

32.6.31 PIO Pad Pull-Down Enable Register

Name: PIO_PPDER

Address: 0x400E0E94 (PIOA), 0x400E1094 (PIOB), 0x4800C094 (PIOC)

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.

32.6.32 PIO Pad Pull-Down Status Register

Name: PIO_PPDSR

Address: 0x400E0E98 (PIOA), 0x400E1098 (PIOB), 0x4800C098 (PIOC)

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.

32.6.33 PIO Output Write Enable Register

Name: PIO_OWER

Address: 0x400E0EA0 (PIOA), 0x400E10A0 (PIOB), 0x4800C0A0 (PIOC)

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.

32.6.34 PIO Output Write Disable Register

Name: PIO_OWDR

Address: 0x400E0EA4 (PIOA), 0x400E10A4 (PIOB), 0x4800C0A4 (PIOC)

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.

32.6.35 PIO Output Write Status Register

Name: PIO_OWSR

Address: 0x400E0EA8 (PIOA), 0x400E10A8 (PIOB), 0x4800C0A8 (PIOC)

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.

32.6.36 PIO Additional Interrupt Modes Enable Register

Name: PIO_AIMER

Address: 0x400E0EB0 (PIOA), 0x400E10B0 (PIOB), 0x4800C0B0 (PIOC)

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.

32.6.37 PIO Additional Interrupt Modes Disable Register

Name: PIO_AIMDR

Address: 0x400E0EB4 (PIOA), 0x400E10B4 (PIOB), 0x4800C0B4 (PIOC)

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).

32.6.38 PIO Additional Interrupt Modes Mask Register

Name: PIO_AIMMR

Address: 0x400E0EB8 (PIOA), 0x400E10B8 (PIOB), 0x4800C0B8 (PIOC)

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.

32.6.39 PIO Edge Select Register

Name: PIO_ESR

Address: 0x400E0EC0 (PIOA), 0x400E10C0 (PIOB), 0x4800C0C0 (PIOC)

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.

32.6.40 PIO Level Select Register

Name: PIO_LSR

Address: 0x400E0EC4 (PIOA), 0x400E10C4 (PIOB), 0x4800C0C4 (PIOC)

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.

32.6.41 PIO Edge/Level Status Register

Name: PIO_ELSR

Address: 0x400E0EC8 (PIOA), 0x400E10C8 (PIOB), 0x4800C0C8 (PIOC)

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.

32.6.42 PIO Falling Edge/Low-Level Select Register

Name: PIO_FELLSR

Address: 0x400E0ED0 (PIOA), 0x400E10D0 (PIOB), 0x4800C0D0 (PIOC)

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.

32.6.43 PIO Rising Edge/High-Level Select Register

Name: PIO_REHLSR

Address: 0x400E0ED4 (PIOA), 0x400E10D4 (PIOB), 0x4800C0D4 (PIOC)

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.

32.6.44 PIO Fall/Rise - Low/High Status Register

Name: PIO_FRLHSR

Address: 0x400E0ED8 (PIOA), 0x400E10D8 (PIOB), 0x4800C0D8 (PIOC)

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).

32.6.45 PIO Write Protection Mode Register

Name: PIO_WPMR

Address: 0x400E0EE4 (PIOA), 0x400E10E4 (PIOB), 0x4800C0E4 (PIOC)

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).

Refer to Section 32.5.14 ”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.

32.6.46 PIO Write Protection Status Register

Name: PIO_WPSR

Address: 0x400E0EE8 (PIOA), 0x400E10E8 (PIOB), 0x4800C0E8 (PIOC)

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.

32.6.47 PIO Schmitt Trigger Register

Name: PIO_SCHMITT

Address: 0x400E0F00 (PIOA), 0x400E1100 (PIOB), 0x4800C100 (PIOC)

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.

32.6.48 PIO I/O Drive Register

Name: PIO_DRIVER

Address: 0x400E0F18 (PIOA), 0x400E1118 (PIOB), 0x4800C118 (PIOC)

Access: Read/Write

31	30	29	28	27	26	25	24
LINE31	LINE30	LINE29	LINE28	LINE27	LINE26	LINE25	LINE24
23	22	21	20	19	18	17	16
LINE23	LINE22	LINE21	LINE20	LINE19	LINE18	LINE17	LINE16
15	14	13	12	11	10	9	8
LINE15	LINE14	LINE13	LINE12	LINE11	LINE10	LINE9	LINE8
7	6	5	4	3	2	1	0
LINE7	LINE6	LINE5	LINE4	LINE3	LINE2	LINE1	LINE0

- **LINEx [x=0..31]: Drive of PIO Line x**

Value	Name	Description
0	LOW_DRIVE	Lowest drive
1	HIGH_DRIVE	Highest drive

33. Serial Peripheral Interface (SPI)

33.1 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:

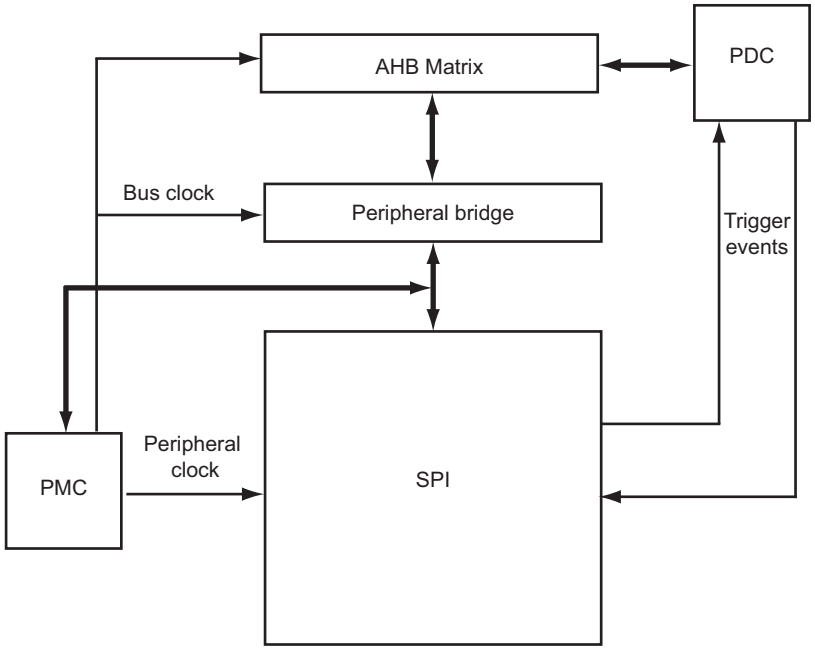
- Master Out Slave In (MOSI): This data line supplies the output data from the master shifted into the input(s) of the slave(s).
- Master In Slave Out (MISO): This data line supplies the output data from a slave to the input of the master. There may be no more than one slave transmitting data during 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.

33.2 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
- Register Write Protection

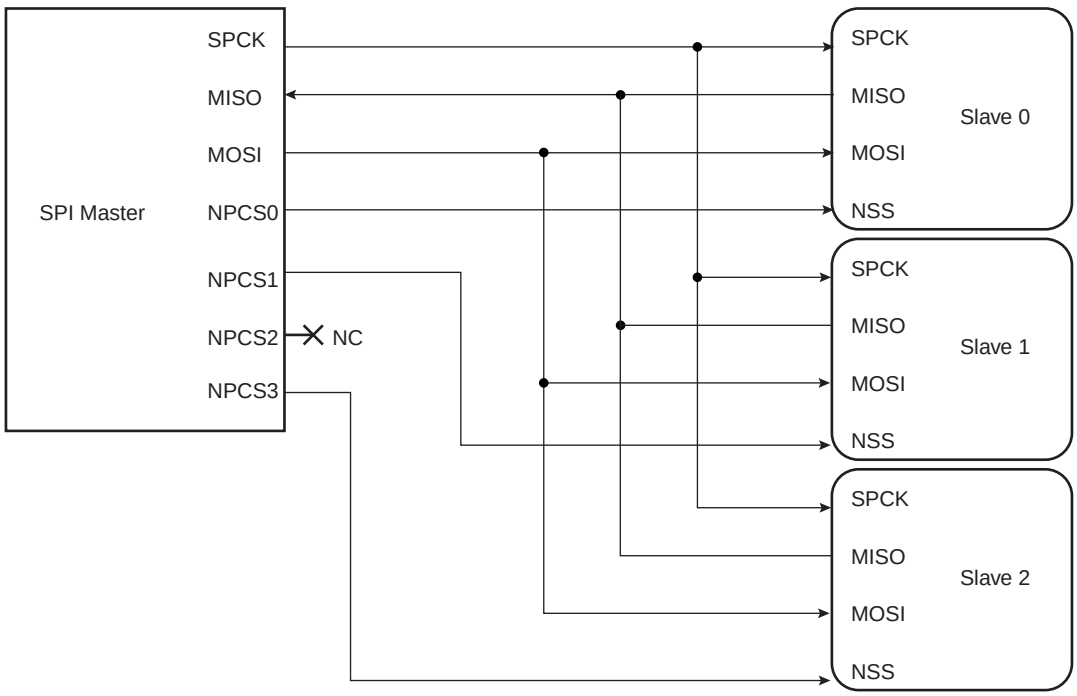
33.3 Block Diagram

Figure 33-1. Block Diagram



33.4 Application Block Diagram

Figure 33-2. Application Block Diagram: Single Master/Multiple Slave Implementation



33.5 Signal Description

Table 33-1. 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

33.6 Product Dependencies

33.6.1 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 33-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
SPI1	SPI1_MISO	PC3	A
SPI1	SPI1_MOSI	PC4	A
SPI1	SPI1_NPCS0	PC2	A
SPI1	SPI1_NPCS1	PC6	B
SPI1	SPI1_NPCS2	PC7	B
SPI1	SPI1_NPCS3	PC8	B
SPI1	SPI1_SPCK	PC5	A

33.6.2 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.

33.6.3 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 33-3. Peripheral IDs

Instance	ID
SPI1	40

33.6.4 Peripheral DMA Controller (PDC)

The SPI interface can be used in conjunction with the PDC in order to reduce processor overhead. For a full description of the PDC, refer to Section 28. ”Peripheral DMA Controller (PDC)” .

33.7 Functional Description

33.7.1 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).
  - Pins 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.

33.7.2 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.

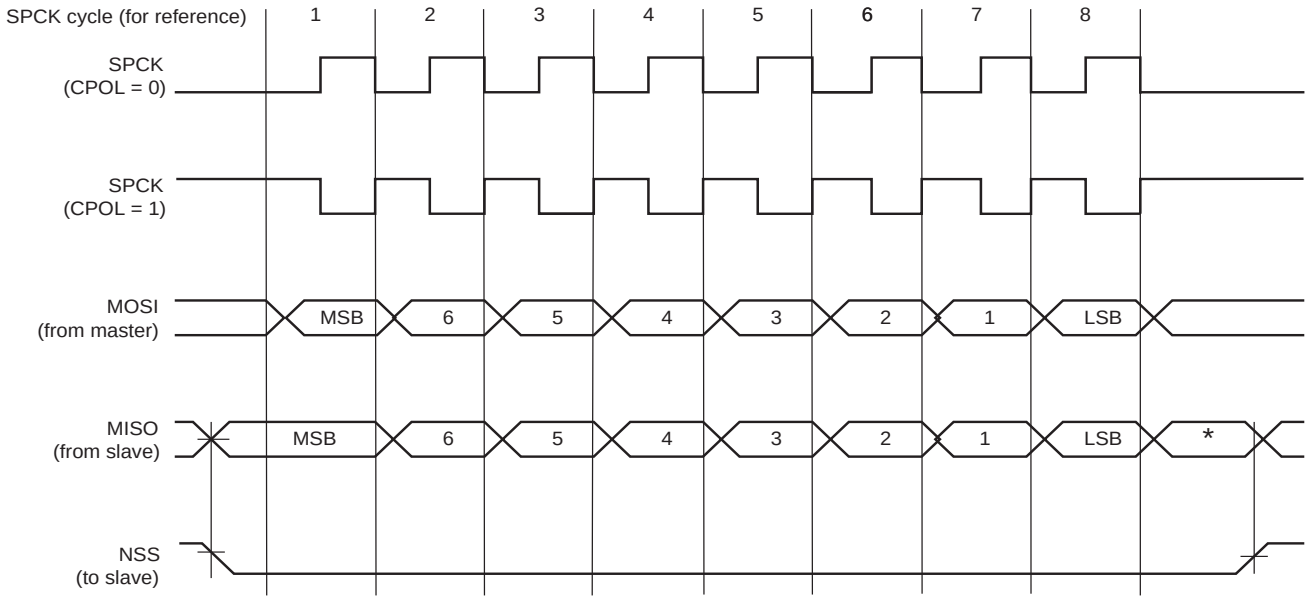
Table 33-4 shows the four modes and corresponding parameter settings.

Table 33-4. SPI Bus Protocol Modes

SPI Mode	CPOL	NCPHA	Shift SPCK Edge	Capture SPCK Edge	SPCK Inactive Level
0	0	1	Falling	Rising	Low
1	0	0	Rising	Falling	Low
2	1	1	Rising	Falling	High
3	1	0	Falling	Rising	High

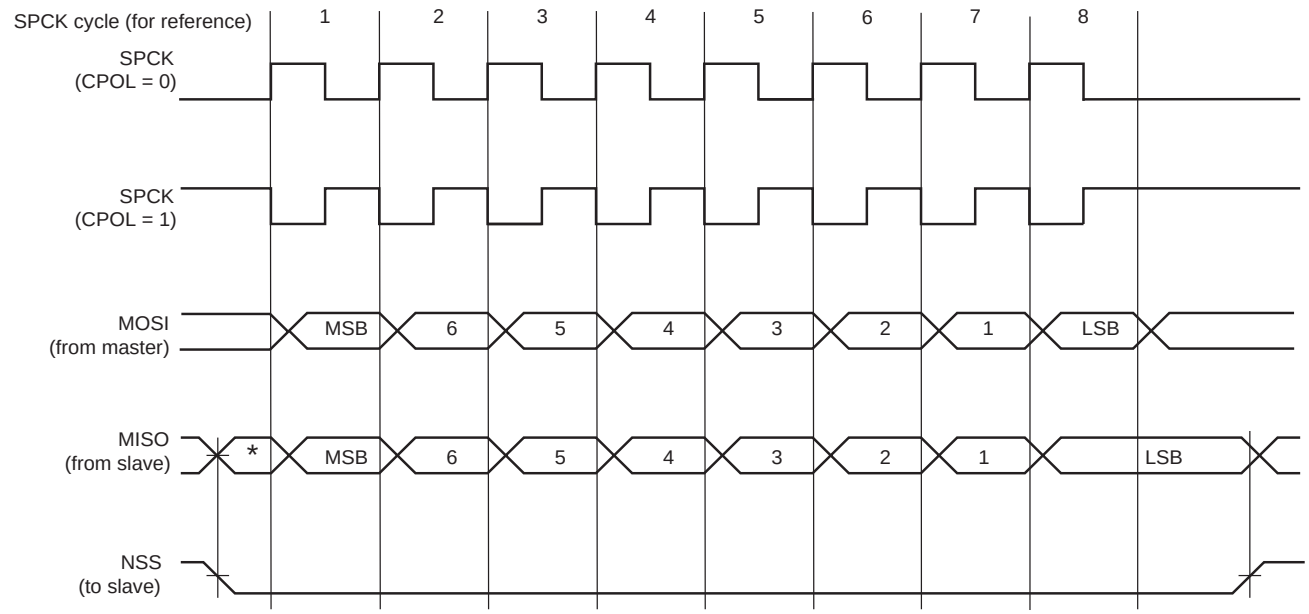
Figure 33-3 and Figure 33-4 show examples of data transfers.

Figure 33-3. SPI Transfer Format (NCPHA = 1, 8 bits per transfer)



* Not defined

Figure 33-4. SPI Transfer Format (NCPHA = 0, 8 bits per transfer)



* Not defined

33.7.3 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.

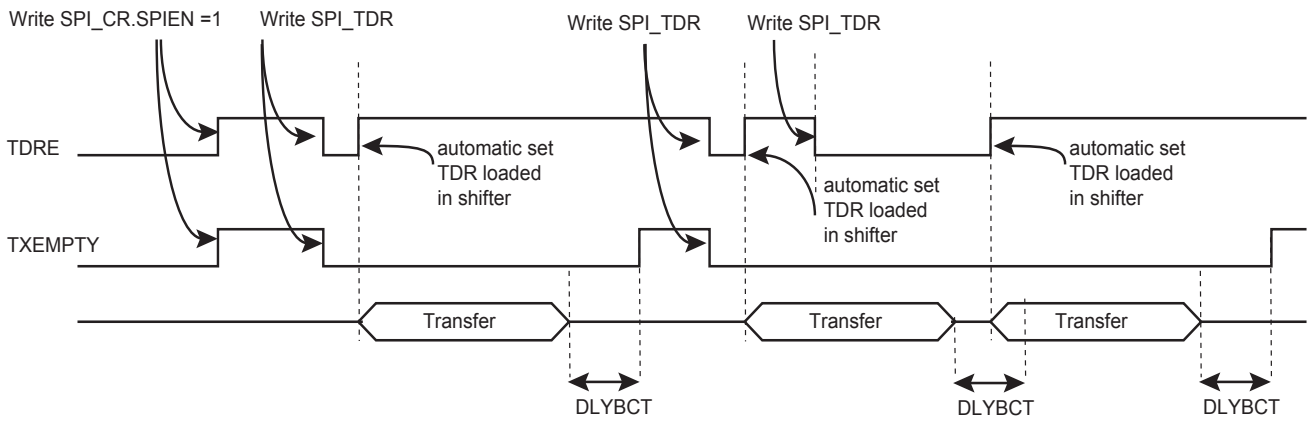
As soon as the SPI_TDR is written, the Transmit Data Register Empty (TDRE) flag in the SPI_SR is cleared. When the data written in the SPI_TDR is loaded into the Shift register, the TDRE flag in the SPI_SR is set. The TDRE bit is used to trigger the Transmit PDC channel.

See Figure 33-5.

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 33-5. TDRE and TXEMPTY flag behavior



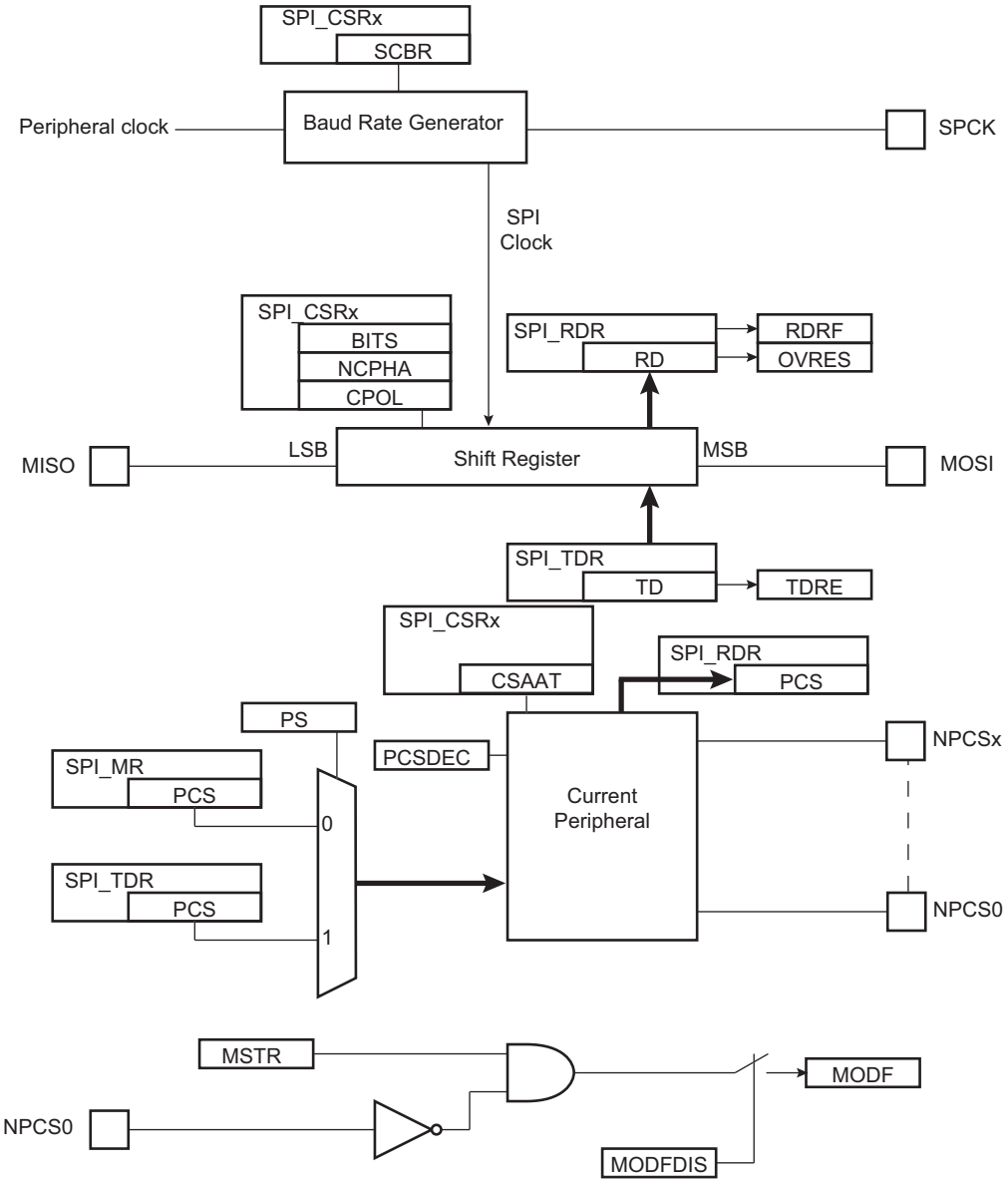
The transfer of received data from the Shift register to the SPI_RDR is indicated by the Receive Data Register Full (RDRF) bit in the SPI_SR. When the received data is read, the RDRF bit is cleared.

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 33-6, shows a block diagram of the SPI when operating in Master mode. Figure 33-7 shows a flow chart describing how transfers are handled.

33.7.3.1 Master Mode Block Diagram

Figure 33-6. Master Mode Block Diagram



33.7.3.2 Master Mode Flow Diagram

Figure 33-7. Master Mode Flow Diagram

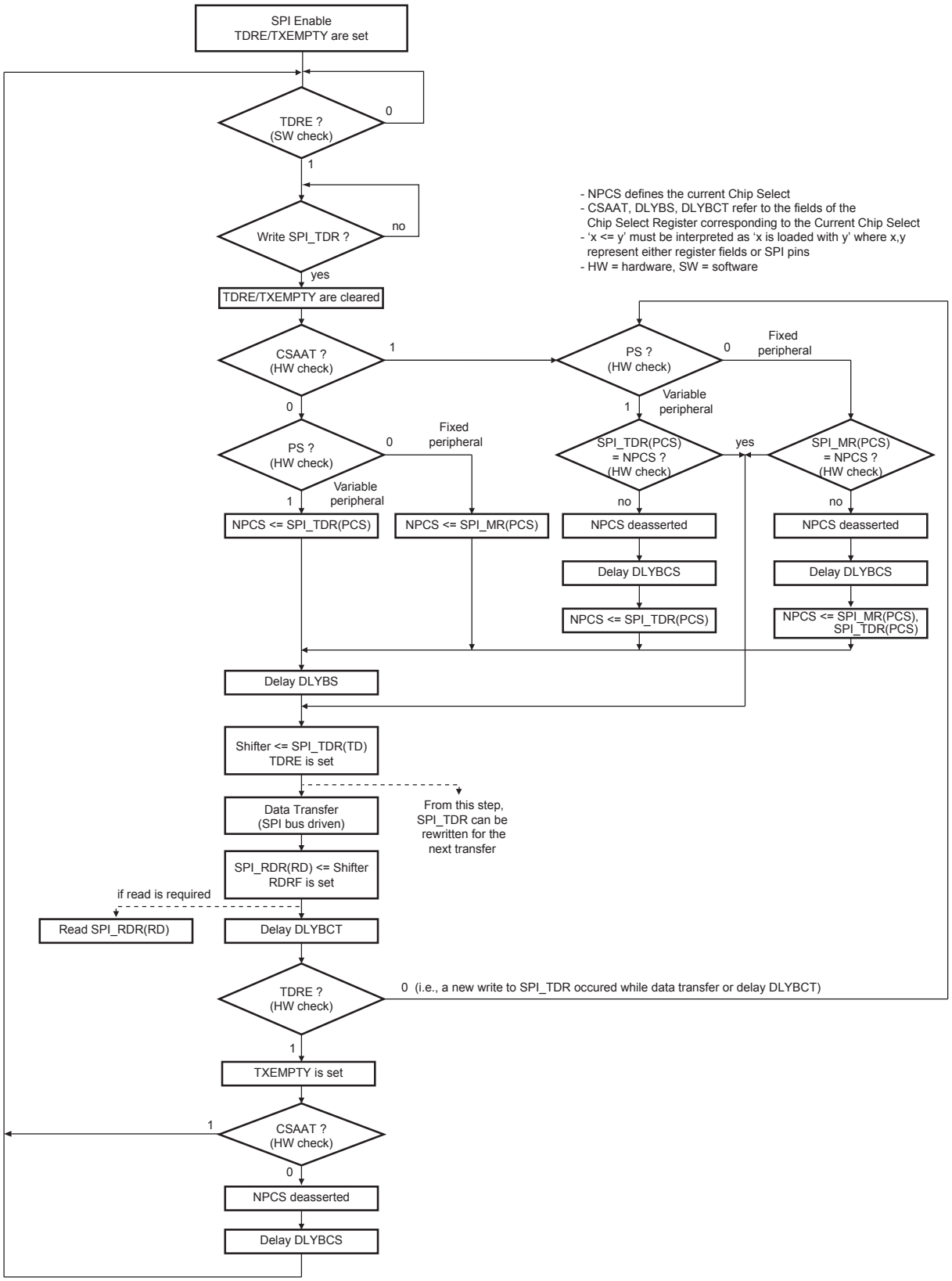


Figure 33-8 shows the behavior of Transmit Data Register Empty (TDRE), Receive Data Register (RDRF) and Transmission Register Empty (TXEMPTY) status flags within the SPI_SR during an 8-bit data transfer in Fixed mode without the PDC involved.

Figure 33-8. Status Register Flags Behavior

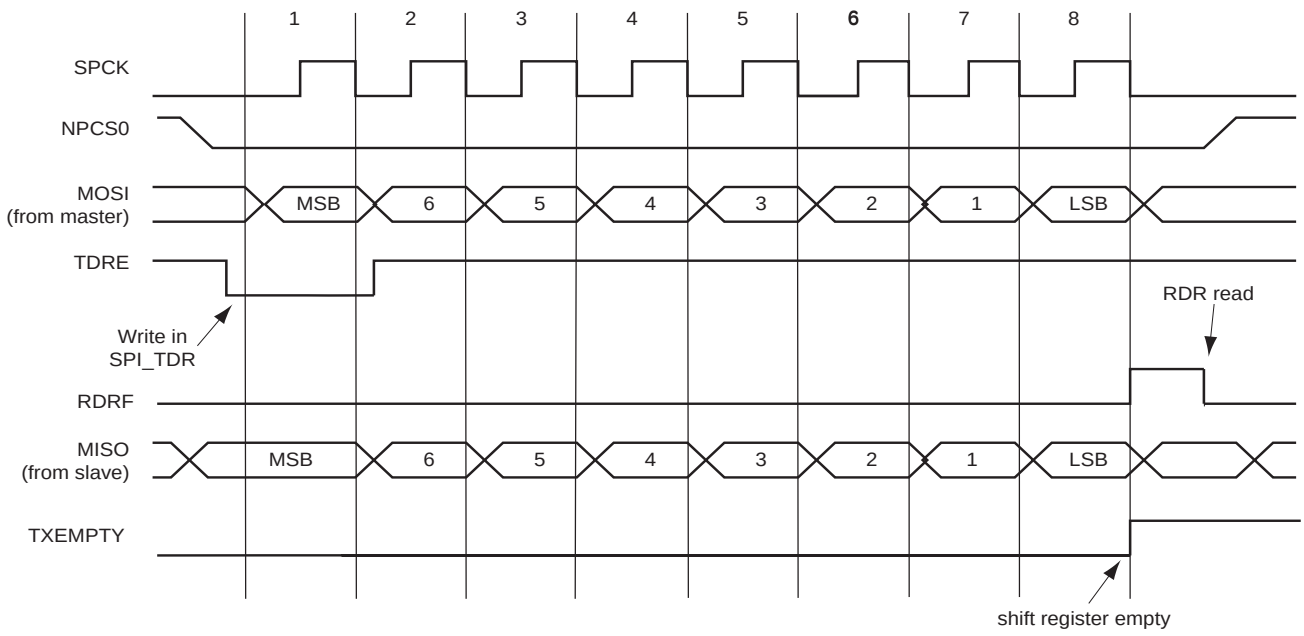
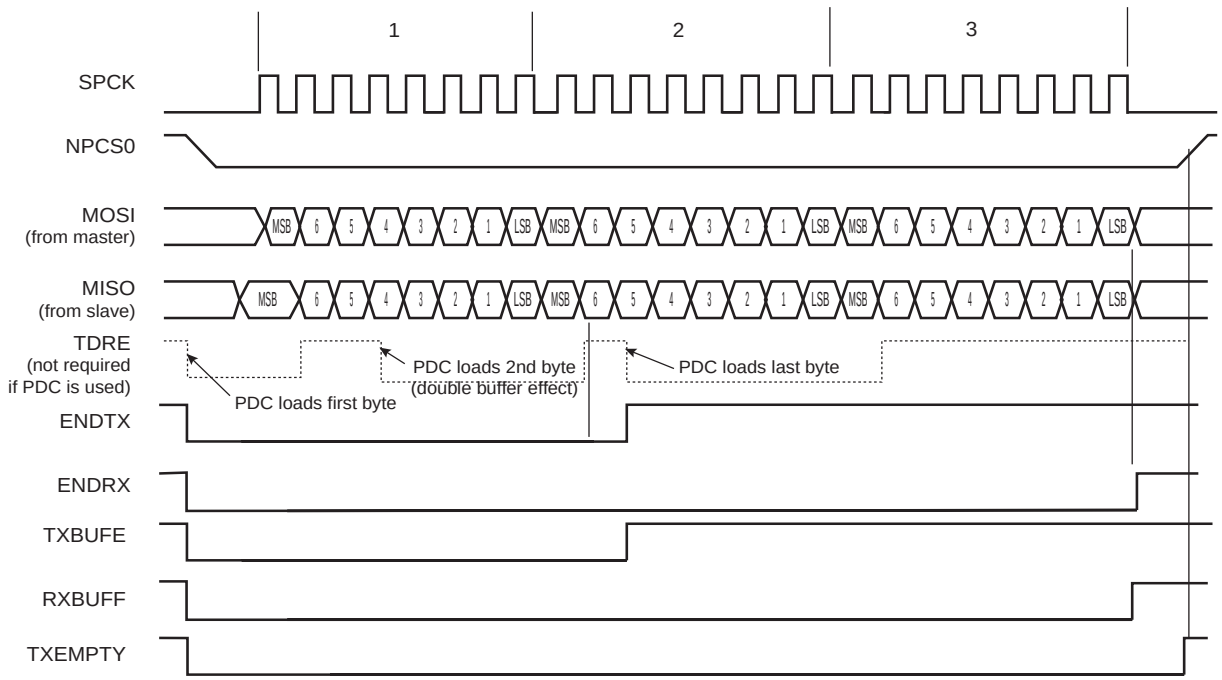


Figure 33-9 shows the behavior of Transmission Register Empty (TXEMPTY), End of RX buffer (ENDRX), End of TX buffer (ENDTX), RX Buffer Full (RXBUFF) and TX Buffer Empty (TXBUFE) status flags within the SPI_SR during an 8-bit data transfer in Fixed mode with the PDC involved. The PDC is programmed to 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 33-9. PDC Status Register Flags Behavior



33.7.3.3 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_CSRx 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.

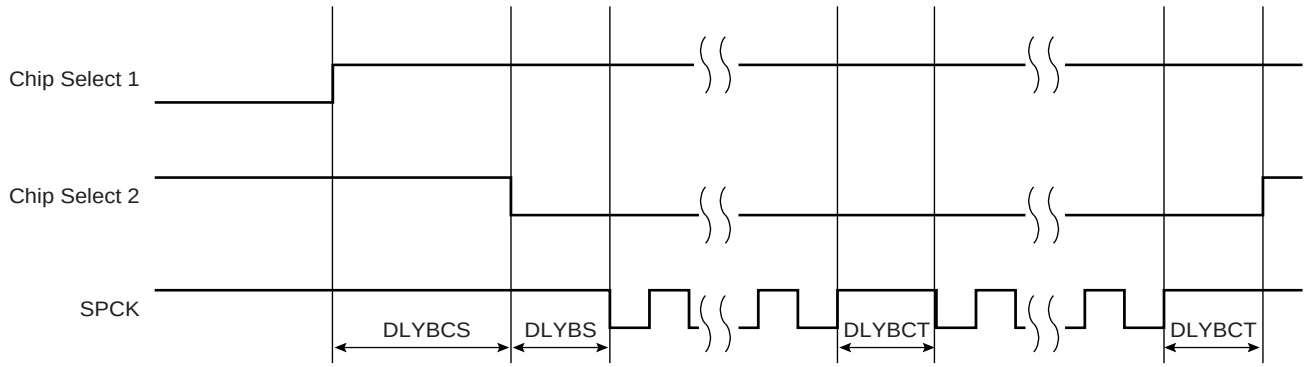
33.7.3.4 Transfer Delays

Figure 33-10 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 33-10. Programmable Delays



33.7.3.5 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 33.8.4 "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 33.7.3.9 "Peripheral Deselection with 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 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 PDC transfer can be started if the SPIEN has previously been written in the SPI_CR.

33.7.3.6 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 Section 28. "Peripheral DMA Controller (PDC)" .

33.7.3.7 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 33-11). 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.

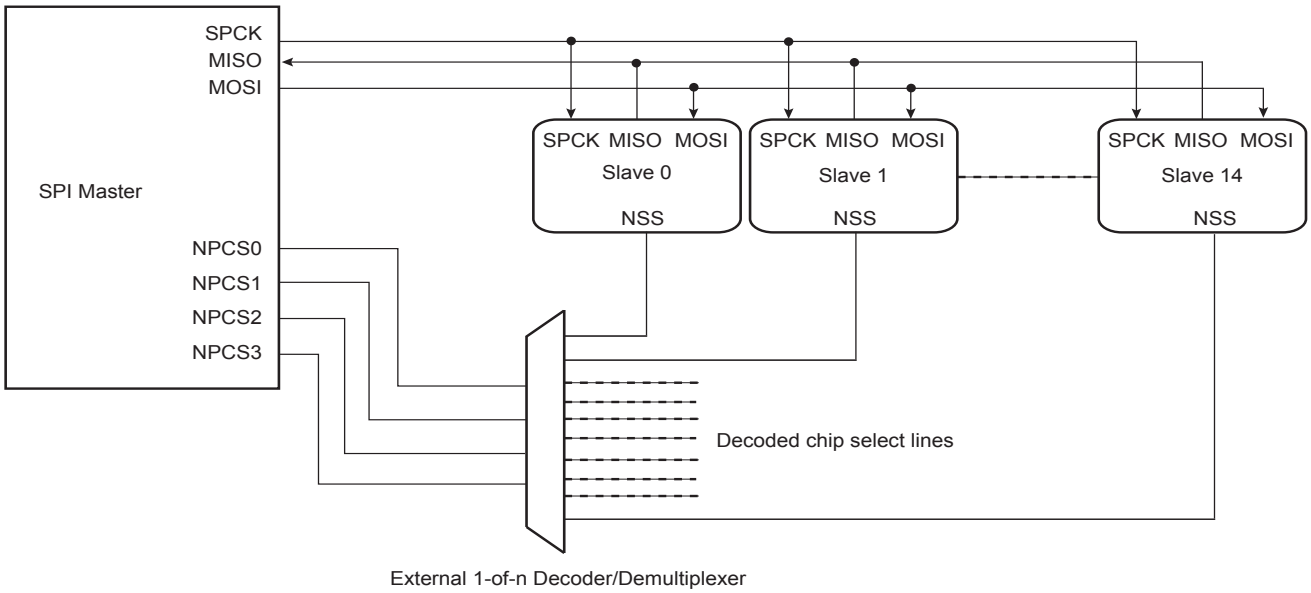
When operating with decoding, the SPI directly outputs the value defined by the PCS field on the NPCS lines of either SPI_MR or SPI_TDR (depending on PS).

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 33-11 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.

Figure 33-11. Chip Select Decoding Application Block Diagram: Single Master/Multiple Slave Implementation



33.7.3.8 Peripheral Deselection without PDC

During a transfer of more than one unit of data on a Chip Select without the 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 deasserted 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 deassert 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.

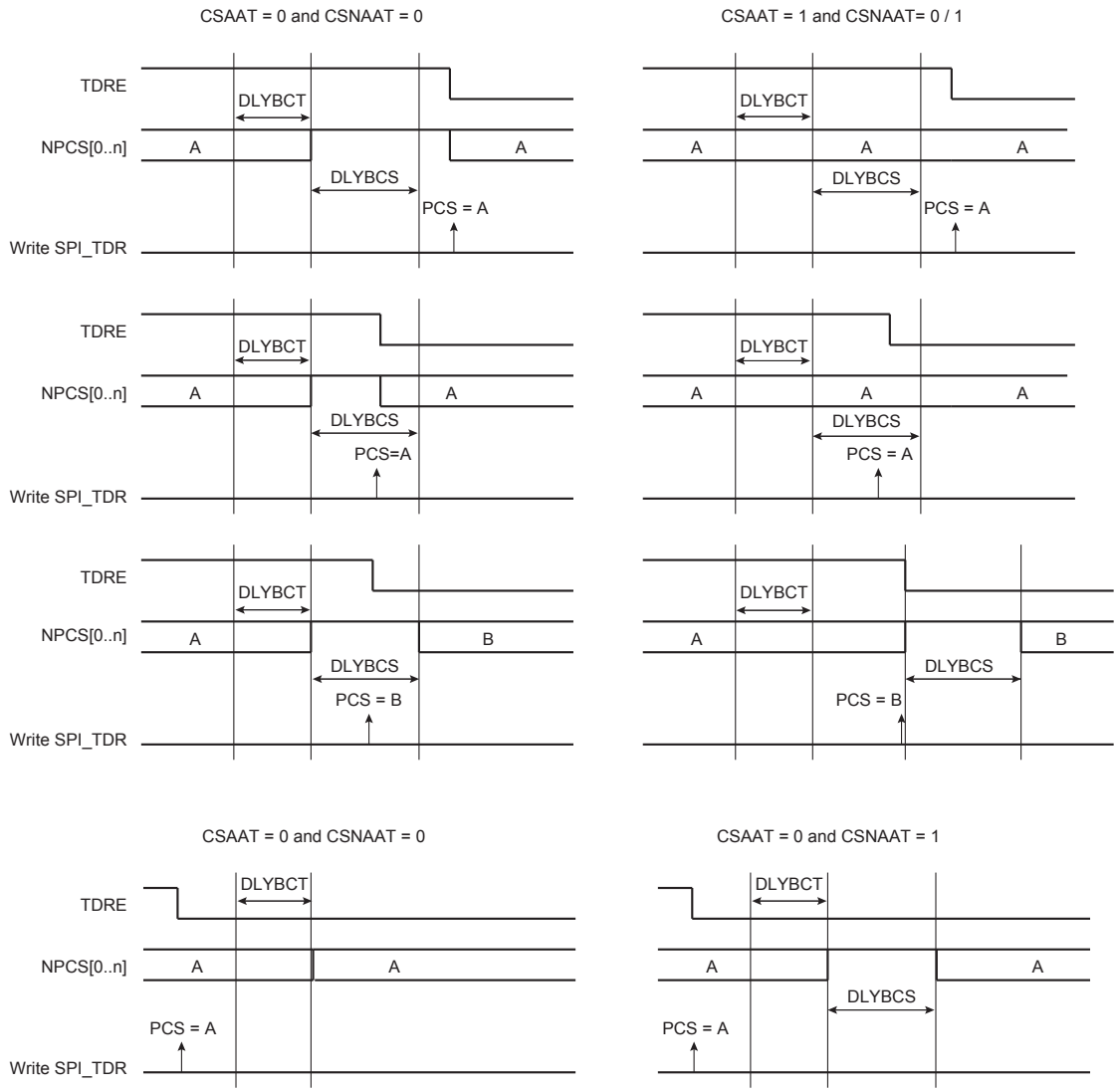
33.7.3.9 Peripheral Deselection with PDC

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 deassertion 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 deasserted between the two transfers. This can lead to difficulties to interface with some serial peripherals requiring the chip select to be deasserted 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 deasserted 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 33-12 shows different peripheral deselection cases and the effect of the CSAAT and CSNAAT bits.

Figure 33-12. Peripheral Deselection



33.7.3.10 Mode Fault Detection

The SPI has the capability to operate in multimaster 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 multimaster 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 reenabled 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.

33.7.4 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 fields 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 (1) below the SPI_CSRx bitmap (Section 33.8.9 "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.

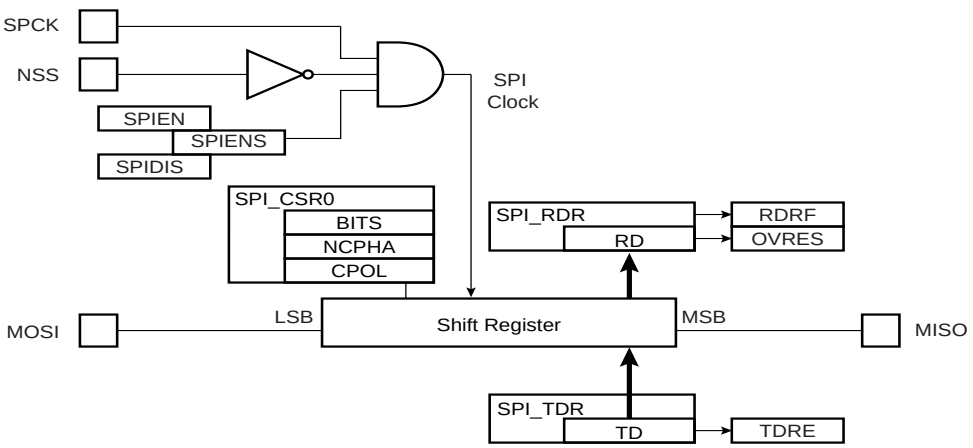
When a transfer starts, the data shifted out is the data present in the Shift register. If no data has been written in the SPI_TDR, the last data received is transferred. If no data has been received since the last reset, all bits are transmitted low, as the Shift register resets to 0.

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 33-13 shows a block diagram of the SPI when operating in Slave mode.

Figure 33-13. Slave Mode Functional Block Diagram



33.7.5 Register Write Protection

To prevent any single software error from corrupting SPI behavior, certain registers in the address space can be write-protected 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 are write-protected when WPEN is set in SPI_WPMR:

- "SPI Mode Register"
- "SPI Chip Select Register"

33.8 Serial Peripheral Interface (SPI) User Interface

Table 33-5. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	SPI_CR	Write-only	–
0x04	Mode Register	SPI_MR	Read/Write	0x0
0x08	Receive Data Register	SPI_RDR	Read-only	0x0
0x0C	Transmit Data Register	SPI_TDR	Write-only	–
0x10	Status Register	SPI_SR	Read-only	0x000000F0
0x14	Interrupt Enable Register	SPI_IER	Write-only	–
0x18	Interrupt Disable Register	SPI_IDR	Write-only	–
0x1C	Interrupt Mask Register	SPI_IMR	Read-only	0x0
0x20 - 0x2C	Reserved	–	–	–
0x30	Chip Select Register 0	SPI_CSR0	Read/Write	0x0
0x34	Chip Select Register 1	SPI_CSR1	Read/Write	0x0
0x38	Chip Select Register 2	SPI_CSR2	Read/Write	0x0
0x3C	Chip Select Register 3	SPI_CSR3	Read/Write	0x0
0x40 - 0xE0	Reserved	–	–	–
0xE4	Write Protection 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	–	–	–

33.8.1 SPI Control Register

Name: SPI_CR

Address: 0x48000000

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.

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.

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 deasserted 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 33.7.3.5 ”Peripheral Selection” for more details.

33.8.2 SPI Mode Register

Name: SPI_MR

Address: 0x48000004

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 nonoverlapping 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:

$$\text{Delay Between Chip Selects} = \frac{DLYBCS}{f_{peripheralclock}}$$

33.8.3 SPI Receive Data Register

Name: SPI_RDR

Address: 0x48000008

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.

33.8.4 SPI Transmit Data Register

Name: SPI_TDR

Address: 0x4800000C

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 deasserted 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).

33.8.5 SPI Status Register

Name: SPI_SR

Address: 0x48000010

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.

33.8.6 SPI Interrupt Enable Register

Name: SPI_IER

Address: 0x48000014

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**

33.8.7 SPI Interrupt Disable Register

Name: SPI_IDR

Address: 0x48000018

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**

33.8.8 SPI Interrupt Mask Register

Name: SPI_IMR

Address: 0x4800001C

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**

33.8.9 SPI Chip Select Register

Name: SPI_CSRx [x=0..3]

Address: 0x48000030, 0x48000034, 0x48000038, 0x4800003C

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: 1. SPI_CSRx 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:

$$\frac{DLYBCS}{f_{peripheralclock}}$$ (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 [(1)] below the SPI_CSR 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 from 1 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.

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:

$$DLYBS = Delay\ Before\ SPCK \times f_{peripheral\ clock}$$

- **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:

$$DLYBCT = (\text{Delay Between Consecutive Transfers} \times f_{\text{peripheral clock}}) / 32$$

33.8.10 SPI Write Protection Mode Register

Name: SPI_WPMR

Address: 0x480000E4

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).

- **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.

See Section 33.7.5 ”Register Write Protection” for the list of registers that can be write-protected.

33.8.11 SPI Write Protection Status Register

Name: SPI_WPSR

Address: 0x480000E8

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.

34. Two-wire Interface (TWI)

34.1 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 34-1 lists the compatibility level of the Atmel Two-wire Interface in Master mode and a full I²C compatible device.

Table 34-1. 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

34.2 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 34-1 for details on compatibility with I²C Standard.

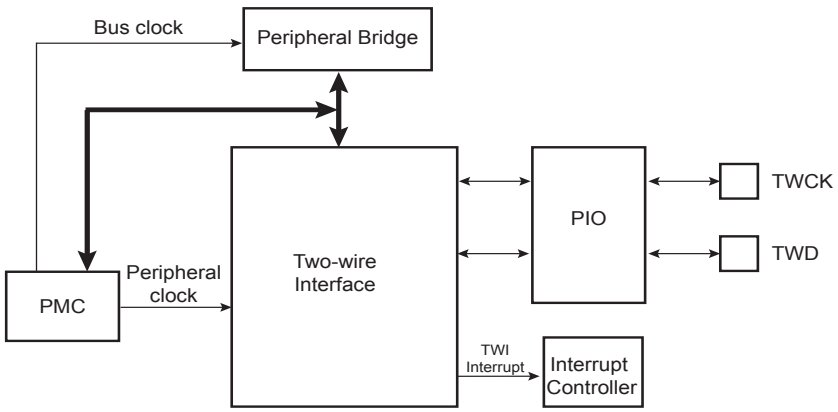
34.3 List of Abbreviations

Table 34-2. 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

34.4 Block Diagram

Figure 34-1. Block Diagram



34.5 I/O Lines Description

Table 34-3. 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

34.6 Product Dependencies

34.6.1 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 34-4. I/O Lines

Instance	Signal	I/O Line	Peripheral
TWI0	TWCK0	PA25	A
TWI0	TWD0	PA24	A
TWI1	TWCK1	PB1	A
TWI1	TWD1	PB0	A

34.6.2 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.

34.6.3 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 34-5. Peripheral IDs

Instance	ID
TWI0	19
TWI1	20

34.7 Functional Description

34.7.1 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 34-3).

Each transfer begins with a START condition and terminates with a STOP condition (see Figure 34-2).

- 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 34-2. START and STOP Conditions

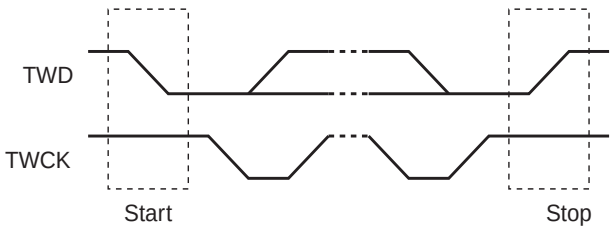
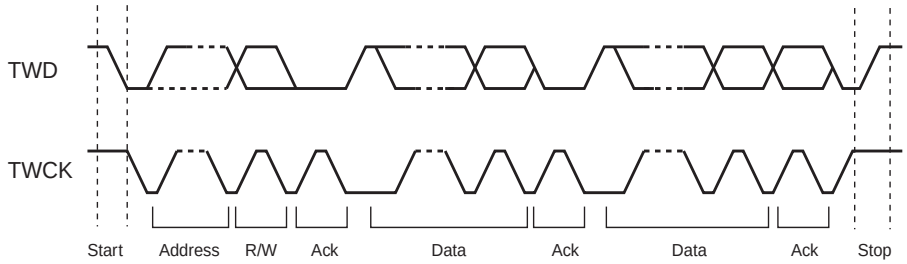


Figure 34-3. Transfer Format



34.7.2 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.

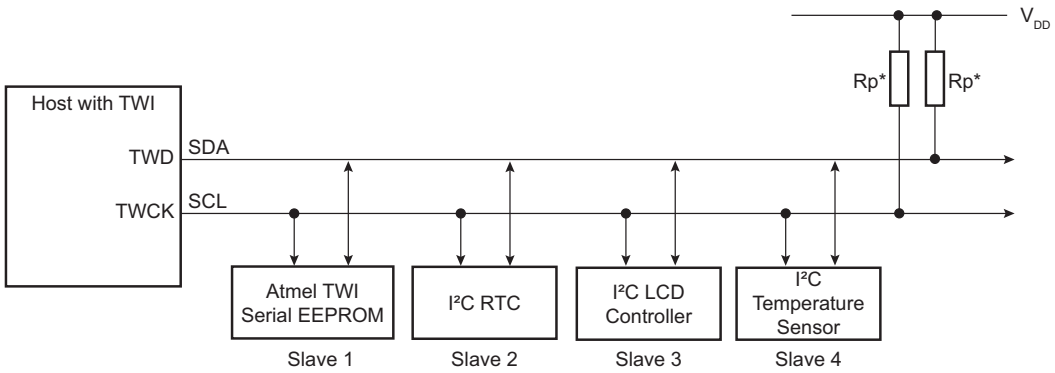
34.7.3 Master Mode

34.7.3.1 Definition

The master is the device that starts a transfer, generates a clock and stops it.

34.7.3.2 Application Block Diagram

Figure 34-4. Master Mode Typical Application Block Diagram



* Rp: Pull-up value as given by the I²C Standard

34.7.3.3 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.

34.7.3.4 Master Transmitter Mode

After the master initiates a START condition when writing into the Transmit Holding Register (TWI_THR), it sends a 7-bit slave address, configured in the Master Mode register (DADR in TWI_MMR), to notify the slave device. The bit following the slave address indicates the transfer direction, 0 in this case (MREAD = 0 in TWI_MMR).

The TWI transfers require the slave to acknowledge each received byte. During the acknowledge clock pulse (9th pulse), the master releases the data line (HIGH), enabling the slave to pull it down in order to generate the acknowledge. 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 34-5, Figure 34-6, and Figure 34-7.

To clear the TXRDY flag, first set the bit TWI_CR.MSDIS, then set the bit TWI_CR.MSEN.

Figure 34-5. Master Write with One Data Byte

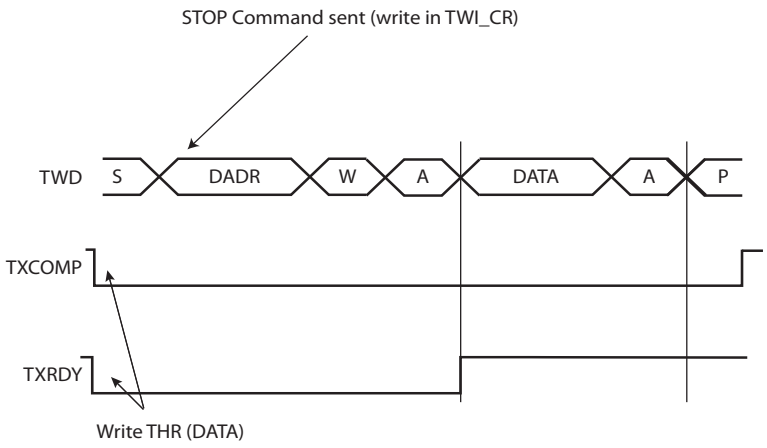


Figure 34-6. Master Write with Multiple Data Bytes

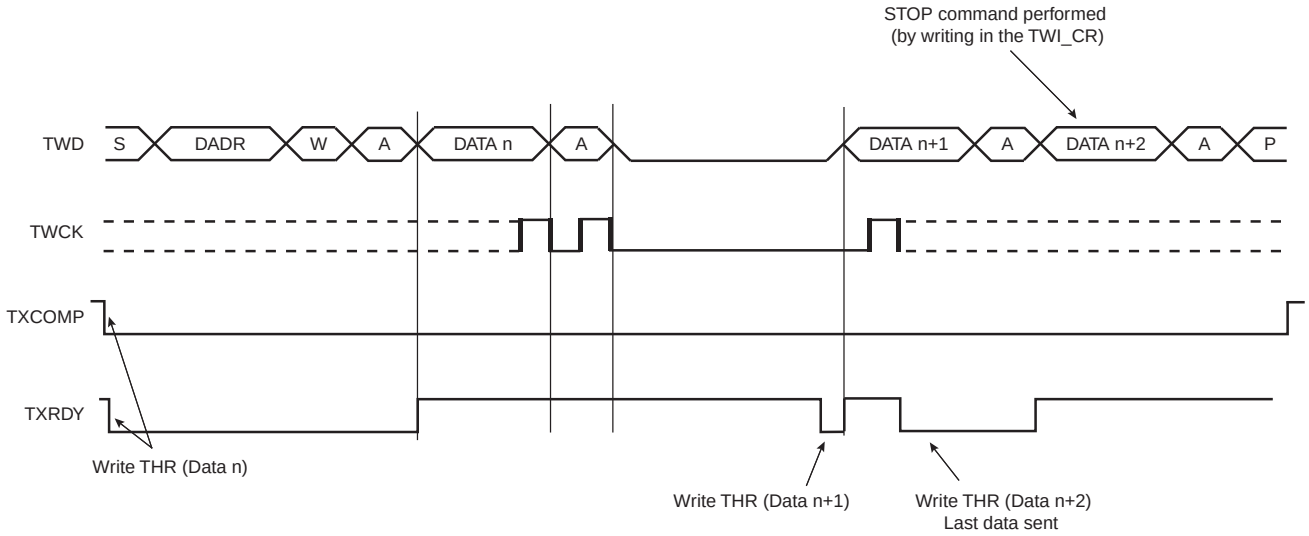
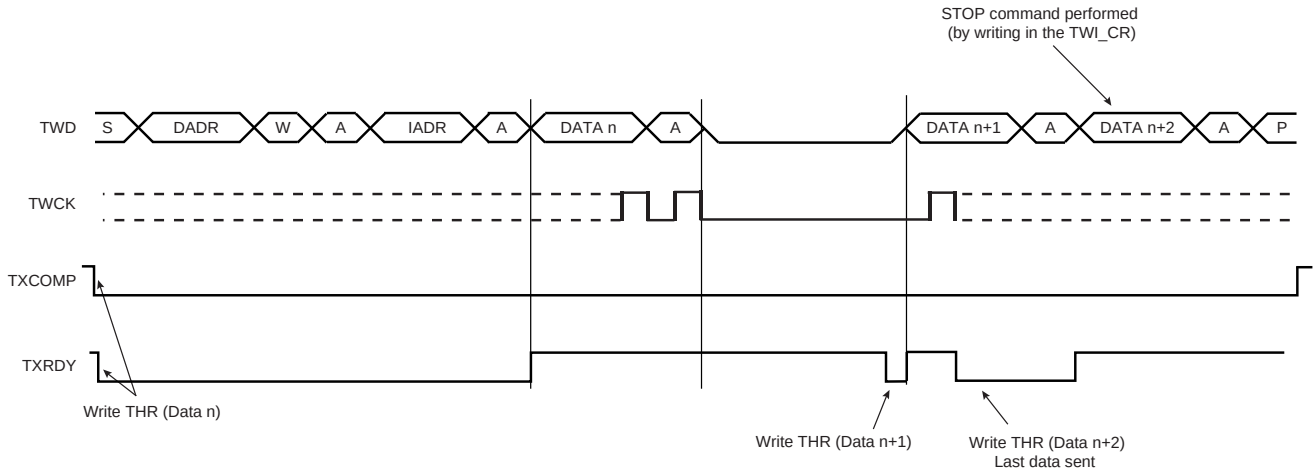


Figure 34-7. Master Write with One Byte Internal Address and Multiple Data Bytes



34.7.3.5 Master Receiver Mode

The read sequence begins by setting the START bit. After the START condition has been sent, the master sends a 7-bit slave address to notify the slave device. The bit following the slave address indicates the transfer direction—1 in this case (MREAD = 1 in TWI_MMR). During the acknowledge clock pulse (9th pulse), the master releases the data line (HIGH), enabling the slave to pull it down in order to generate the acknowledge. The master polls the data line during this clock pulse and sets the NACK bit in the 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 34-8. 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 34-8. 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 34-9. For internal address usage, see Section 34.7.3.6.

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 34-10.

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 34-8. Master Read with One Data Byte

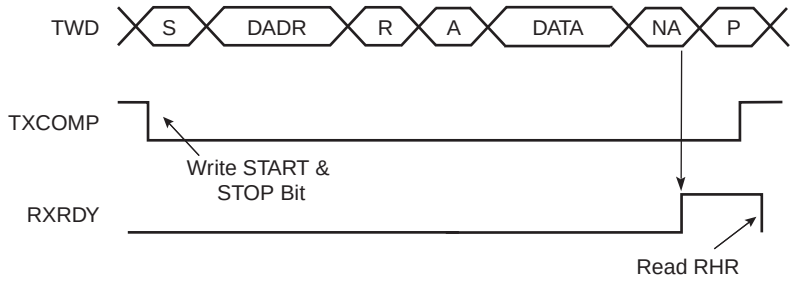


Figure 34-9. Master Read with Multiple Data Bytes

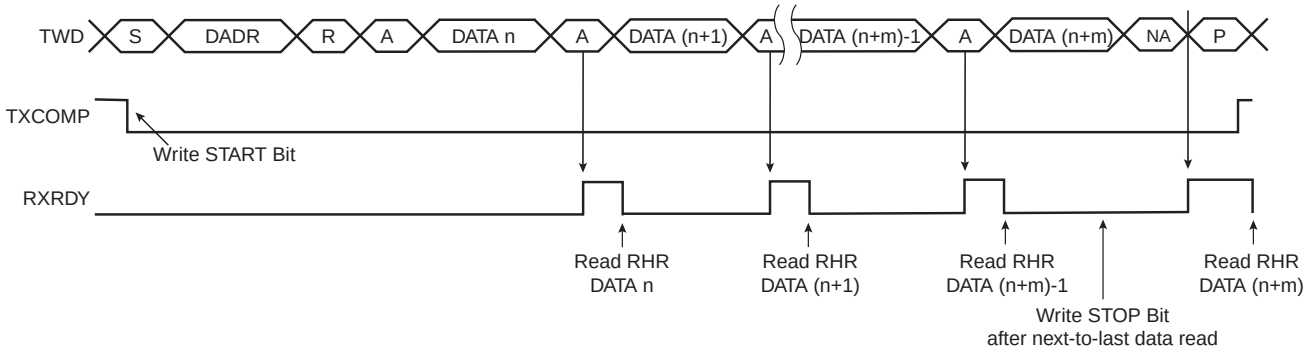
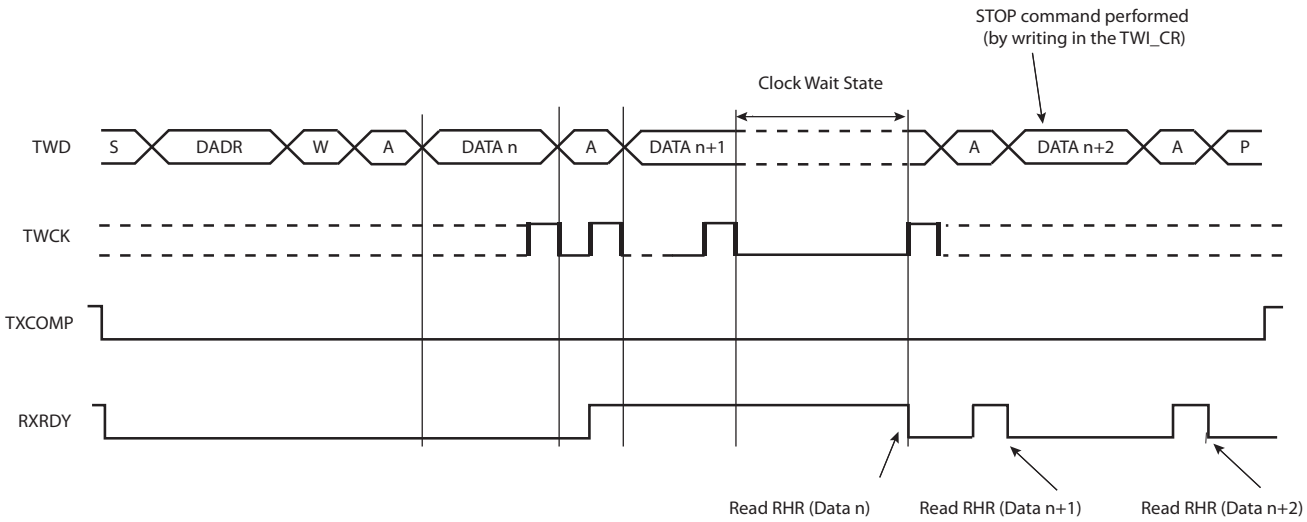


Figure 34-10. Master Read Wait State with Multiple Data Bytes



34.7.3.6 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, i.e., 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^2C fully-compatible devices. See Figure 34-12. See Figure 34-11 and Figure 34-13 for master write operation with internal address.

The three internal address bytes are configurable through the Master Mode register (TWI_MMR).

If the slave device supports only a 7-bit address, i.e., no internal address, IADRSZ must be set to 0.

Table 34-6 shows the abbreviations used in Figure 34-11 and Figure 34-12.

Table 34-6. 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 34-11. Master Write with One, Two or Three Bytes Internal Address and One Data Byte

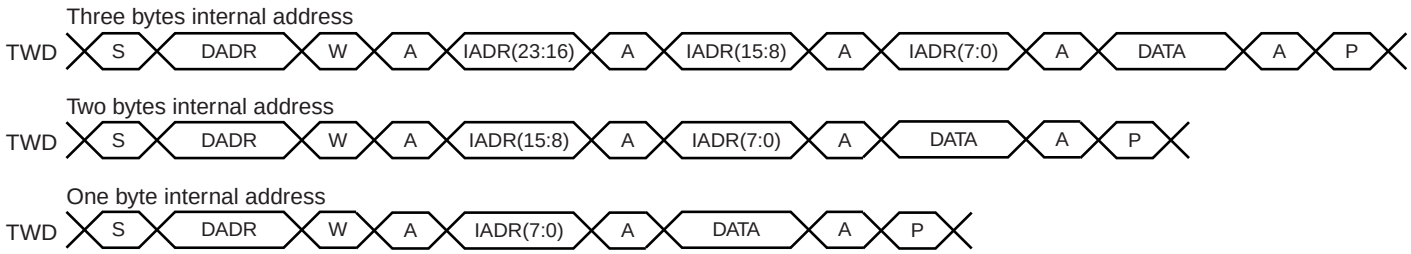
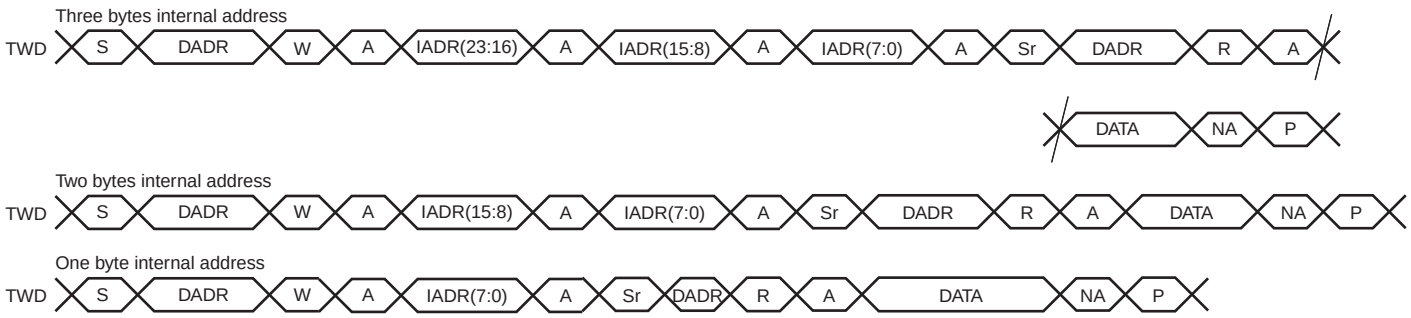


Figure 34-12. Master Read with One, Two or Three Bytes Internal Address and One Data Byte



10-bit Slave Addressing

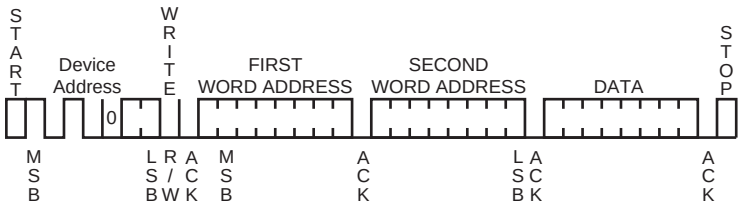
For a slave address higher than seven bits, the user must configure the address size **(**IADRSZ) and set the other slave address bits in the Internal Address register (TWI_IADR). The two remaining internal address bytes, IADR[15:8] and IADR[23:16] can be used the same way as in 7-bit slave addressing.

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 34-13 below shows a byte write to a memory device. This demonstrates the use of internal addresses to access the device.

Figure 34-13. Internal Address Usage



34.7.3.7 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.

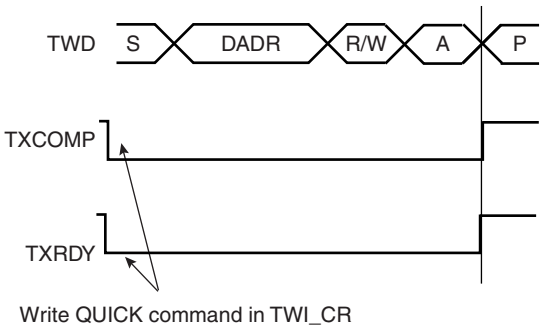
10. Wait for the RXRDY flag in TWI_SR.
11. Read the last character in TWI_RHR.
12. (Only if peripheral clock must be disabled) Wait for the TXCOMP flag to be raised in TWI_SR.

34.7.3.8 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 34-14. SMBus Quick Command



34.7.3.9 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 34-15. TWI Write Operation with Single Data Byte without Internal Address

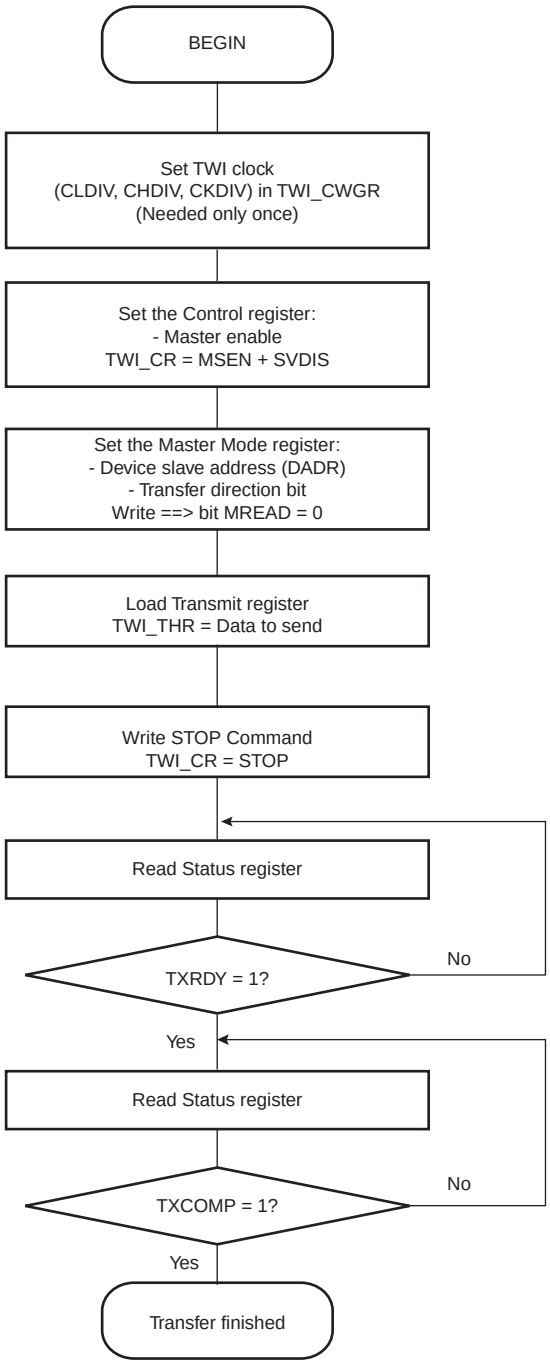


Figure 34-16. TWI Write Operation with Single Data Byte and Internal Address

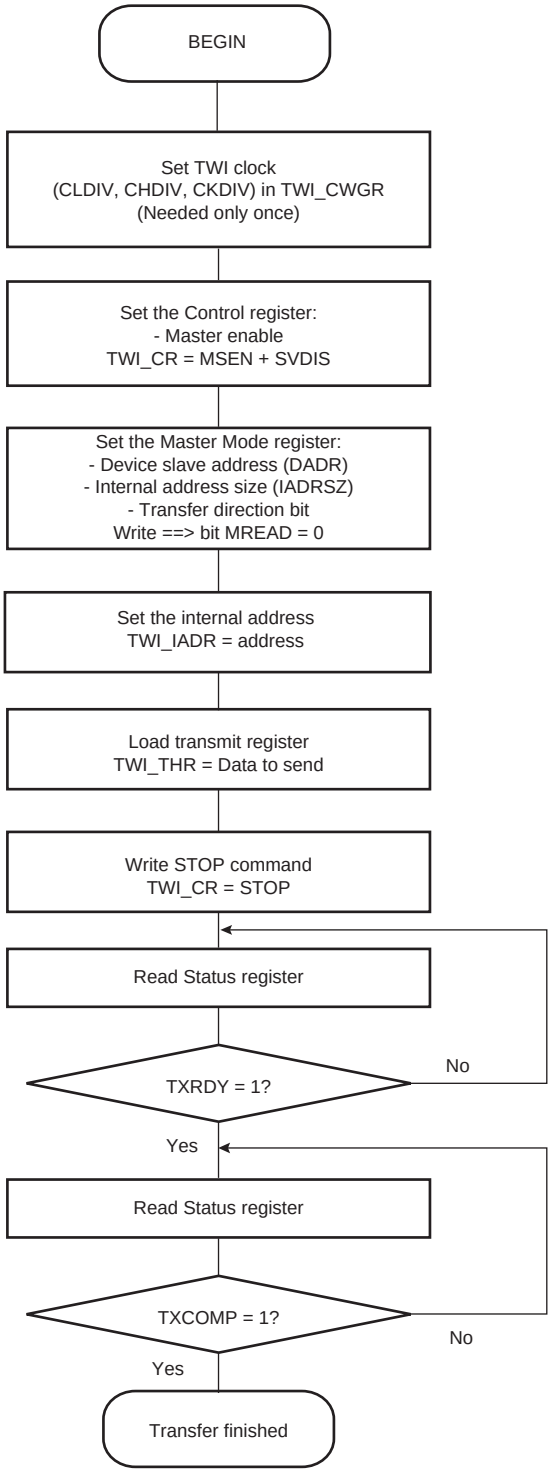


Figure 34-17. TWI Write Operation with Multiple Data Bytes with or without Internal Address

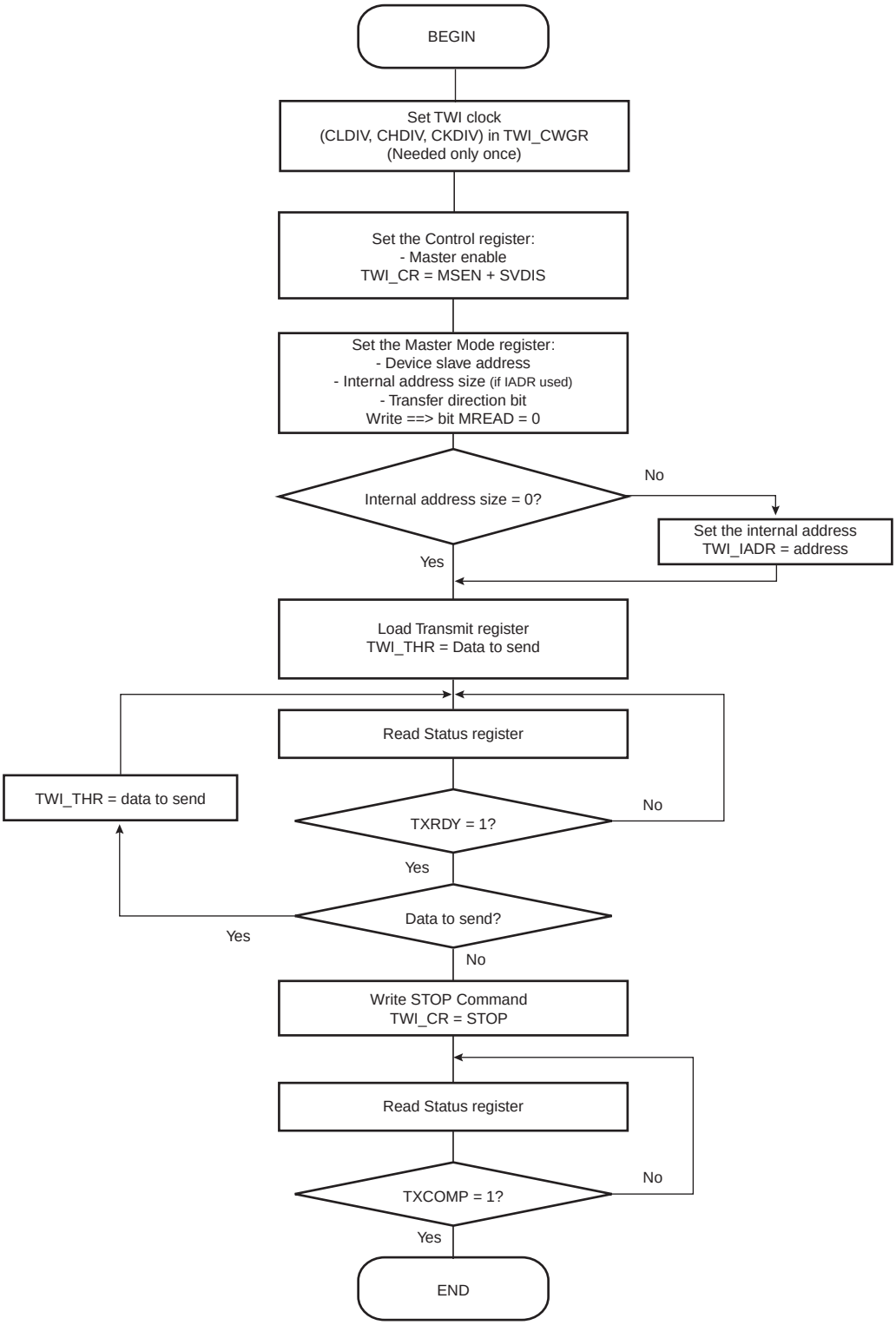


Figure 34-18. TWI Read Operation with Single Data Byte without Internal Address

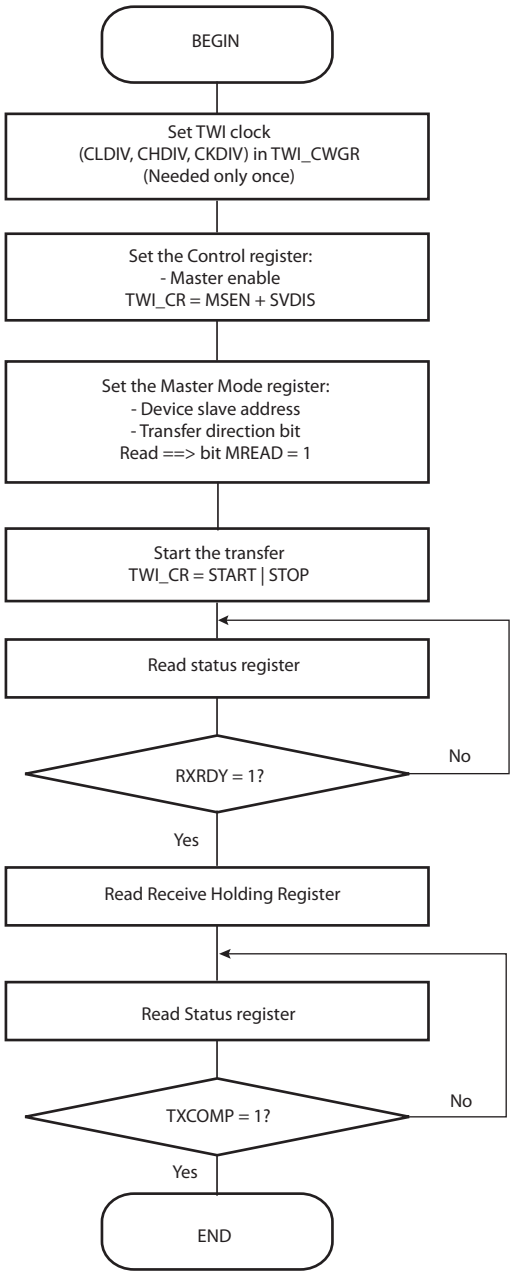


Figure 34-19. TWI Read Operation with Single Data Byte and Internal Address

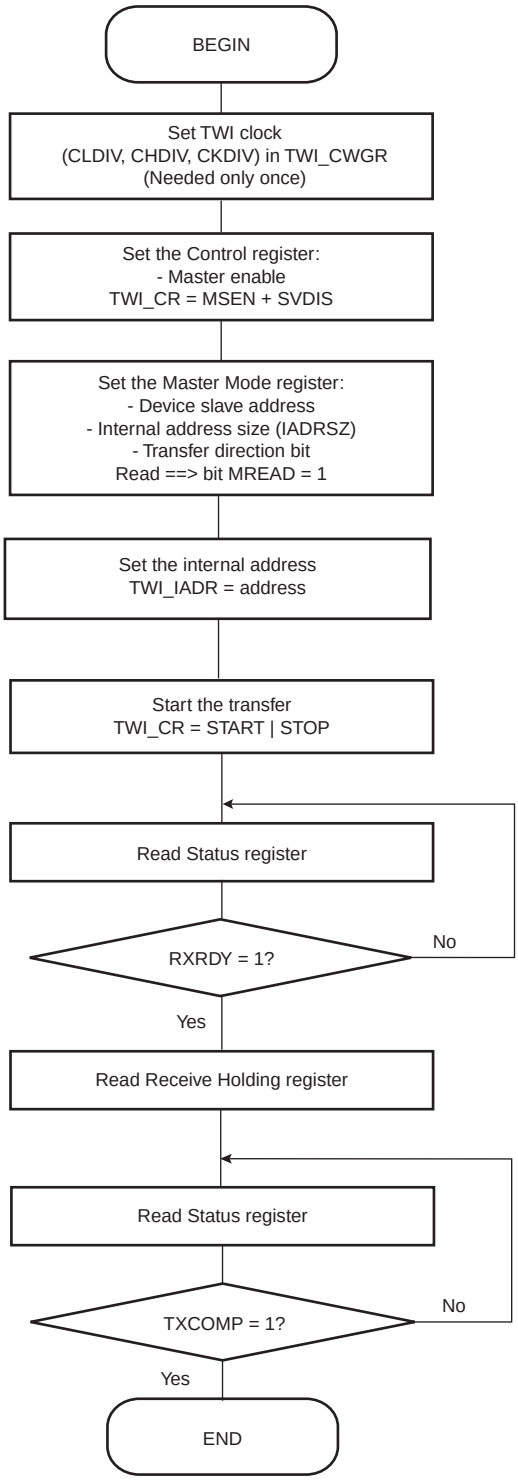
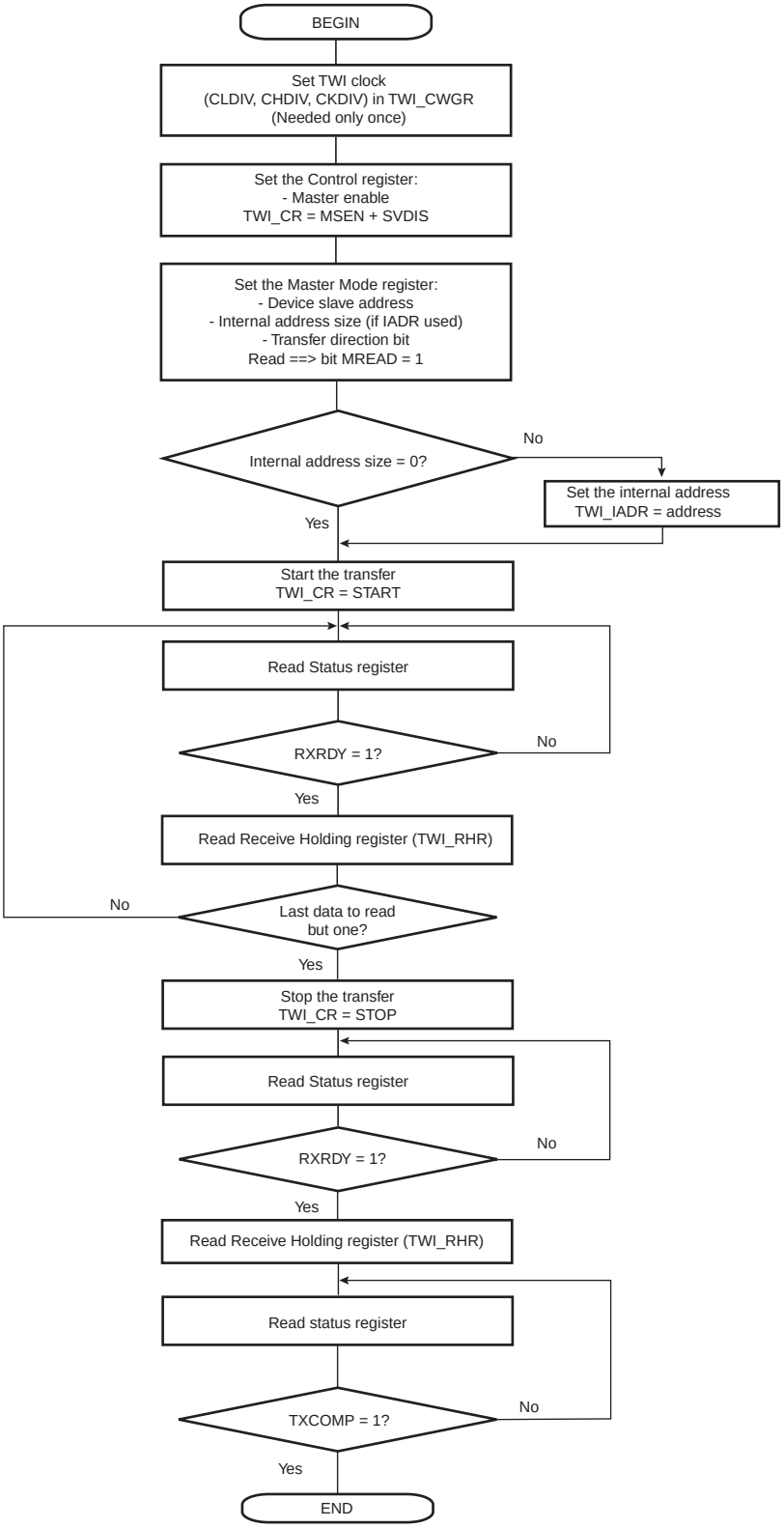


Figure 34-20. TWI Read Operation with Multiple Data Bytes with or without Internal Address



34.7.4 Multi-master Mode

34.7.4.1 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 34-22.

34.7.4.2 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 (i.e., 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 34-21).

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 34-21. Programmer Sends Data While the Bus is Busy

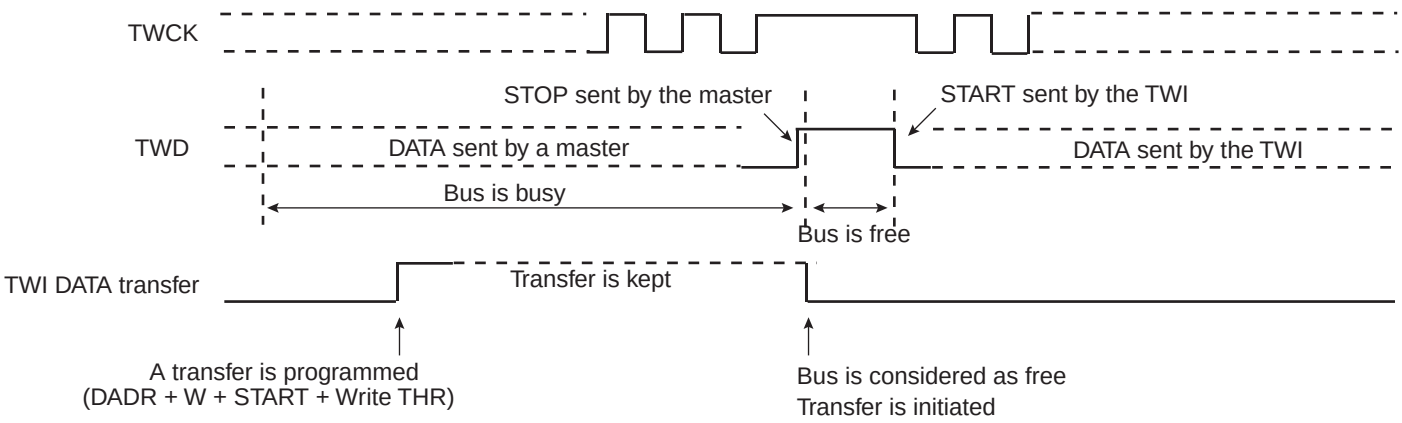
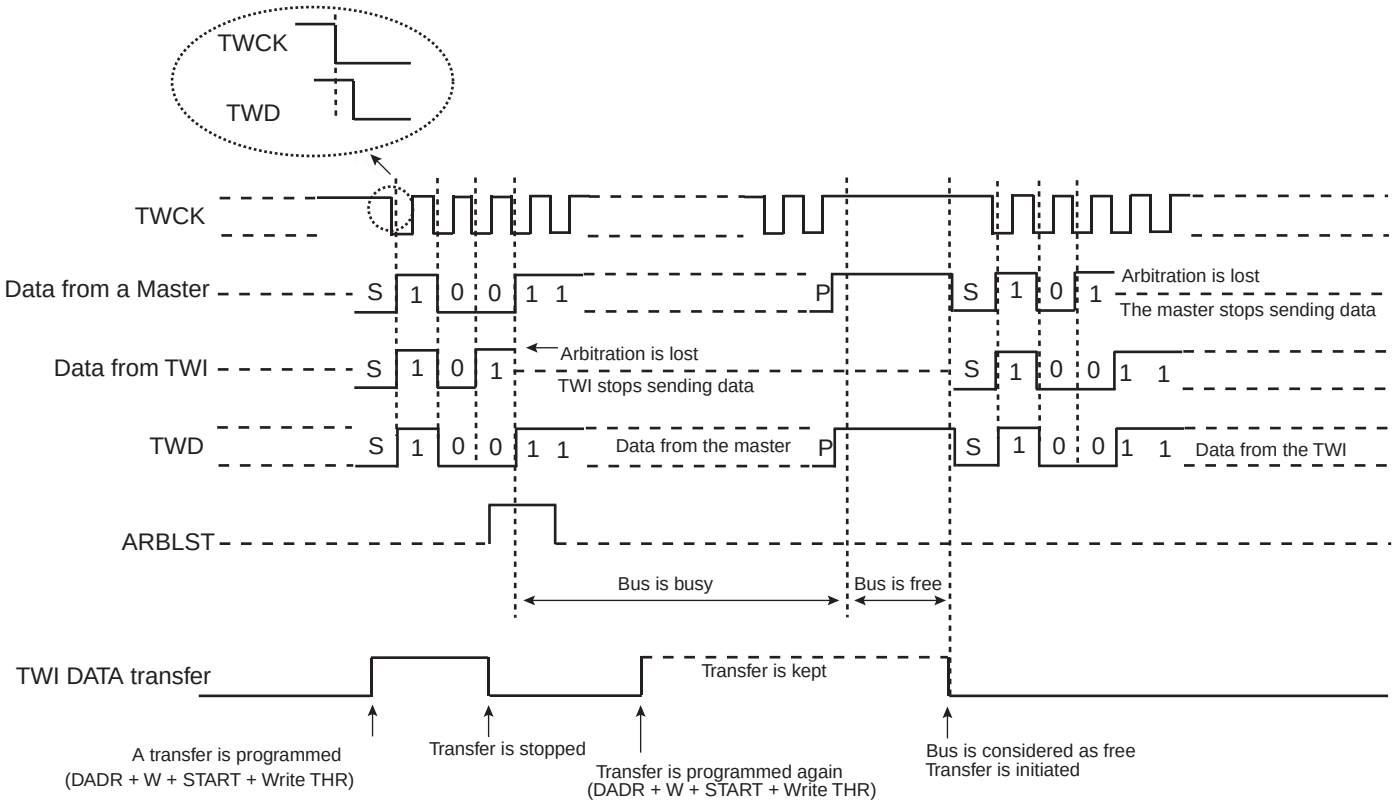
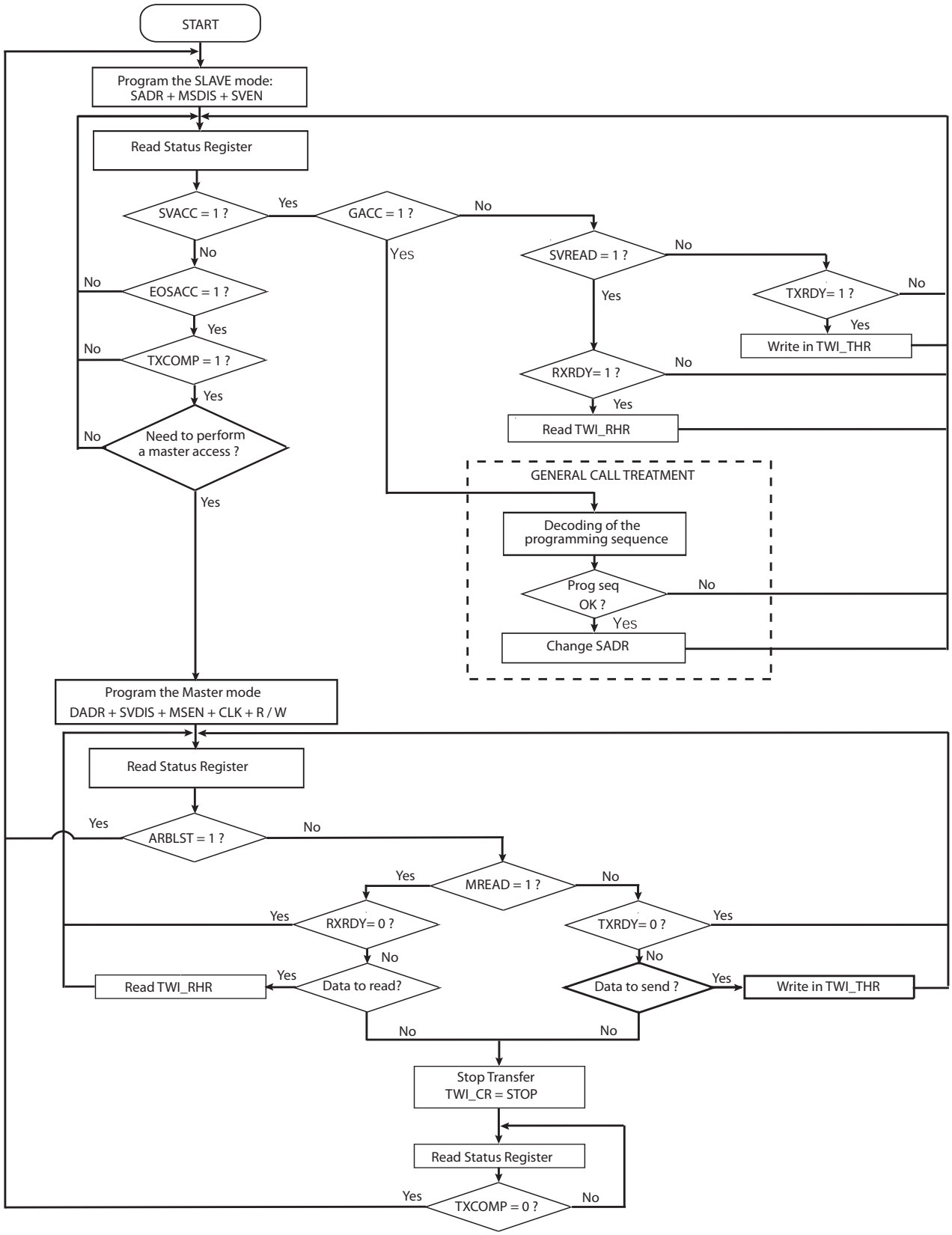


Figure 34-22. Arbitration Cases



The flowchart shown in Figure 34-23 gives an example of read and write operations in Multi-master mode.

Figure 34-23. Multi-master Flowchart



34.7.5 Slave Mode

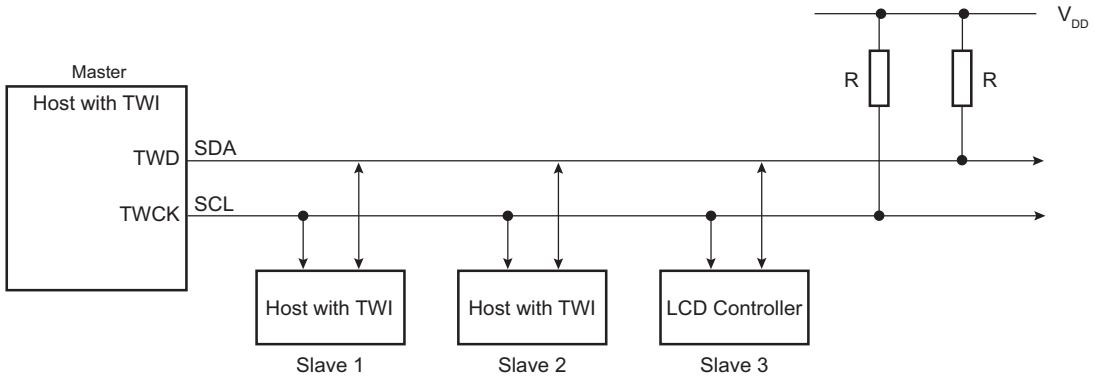
34.7.5.1 Definition

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).

34.7.5.2 Application Block Diagram

Figure 34-24. Slave Mode Typical Application Block Diagram



34.7.5.3 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.

34.7.5.4 Receiving Data

After a START or REPEATED START condition is detected and if the address sent by the Master matches with the Slave address programmed in the SADR (Slave Address) field, SVACC (Slave Access) flag is set and SVREAD (Slave Read) indicates the direction of the transfer.

SVACC remains high until a STOP condition or a repeated START is detected. When such a condition is detected, 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 the TWI_THR (TWI Transmit Holding Register) until a STOP condition or a REPEATED START + an address different from SADR is detected. Note that at the end of the read sequence TXCOMP (Transmission Complete) flag is set and SVACC reset.

As soon as data is written in the TWI_THR, 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 34-25.

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 34-26.

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 34-29.

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 34-28.

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 34-27.

34.7.5.5 Data Transfer

Read Operation

The Read mode is defined as a data requirement from the master.

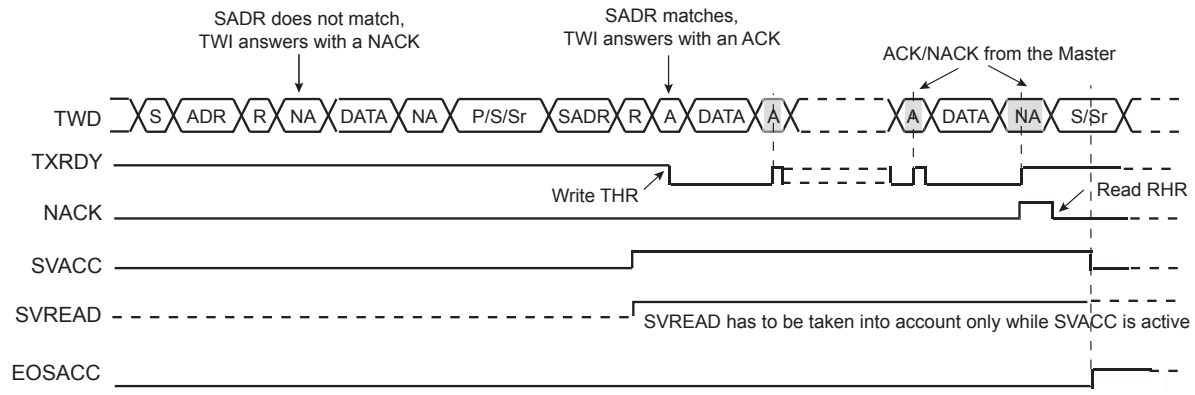
After a START or a REPEATED START condition is detected, the decoding of the address starts. If the slave address (SADR) is decoded, SVACC is set and SVREAD indicates the direction of the transfer.

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 34-25 describes the write operation.

Figure 34-25. Read Access Ordered by a Master



Notes: 1. When SVACC is low, the state of SVREAD becomes irrelevant.

2. TXRDY is reset when data has been transmitted from TWI_THR to the 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.

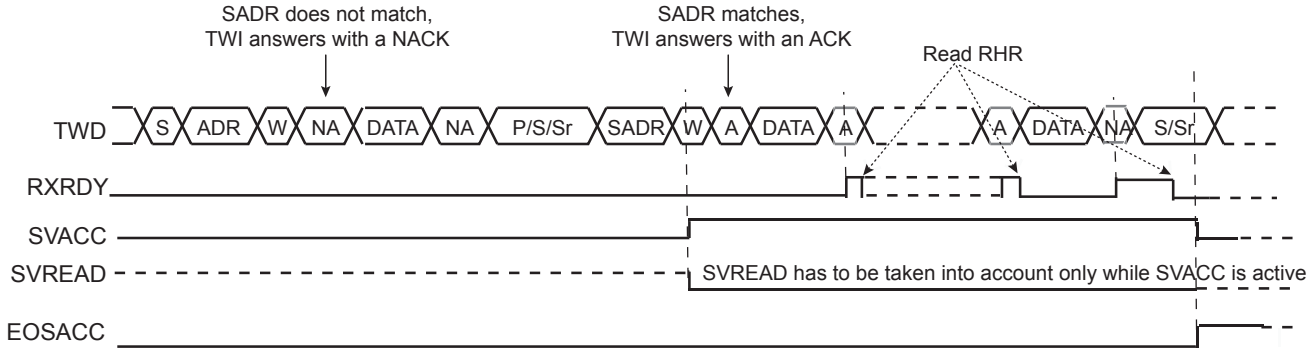
After a START or a REPEATED START, the decoding of the address starts. If the slave address is decoded, SVACC is set and SVREAD indicates the direction of the transfer (SVREAD is low in this case).

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 34-26 describes the write operation.

Figure 34-26. Write Access Ordered by a Master



Notes:
1. When SVACC is low, the state of SVREAD becomes irrelevant.
2. RXRDY is set when data has been transmitted from the 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.

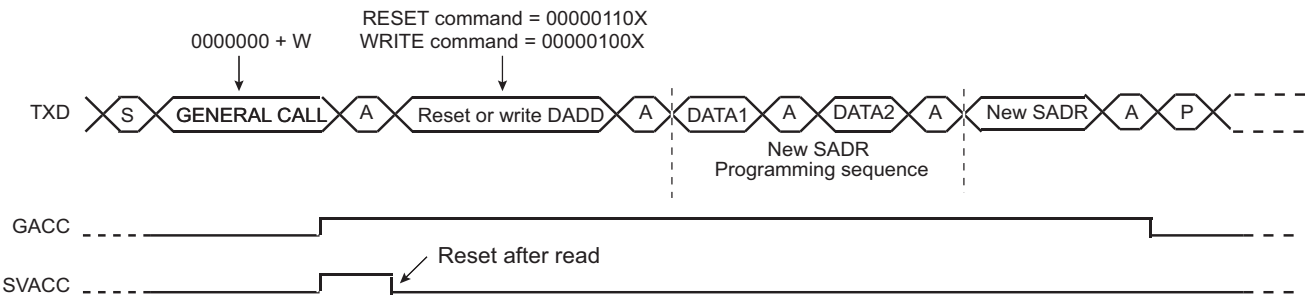
If a GENERAL CALL is detected, GACC is set.

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 34-27 describes the GENERAL CALL access.

Figure 34-27. 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

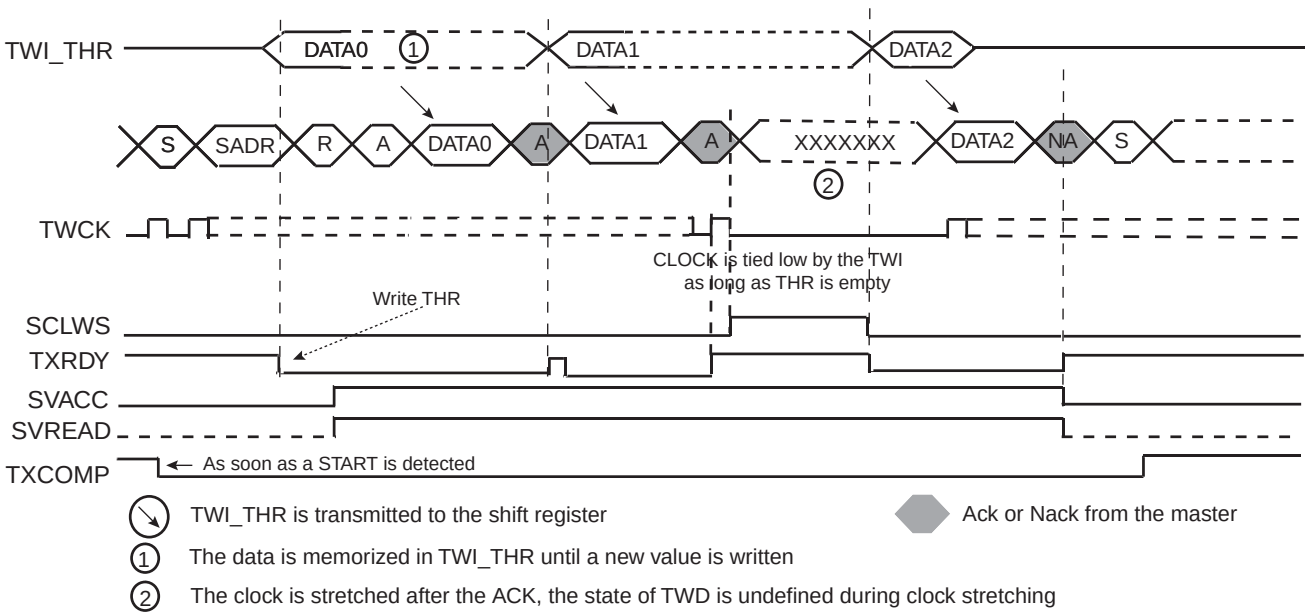
In both Read and Write modes, it may occur that TWI_THR/TWI_RHR buffer is not filled/emptied before transmission/reception of a new character. In this case, to avoid sending/receiving undesired data, a clock stretching/synchronization mechanism is implemented.

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 34-28 describes clock stretching in Read mode.

Figure 34-28. Clock Stretching in Read Mode



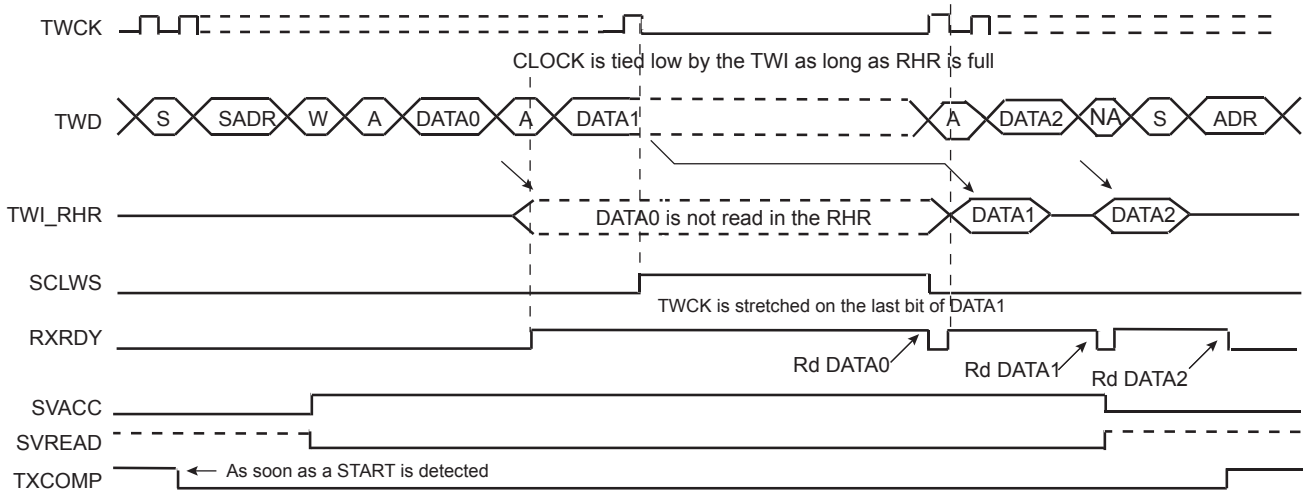
Notes:
1. TXRDY is reset when data has been written in the TWI_THR to the internal shifter and set when this data has been acknowledged or non acknowledged.
2. At the end of the read sequence, TXCOMP is set after a STOP or after a REPEATED START + an address different from SADR.
3. SCLWS is automatically 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 34-29 describes the clock synchronization in Write mode.

Figure 34-29. Clock Synchronization in Write Mode



Notes: 1. At the end of the read sequence, TXCOMP is set after a STOP or after a REPEATED START + an address different from SADR.

2. SCLWS is automatically set when the clock synchronization mechanism is started and automatically reset when the mechanism is finished.

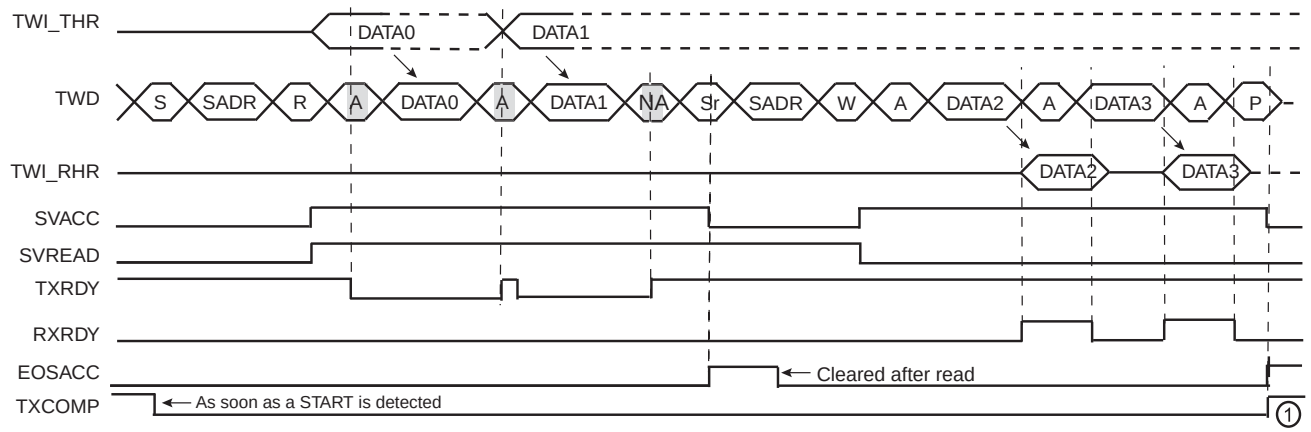
Reversal After a Repeated Start

Reversal of Read to Write

The master initiates the communication by a read command and finishes it by a write command.

Figure 34-30 describes the repeated start + reversal from Read to Write mode.

Figure 34-30. Repeated Start + Reversal from Read to Write Mode

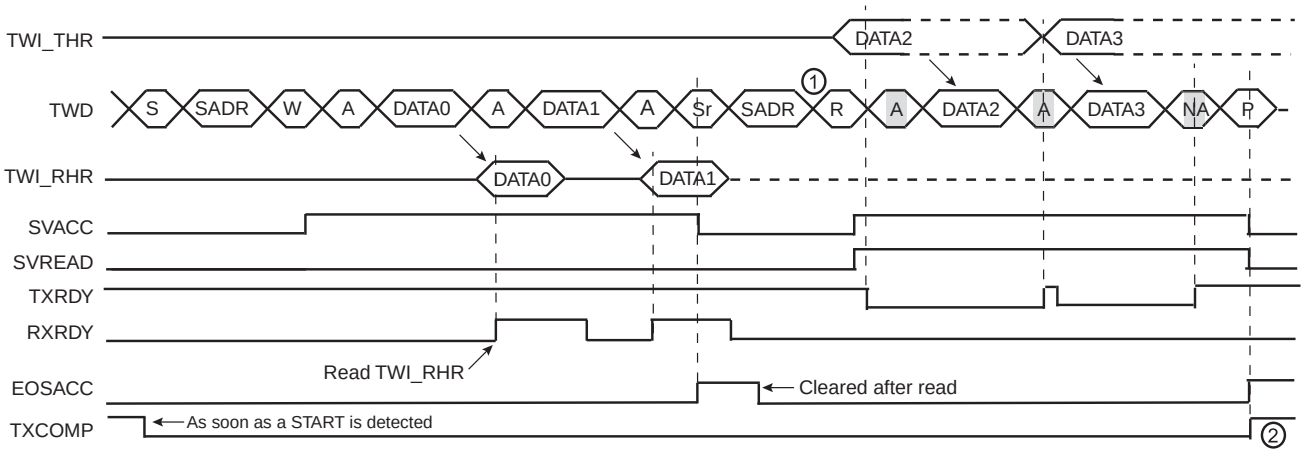


Notes: 1. TXCOMP is only set at the end of the transmission because after the repeated start, SADR is detected again.

Reversal of Write to Read

The master initiates the communication by a write command and finishes it by a read command. Figure 34-31 describes the repeated start + reversal from Write to Read mode.

Figure 34-31. Repeated Start + Reversal from Write to Read Mode



Notes:
1. In this case, if TWI_THR has not been written at the end of the read command, the clock is automatically stretched before the ACK.
2. TXCOMP is only set at the end of the transmission because after the repeated start, SADR is detected again.

34.7.5.6 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

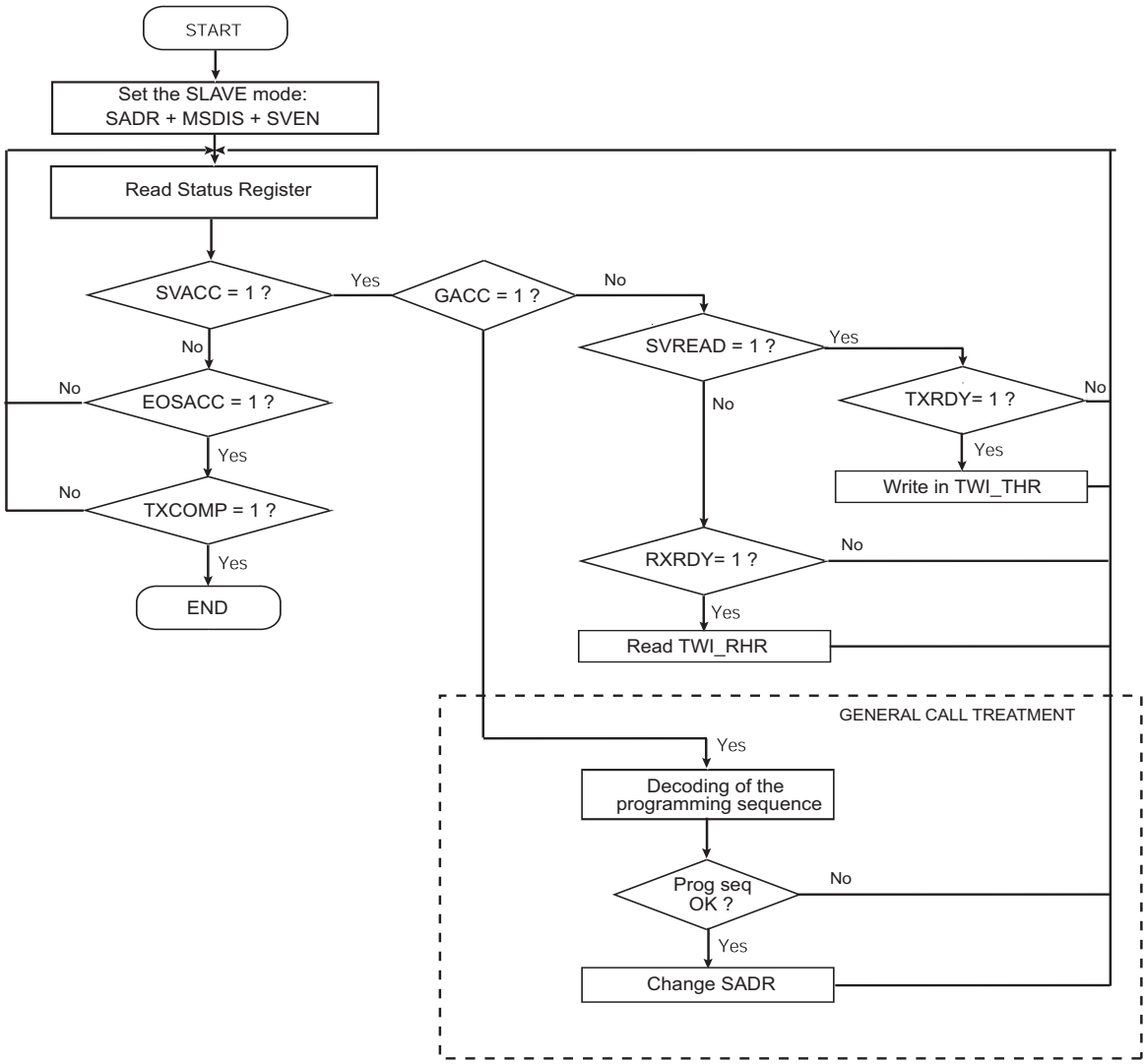
The following procedure shows an example of data transmission with PDC where the number of characters to be received is known.

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.

34.7.5.7 Read Write Flowcharts

The flowchart shown in Figure 34-32 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 34-32. Read Write Flowchart in Slave Mode



34.7.6 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"

34.8 Two-wire Interface (TWI) User Interface

Table 34-7. 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”.

34.8.1 TWI Control Register

Name: TWI_CR

Address: 0x40018000 (0), 0x4001C000 (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.

34.8.2 TWI Master Mode Register

Name: TWI_MMR

Address: 0x40018004 (0), 0x4001C004 (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.

34.8.3 TWI Slave Mode Register

Name: TWI_SMR

Address: 0x40018008 (0), 0x4001C008 (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.

34.8.4 TWI Internal Address Register

Name: TWI_IADR

Address: 0x4001800C (0), 0x4001C00C (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.

34.8.5 TWI Clock Waveform Generator Register

Name: TWI_CWGR

Address: 0x40018010 (0), 0x4001C010 (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:

$$t_{low} = (\text{CLDIV} \times 2^{\text{CKDIV}}) + 4 \times t_{peripheralclock}$$

- **CHDIV: Clock High Divider**

The TWCK high period is defined as follows:

$$t_{high} = (\text{CHDIV} \times 2^{\text{CKDIV}}) + 4 \times t_{peripheralclock}$$

- **CKDIV: Clock Divider**

The CKDIV field is used to increase both TWCK high and low periods.

34.8.6 TWI Status Register

Name: TWI_SR

Address: 0x40018020 (0), 0x4001C020 (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 34-7 and in Figure 34-9.

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 34-28, Figure 34-29, Figure 34-30 and Figure 34-31.

- **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 34-9.

RXRDY behavior in Slave mode can be seen in Figure 34-26, Figure 34-29, Figure 34-30 and Figure 34-31.

- **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 34-5, Figure 34-6 and Figure 34-7.

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 34-25, Figure 34-28, Figure 34-30 and Figure 34-31.

- **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 34-25, Figure 34-26, Figure 34-30 and Figure 34-31.

- **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 34-25, Figure 34-26, Figure 34-30 and Figure 34-31.

- **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 34-27.

- **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 34-28 and Figure 34-29.

- **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 34-30 and Figure 34-31.

- **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.

34.8.7 TWI Interrupt Enable Register

Name: TWI_IER

Address: 0x40018024 (0), 0x4001C024 (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**

34.8.8 TWI Interrupt Disable Register

Name: TWI_IDR

Address: 0x40018028 (0), 0x4001C028 (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**

34.8.9 TWI Interrupt Mask Register

Name: TWI_IMR

Address: 0x4001802C (0), 0x4001C02C (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**

34.8.10 TWI Receive Holding Register

Name: TWI_RHR

Address: 0x40018030 (0), 0x4001C030 (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**

34.8.11 TWI Transmit Holding Register

Name: TWI_THR

Address: 0x40018034 (0), 0x4001C034 (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**

34.8.12 TWI Write Protection Mode Register

Name: TWI_WPMR

Address: 0x400180E4 (0), 0x4001C0E4 (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 34.7.6 ”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.

34.8.13 TWI Write Protection Status Register

Name: TWI_WPSR

Address: 0x400180E8 (0), 0x4001C0E8 (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.

35. Universal Asynchronous Receiver Transmitter (UART)

35.1 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.

The optical link transceiver establishes electrically isolated serial communication with hand-held equipment, such as calibrators compliant with ANSI-C12.18 or IEC62056-21 norms.

35.2 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
  - Digital Filter on Receive Line
  - Interrupt Generation
  - Support for Two PDC Channels with Connection to Receiver and Transmitter
  - Optical Link Transceiver for Communication Compliant with ANSI-C12.18 or IEC62056-21 Norms

35.3 Block Diagram

Figure 35-1. UART Block Diagram

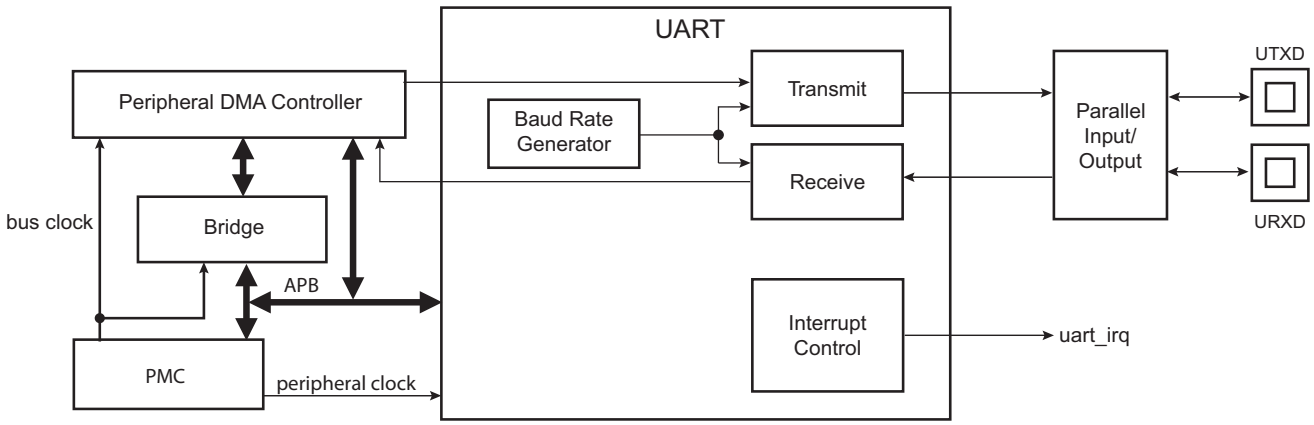


Table 35-1. UART Pin Description

Pin Name	Description	Type
URXD	UART Receive Data	Input
UTXD	UART Transmit Data	Output

35.4 Product Dependencies

35.4.1 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 35-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
UART0	URXD0	PB4	A
UART0	UTXD0	PB5	A
UART1	URXD1	PC1	A
UART1	UTXD1	PC0	A

35.4.2 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.

35.4.3 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 35-3. Peripherals IDs

Instance	ID
UART0	8
UART1	38

35.4.4 Optical Interface

The UART optical interface requires configuration of the PMC to generate 4096 kHz or 8192 kHz on the PLLA prior to any transfer.

35.5 Functional Description

The UART operates in Asynchronous mode only and supports only 8-bit character handling (with parity). It has no clock pin.

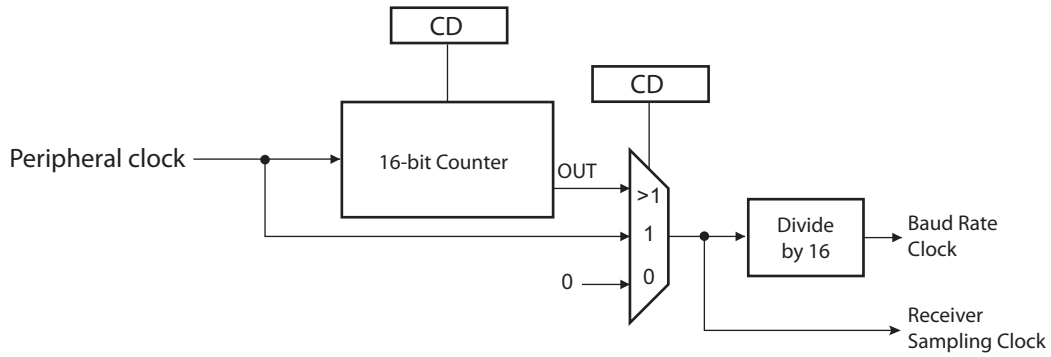
The UART is made up of a receiver and a transmitter that operate independently, and a common baud rate generator. Receiver timeout and transmitter time guard are not implemented. However, all the implemented features are compatible with those of a standard USART.

35.5.1 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 35-2. Baud Rate Generator



35.5.2 Receiver

35.5.2.1 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.

35.5.2.2 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 35-3. Start Bit Detection

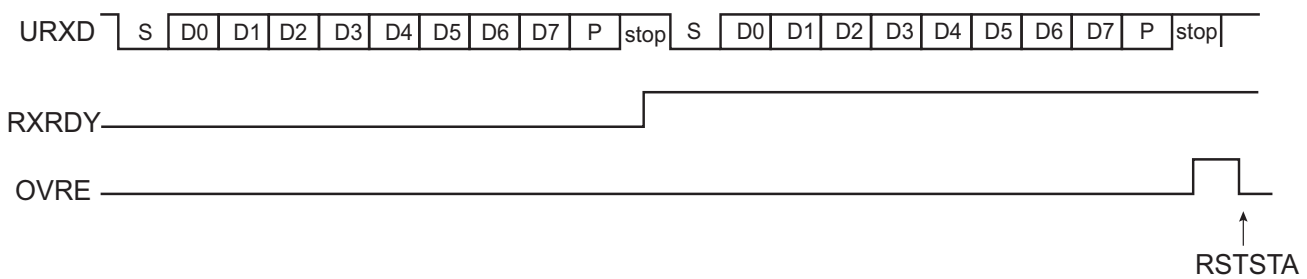
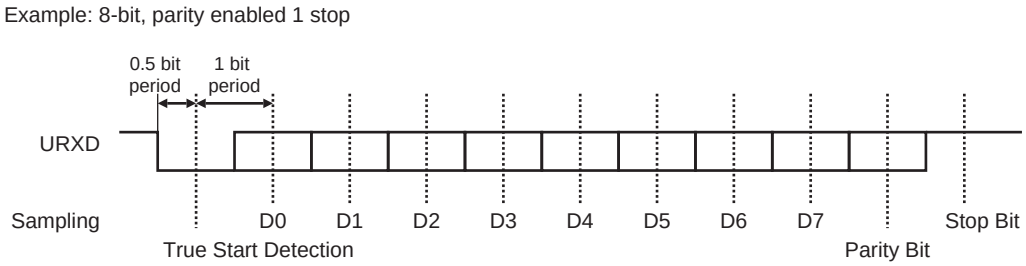


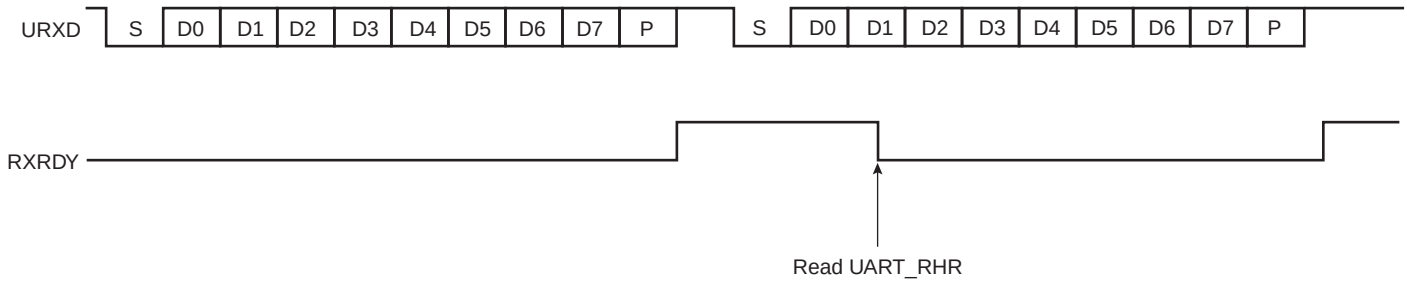
Figure 35-4. Character Reception



35.5.2.3 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.

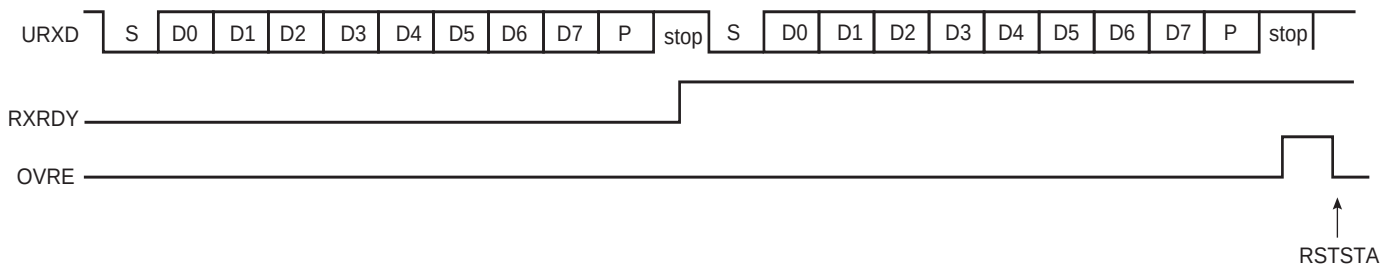
Figure 35-5. Receiver Ready



35.5.2.4 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.

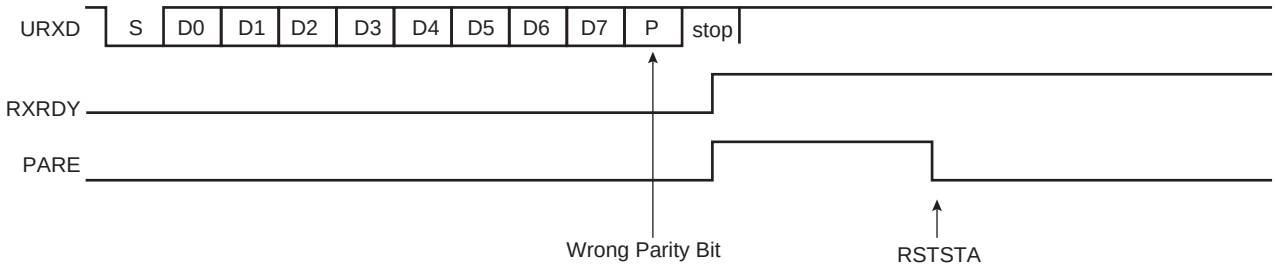
Figure 35-6. Receiver Overrun



35.5.2.5 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.

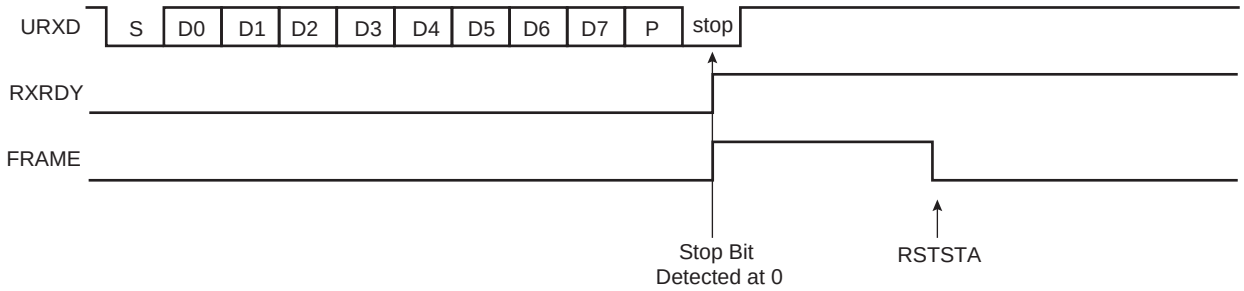
Figure 35-7. Parity Error



35.5.2.6 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 35-8. Receiver Framing Error



35.5.2.7 Receiver Digital Filter

The UART embeds a digital filter on the receive line. It is disabled by default and can be enabled by writing a logical 1 in the FILTER bit of UART_MR. When enabled, the receive line is sampled using the 16x bit clock and a three-sample filter (majority 2 over 3) determines the value of the line.

35.5.3 Transmitter

35.5.3.1 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.

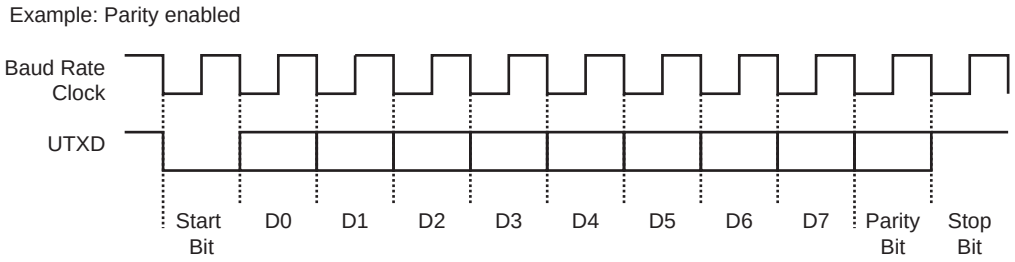
The programmer can disable the transmitter by writing UART_CR with the bit TXDIS at 1. If the transmitter is not operating, it is immediately stopped. However, if a character is being processed into the internal shift register and/or a character has been written in the UART_THR, the characters are completed 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.

35.5.3.2 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 35-9. Character Transmission

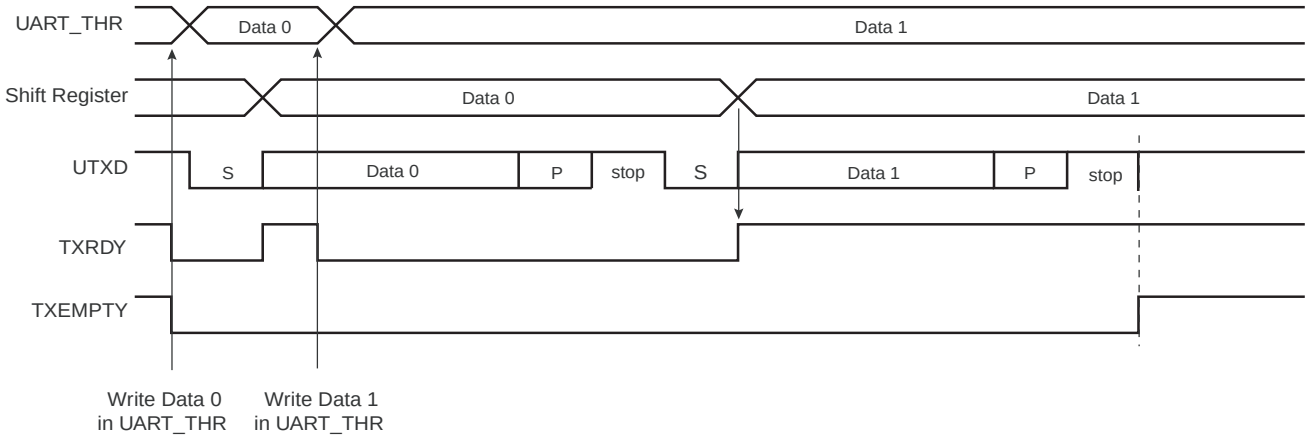


35.5.3.3 Transmitter Control

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 35-10. Transmitter Control



35.5.4 Optical Interface

To use the optical interface circuitry, the PLLA clock must be ready and programmed to generate a frequency within the range of 4096 up to 8192 kHz. This range allows a modulation by a clock with an adjustable frequency from 30 up to 60 kHz.

The optical interface is enabled by writing a 1 to the bit OPT_EN in UART_MR (see “UART Mode Register”).

When OPT_EN = 1, the URXD pad is automatically configured in Analog mode and the analog comparator is enabled (see Figure 35-11).

To match the characteristics of the off-chip optical receiver circuitry, the voltage reference threshold of the embedded comparator can be adjusted from VDDIO/10 up to VDD/2 by programming the OPT_CMPTH field in UART_MR.

The NRZ output of the UART transmitter sub-module is modulated with the 30 up to 60 kHz modulation clock prior to driving the PIO controller.

A logical 0 on the UART transmitter sub-module output generates the said modulated signal (see Figure 35-12) having a frequency programmable from 30 kHz up to 60 kHz (38 kHz is the default value assuming the PLLA clock

frequency is 8192 kHz). A logical 1 on the UART transmitter sub-module output generates a stuck-at 1 output signal (no modulation). The idle polarity of the modulated signal is 1 (OPT_MDINV = 0 in UART_MR).

The idle polarity of the modulated signal can be inverted by writing a 1 to the OPT_MDINV bit in UART_MR.

The duty cycle of the modulated signal can be adjusted from 6.25% up to 50% (default value) by steps of 6.25% by programming the OPT_DUTY field in UART_MR.

Figure 35-11. Optical Interface Block Diagram

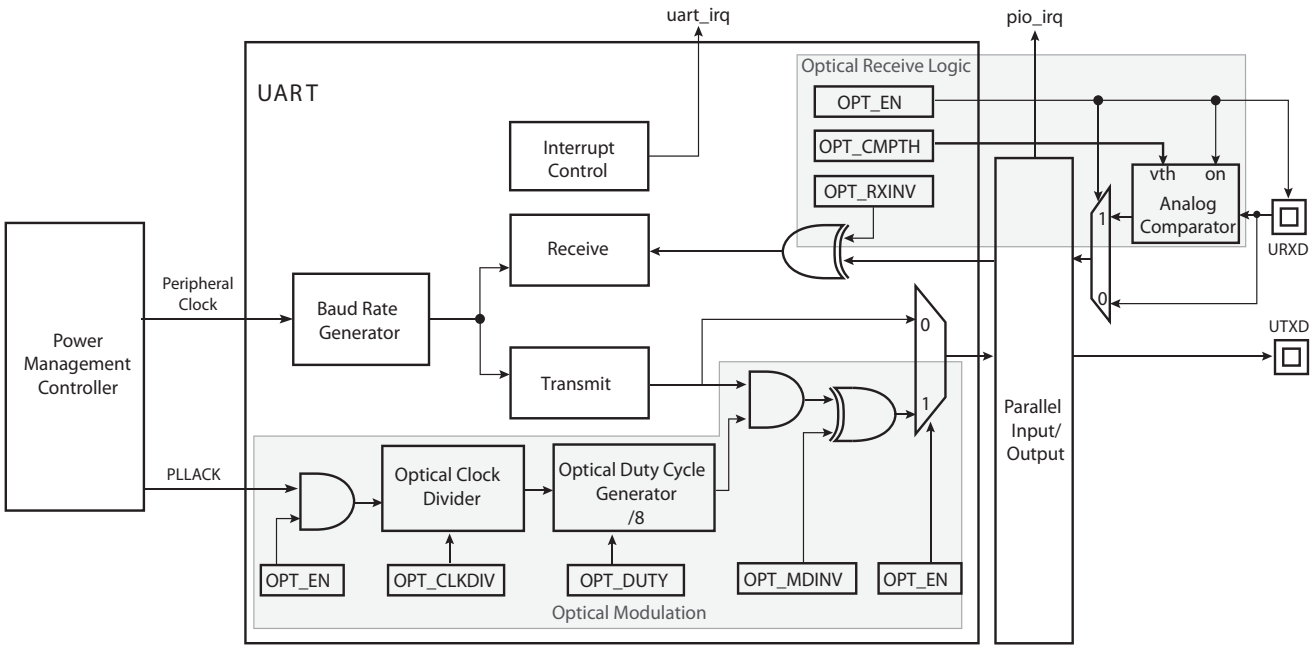
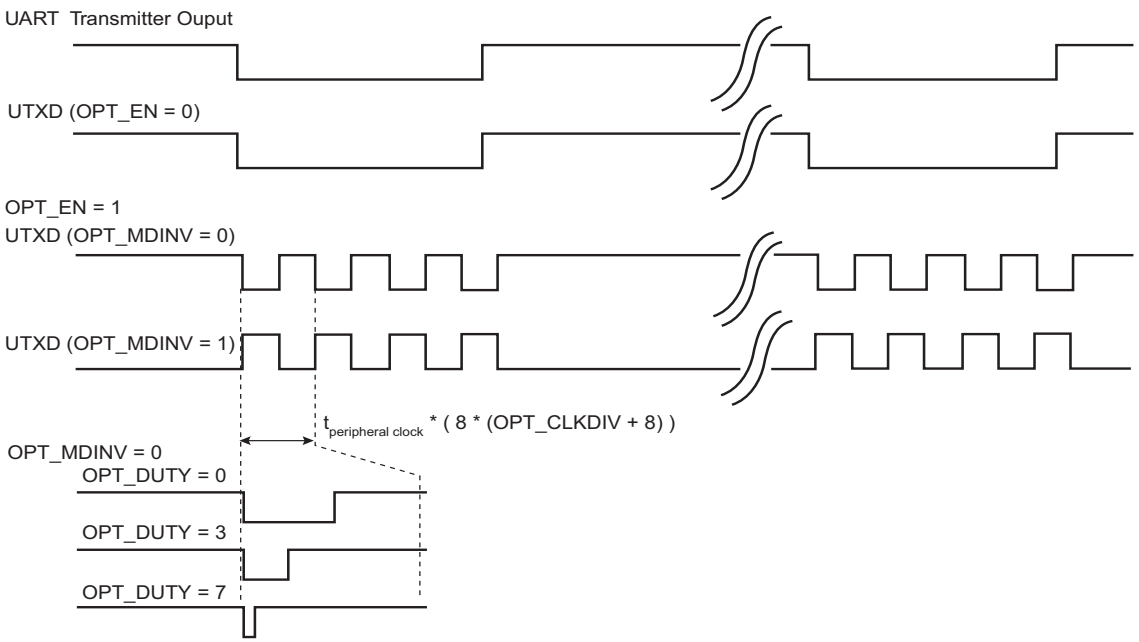


Figure 35-12. Optical Interface Waveforms



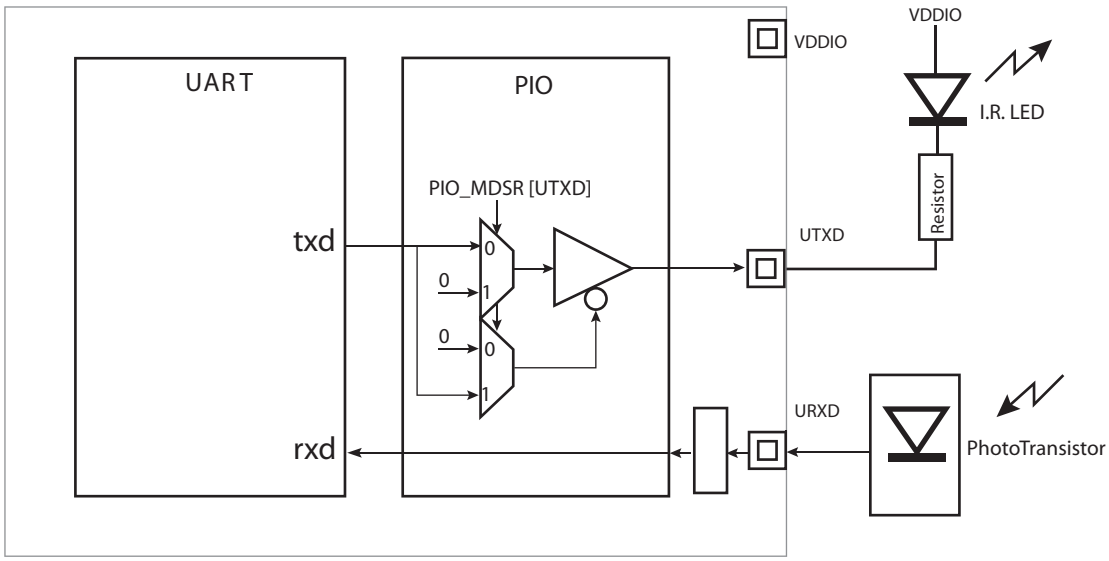
The default configuration values of the optical link circuitries allow the 38 kHz modulation, a 50% duty cycle and an idle polarity allowing a direct drive of an IR LED through a resistor (see Figure 35-13).

Refer to the section “Electrical Characteristics” for drive capability of the buffer associated with the UTXD output.

In case of direct drive of the IR LED as shown in Figure 35-13, the PIO must be programmed in Multi-driver mode (open-drain). To do so, the adequate index and values must be programmed into the PIO Multi-driver Enable Register (PIO_MDER) (status reported on the PIO Multi-driver Status Register (PIO_MDSR)). Refer to the section “Parallel Input/Output (PIO)” for details.

If an off-chip current amplifier is used to drive the transmitting of the IR LED, the PIO may be programmed in Default drive mode (non open-drain) for the line index driving the UTXD output, or in Open-drain mode depending on the type of external circuitry.

Figure 35-13. Optical Interface Connected to IR Components



35.5.5 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.

35.5.6 Test Modes

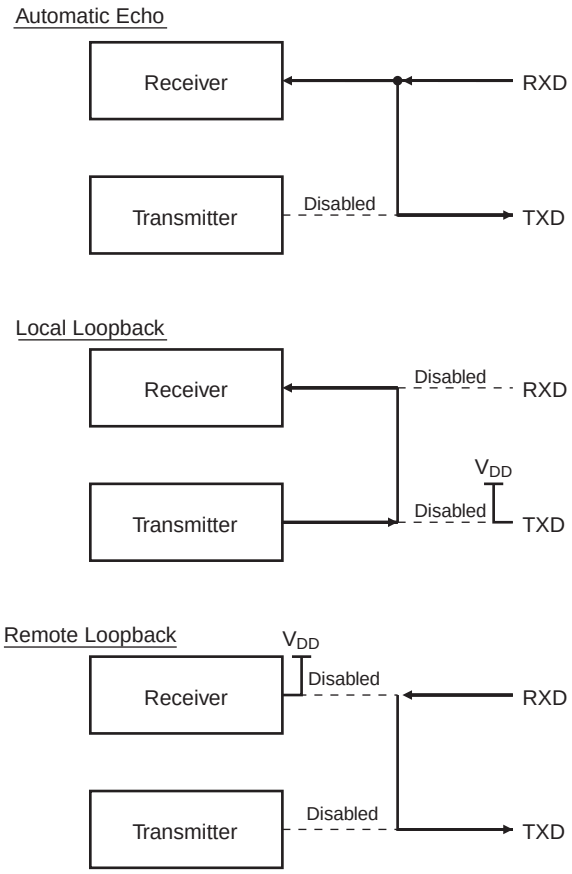
The UART supports three test modes. These modes of operation are programmed by using the CHMODE field in UART_MR.

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 35-14. Test Modes



35.6 Universal Asynchronous Receiver Transmitter (UART) User Interface

Table 35-4. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	Control Register	UART_CR	Write-only	–
0x0004	Mode Register	UART_MR	Read/Write	0x0013_0000
0x0008	Interrupt Enable Register	UART_IER	Write-only	–
0x000C	Interrupt Disable Register	UART_IDR	Write-only	–
0x0010	Interrupt Mask Register	UART_IMR	Read-only	0x0
0x0014	Status Register	UART_SR	Read-only	–
0x0018	Receive Holding Register	UART_RHR	Read-only	0x0
0x001C	Transmit Holding Register	UART_THR	Write-only	–
0x0020	Baud Rate Generator Register	UART_BRGR	Read/Write	0x0
0x0024 - 0x003C	Reserved	–	–	–
0x0040 - 0x00E8	Reserved	–	–	–
0x00EC - 0x00FC	Reserved	–	–	–
0x0100 - 0x0128	Reserved for PDC registers	–	–	–

35.6.1 UART Control Register

Name: UART_CR

Address: 0x400E0600 (0), 0x48004000 (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.

35.6.2 UART Mode Register

Name: UART_MR

Address: 0x400E0604 (0), 0x48004004 (1)

Access: Read/Write

31	30	29	28	27	26	25	24
–	OPT_CMPTH			–	OPT_DUTY		

23	22	21	20	19	18	17	16
–	–	–	OPT_CLKDIV				

15	14	13	12	11	10	9	8
CHMODE		–	–	PAR			–

7	6	5	4	3	2	1	0
–	–	–	FILTER	–	OPT_MDINV	OPT_RXINV	OPT_EN

- **OPT_EN: UART Optical Interface Enable**

Value	Name	Description
0	DISABLED	The UART transmitter data is not inverted before modulation.
1	ENABLED	The UART transmitter data is inverted before modulation.

- **OPT_RXINV: UART Receive Data Inverted**

Value	Name	Description
0	DISABLED	The comparator data output is not inverted before entering UART.
1	ENABLED	The comparator data output is inverted before entering UART.

- **OPT_MDINV: UART Modulated Data Inverted**

Value	Name	Description
0	DISABLED	The output of the modulator is not inverted.
1	ENABLED	The output of the modulator is inverted.

- **FILTER: Receiver Digital Filter**

0 (DISABLED): UART does not filter the receive line.

1 (ENABLED): UART filters the receive line using a three-sample filter (16x-bit clock) (2 over 3 majority).

- **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

- **OPT_CLKDIV: Optical Link Clock Divider**

0 - 31: The optical modulation clock frequency is defined by PLLACK / (8 * (OPT_CLKDIV + 8)).

- **OPT_DUTY: Optical Link Modulation Clock Duty Cycle**

Value	Name	Description
0	DUTY_50	Modulation clock duty cycle is 50%.
1	DUTY_43P75	Modulation clock duty cycle is 43.75%.
2	DUTY_37P5	Modulation clock duty cycle is 37.5%.
3	DUTY_31P25	Modulation clock duty cycle is 31.75%.
4	DUTY_25	Modulation clock duty cycle is 25%.
5	DUTY_18P75	Modulation clock duty cycle is 18.75%.
6	DUTY_12P5	Modulation clock duty cycle is 12.5%.
7	DUTY_6P25	Modulation clock duty cycle is 6.25%.

- **OPT_CMPTH: Receive Path Comparator Threshold**

Value	Name	Description
0	VDDIO_DIV2	Comparator threshold is VDDIO/2 volts.
1	VDDIO_DIV2P5	Comparator threshold is VDDIO/2.5 volts.
2	VDDIO_DIV3P3	Comparator threshold is VDDIO/3.3 volts.
3	VDDIO_DIV5	Comparator threshold is VDDIO/5 volts.
4	VDDIO_DIV10	Comparator threshold is VDDIO/10 volts.

35.6.3 UART Interrupt Enable Register

Name: UART_IER

Address: 0x400E0608 (0), 0x48004008 (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 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**

35.6.4 UART Interrupt Disable Register

Name: UART_IDR

Address: 0x400E060C (0), 0x4800400C (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 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**

35.6.5 UART Interrupt Mask Register

Name: UART_IMR

Address: 0x400E0610 (0), 0x48004010 (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: Mask TXRDY Interrupt**
- **ENDRX: Mask End of Receive 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**

35.6.6 UART Status Register

Name: UART_SR

Address: 0x400E0614 (0), 0x48004014 (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.

35.6.7 UART Receiver Holding Register

Name: UART_RHR

Address: 0x400E0618 (0), 0x48004018 (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.

35.6.8 UART Transmit Holding Register

Name: UART_THR

Address: 0x400E061C (0), 0x4800401C (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.

35.6.9 UART Baud Rate Generator Register

Name: UART_BRGR

Address: 0x400E0620 (0), 0x48004020 (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:

$$CD = \frac{f_{peripheral\ clock}}{16 \times Baud\ Rate}$$

36. Universal Synchronous Asynchronous Receiver Transmitter (USART)

36.1 Description

The Universal Synchronous Asynchronous Receiver Transmitter (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 and infrared transceivers. The hardware handshaking feature enables an out-of-band flow control by automatic management of the pins RTS and CTS.

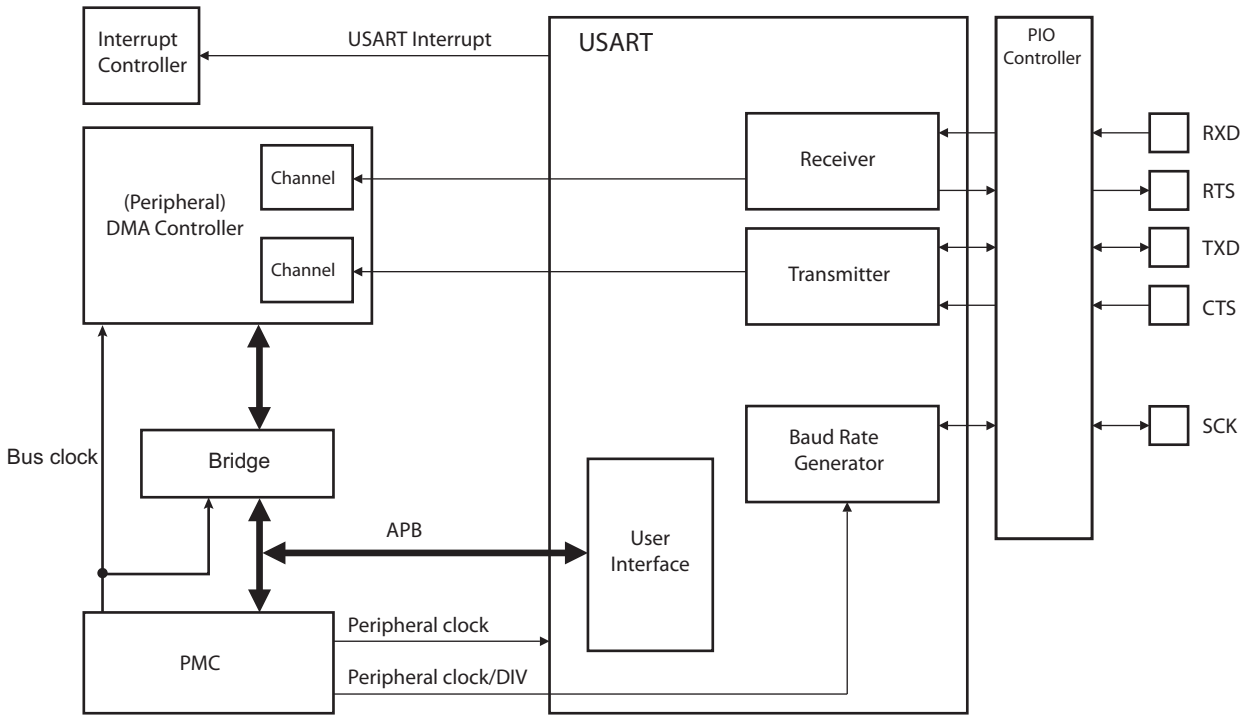
The USART supports the connection to the Peripheral DMA Controller, which enables data transfers to the transmitter and from the receiver. The PDC provides chained buffer management without any intervention of the processor.

36.2 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
  - 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 Peripheral DMA Controller Channels (PDC)
- Offers Buffer Transfer without Processor Intervention
- Register Write Protection

36.3 Block Diagram

Figure 36-1. USART Block Diagram



36.4 I/O Lines Description

Table 36-1. 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	--
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

36.5 Product Dependencies

36.5.1 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.

Table 36-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
USART0	CTS0	PA20	A
USART0	RTS0	PA19	A
USART0	RXD0	PB16	A
USART0	SCK0	PB18	A
USART0	TXD0	PB17	A
USART1	CTS1	PA18	A
USART1	RTS1	PA17	A
USART1	RXD1	PA11	A
USART1	SCK1	PA16	A
USART1	TXD1	PA12	A
USART2	CTS2	PA15	A
USART2	RTS2	PA14	A
USART2	RXD2	PA9	A
USART2	SCK2	PA13	A
USART2	TXD2	PA10	A
USART3	CTS3	PA1	A
USART3	RTS3	PA0	A
USART3	RXD3	PA3	A
USART3	SCK3	PA2	A
USART3	TXD3	PA4	A
USART4	CTS4	PA26	A
USART4	RTS4	PB22	A
USART4	RXD4	PB19	A
USART4	SCK4	PB21	A
USART4	TXD4	PB20	A

36.5.2 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.

36.5.3 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 36-3. Peripheral IDs

Instance	ID
USART0	14
USART1	15
USART2	16
USART3	17
USART4	18

36.6 Functional Description

36.6.1 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.

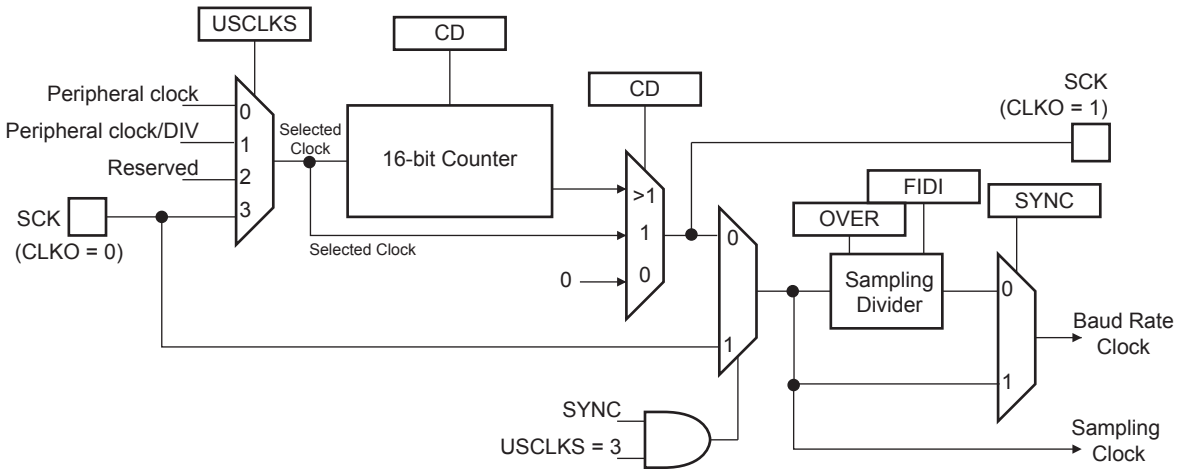
The baud rate generator clock source is selected by configuring the USCLKS field in the USART Mode Register (US_MR) to one of the following:

- 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 configured using the CD field of the Baud Rate Generator Register (US_BRGR). If CD is configured to ‘0’, the baud rate generator does not generate any clocks. If CD is configured to ‘1’, the divider is bypassed and becomes inactive.

If the external SCK clock is selected, the duration of the low and high levels of the signal provided on the SCK pin must be longer than a 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 36-2. Baud Rate Generator



36.6.1.1 Baud Rate in Asynchronous Mode

If the USART is programmed to operate in Asynchronous mode, the selected clock is first divided by the value of US_BRGR.CD. The resulting clock is provided to the receiver as a sampling clock and then divided by 16 or 8, depending on the value of US_MR.OVER.

If OVER is set to '1', the receiver sampling is eight times higher than the baud rate clock. If OVER is set to '0', the sampling is performed at 16 times the baud rate clock.

The baud rate is calculated as per the following formula:

$$Baudrate = \frac{SelectedClock}{(8(2 - Over)CD)}$$

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 is written to '1'.

Baud Rate Calculation Example

Table 36-4 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 36-4. Baud Rate Example (OVER = 0)

Source Clock (Hz)	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:

Baud Rate = Selected Clock / CD x 16

The baud rate error is calculated with the following formula. It is not recommended to work with an error higher than 5%.

$$Error = 1 - \left(\frac{ExpectedBaudRate}{ActualBaudRate}\right)$$

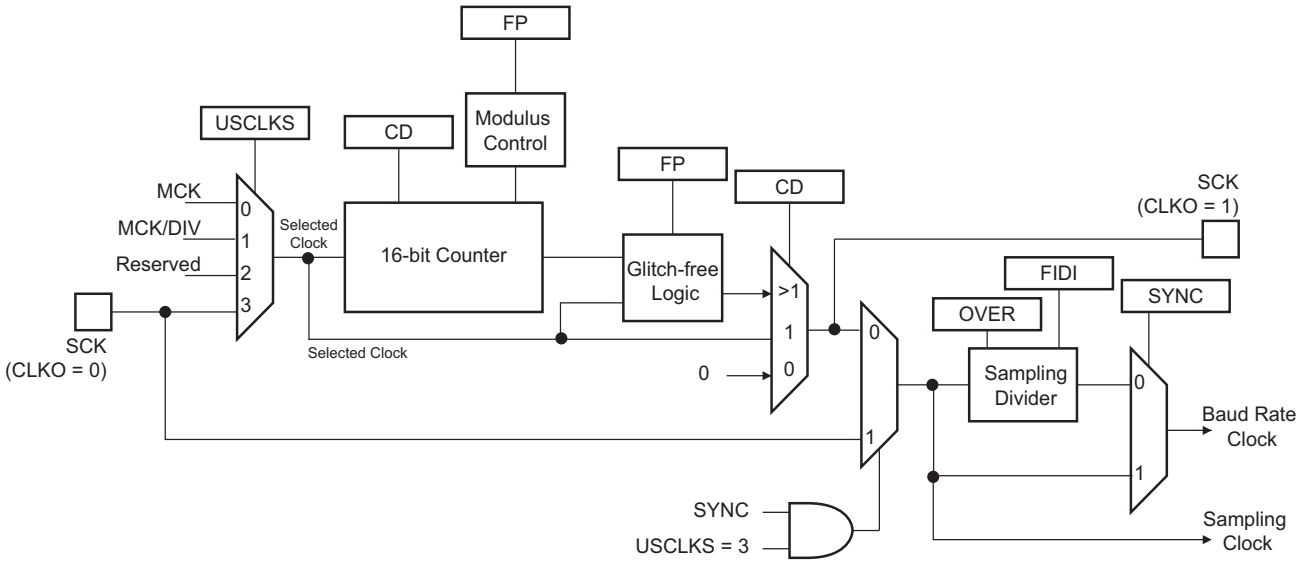
36.6.1.2 Fractional Baud Rate in Asynchronous Mode

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 using US_BRGR.FP. If FP is not 0, the fractional part is activated. The resolution is one-eighth of the clock divider. The fractional baud rate is calculated using the following formula:

$$Baudrate = \frac{SelectedClock}{\left(8(2 - Over)\left(CD + \frac{FP}{8}\right)\right)}$$

The modified architecture is presented in the Figure 36-3.

Figure 36-3. Fractional Baud Rate Generator



Warning: When the value of US_BRGR.FP is greater than 0, the SCK (oversampling clock) generates nonconstant duty cycles. The SCK high duration is increased by “selected clock” period from time to time. The duty cycle depends on the value of US_BRGR.CD.

36.6.1.3 Baud Rate in Synchronous Mode or SPI Mode

If the USART is programmed to operate in Synchronous mode, the selected clock is divided by the value of US_BRGR.CD.

$$BaudRate = \frac{SelectedClock}{CD}$$

In Synchronous mode, if the external clock is selected (USCLKS = 3), the clock is provided directly by the signal on the USART SCK pin. No division is active. The value written in US_BRGR has no effect. The external clock frequency must be at least 3 times lower than the system clock. In 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 of 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 of CD is odd.

36.6.1.4 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 36-5.

Table 36-5. 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 36-6.

Table 36-6. 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 36-7 shows the resulting Fi/Di ratio, which is the ratio between the ISO7816 clock and the baud rate clock.

Table 36-7. 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

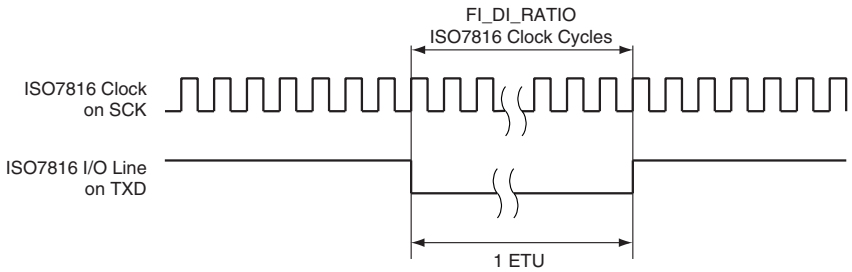
If the USART is configured in ISO7816 mode, the clock selected by US_MR.USCLKS is first divided by the value programmed in US_BRGR.CD. The resulting clock can be provided to the SCK pin to feed the smart card clock inputs. This means that the US_MR.CLKO bit can be written to ‘1’.

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 noninteger values of the Fi/Di ratio are not supported and the user must program FI_DI_RATIO to a value as close as possible to the expected value.

FI_DI_RATIO 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 36-4 shows the relation between the Elementary Time Unit, corresponding to a bit time, and the ISO 7816 clock.

Figure 36-4. Elementary Time Unit (ETU)



36.6.2 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 writing a '1' to US_CR.TXEN. 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 writing a '1' to the corresponding bit US_CR.RSTRX and US_CR.RSTTX respectively. 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 writing a '1' to US_CR.RXDIS and US_CR.TXDIS, respectively. 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.

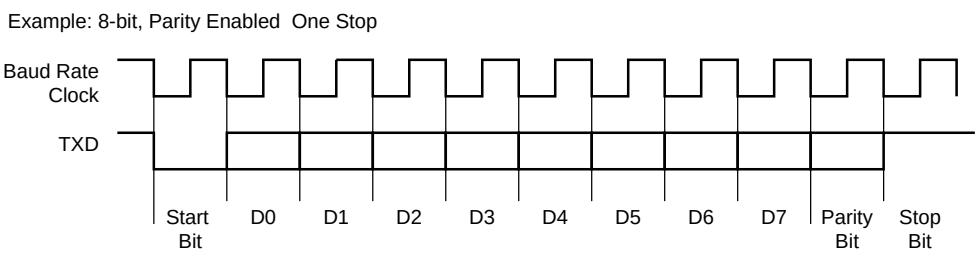
36.6.3 Synchronous and Asynchronous Modes

36.6.3.1 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 configured in the US_MR.CHRL and the US_MR.MODE9. Nine bits are selected by writing a '1' to US_MR.MODE9 regardless of the CHRL field. The parity is selected by US_MR.PAR. Even, odd, space, marked or none parity bit can be configured. US_MR.MSBF 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 US_MR.NBSTOP. The 1.5 stop bit is supported in Asynchronous mode only.

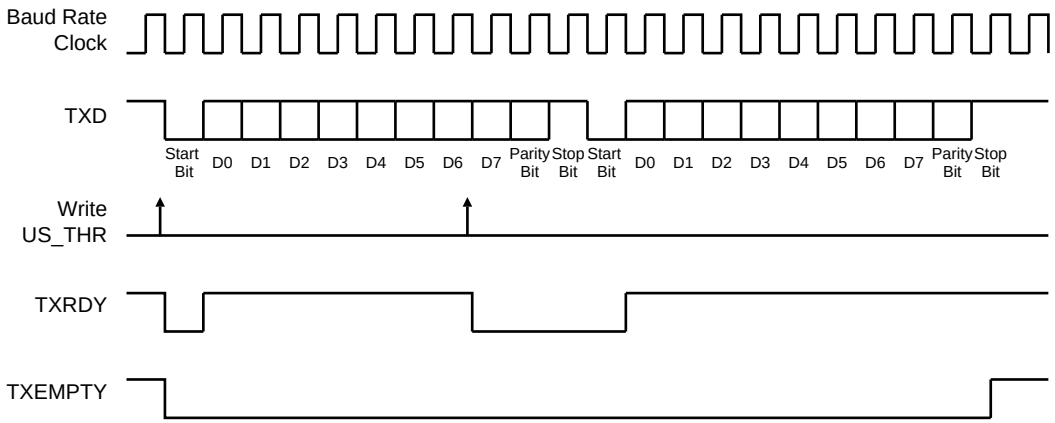
Figure 36-5. Character Transmit



The characters are sent by writing in the 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 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.

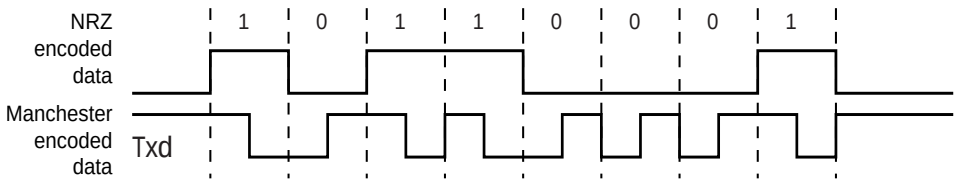
Figure 36-6. Transmitter Status



36.6.3.2 Manchester Encoder

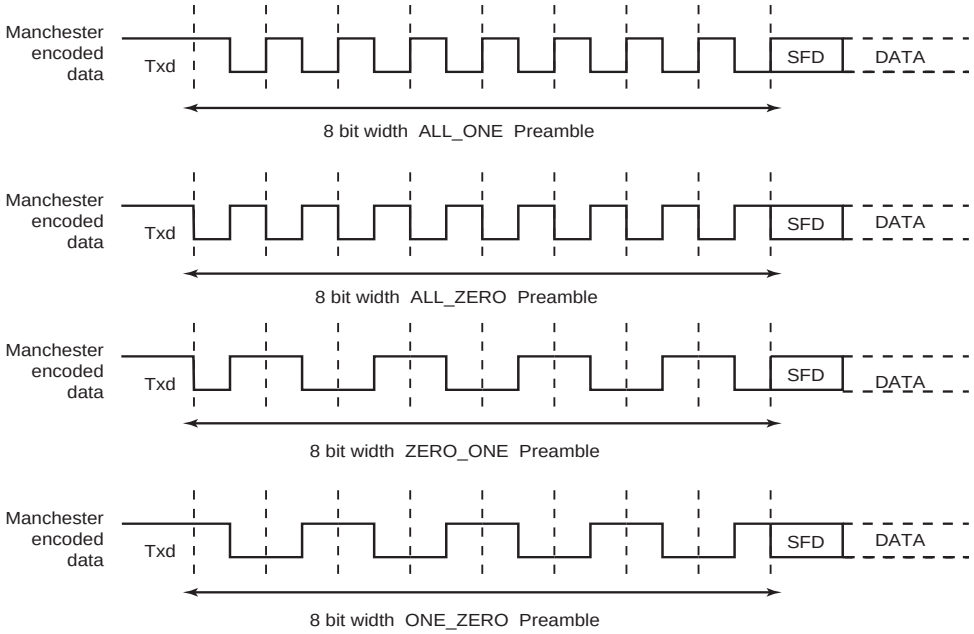
When the Manchester encoder is in use, characters transmitted through the USART are encoded based on biphase Manchester II format. To enable this mode, write a '1' to USART_MR.MAN. Depending on polarity configuration, a logic level (zero or one), is transmitted as a coded signal one-to-zero or zero-to-one. 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 36-7 illustrates this coding scheme.

Figure 36-7. NRZ to Manchester Encoding



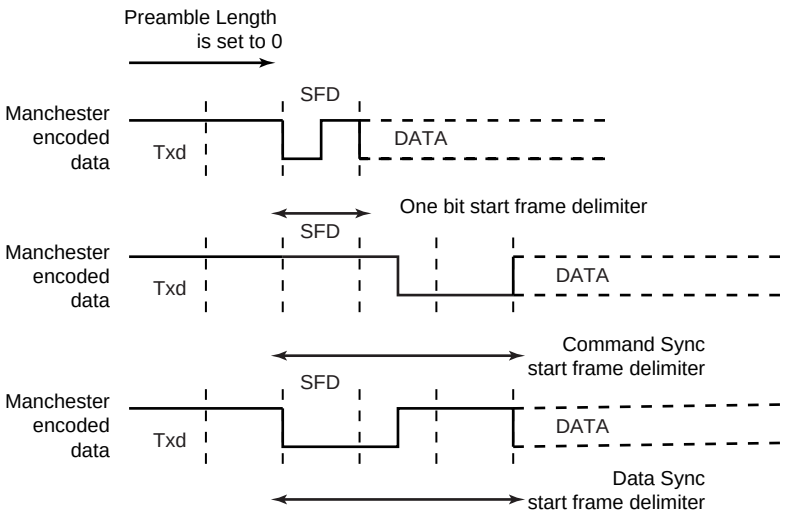
The Manchester encoded character can also be encapsulated by adding both a configurable preamble and a start frame delimiter pattern. Depending on the configuration, the preamble is a training sequence, composed of a predefined pattern with a programmable length from 1 to 15 bit times. If the preamble 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 by configuring US_MAN.TX_PP. US_MAN.TX_PL is used to configure the preamble length. Figure 36-8 illustrates and defines the valid patterns. To improve flexibility, the encoding scheme can be configured using US_MAN.TX_MPOL. If TX_MPOL is set to '0' (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 TX_MPOL 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 36-8. Preamble Patterns, Default Polarity Assumed



A start frame delimiter is configured using US_MR.ONEBIT. It consists of a user-defined pattern that indicates the beginning of a valid data. Figure 36-9 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 = 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 US_MR.MODSYNC is written to '1', the next character is a command. If it is written to '0', the next character is a data. When direct memory access is used, MODSYNC can be immediately updated with a modified character located in memory. To enable this mode, US_MR.VAR_SYNC must be written to '1'. In this case, MODSYNC is bypassed and the sync configuration is held in US_THR.TXSYNH. The USART character format is modified and includes sync information.

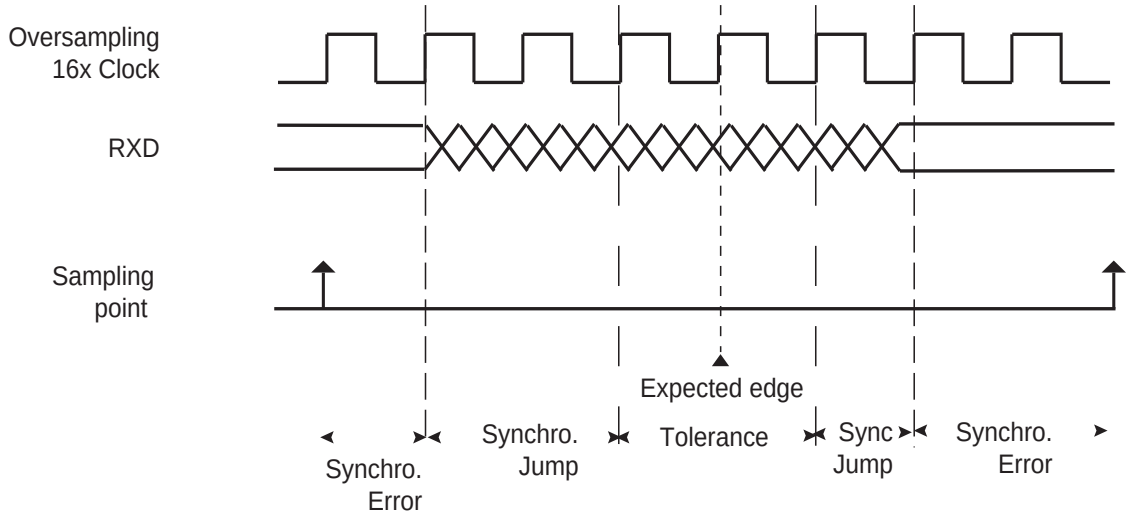
Figure 36-9. Start Frame Delimiter



Drift Compensation

Drift compensation is available only in 16X oversampling mode. A hardware recovery system allows a larger clock drift. To enable the hardware system, USART_MAN.DRIFT must be written to ‘1’. If the RXD edge is one 16X clock cycle from the expected edge, this is considered as normal jitter and no corrective action 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 36-10. Bit Resynchronization



36.6.3.3 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 value of US_MR.OVER.

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 36-11 and Figure 36-12 illustrate start detection and character reception when USART operates in Asynchronous mode.

Figure 36-11. Asynchronous Start Detection

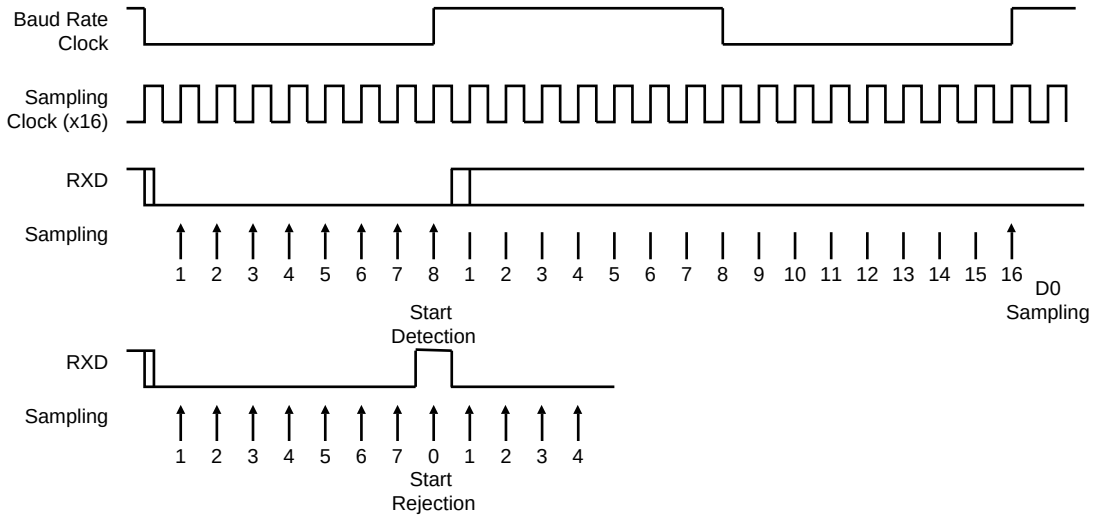
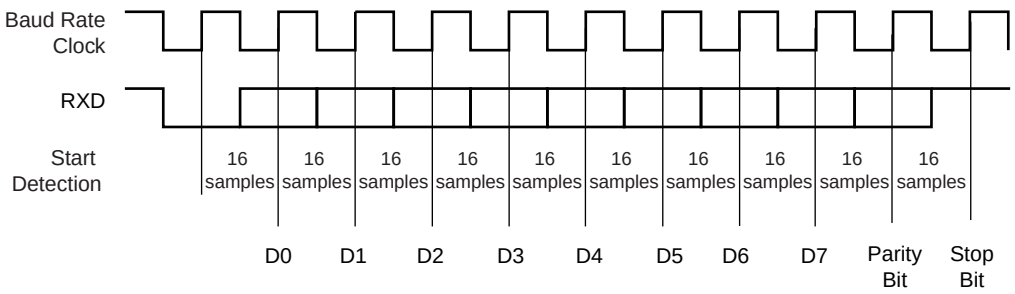


Figure 36-12. Asynchronous Character Reception

Example: 8-bit, Parity Enabled



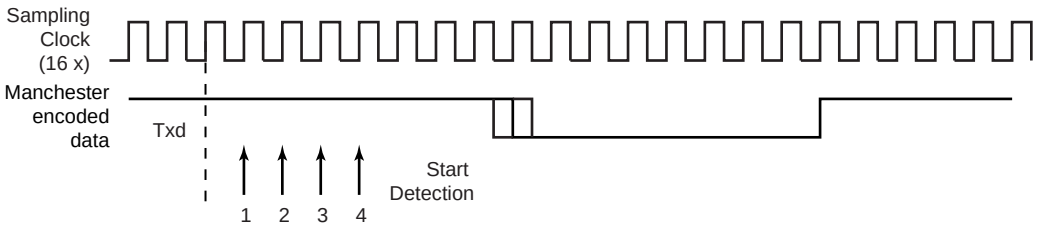
36.6.3.4 Manchester Decoder

When US_MR.MAN is ‘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, and its length is user-defined and totally independent of the emitter side. The length of the preamble sequence is configured using US_MAN.RX_PL. If RX_PL is ‘0’, no preamble is detected and the function is disabled. The polarity of the input stream is configured with US_MAN.RX_MPOL. Depending on the desired application, the preamble pattern matching is to be defined via the US_MAN. See Figure 36-8 for available preamble patterns.

Unlike preamble, the start frame delimiter is shared between Manchester Encoder and Decoder. If US_MR.ONEBIT is written to ‘1’, only a zero encoded Manchester can be detected as a valid start frame delimiter. If US_MR.ONEBIT is written 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 36-13. The sample pulse rejection mechanism applies.

Figure 36-13. 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 the USART for processing. Figure 36-14 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 US_CSR.MANERR flag is raised. It is cleared by writing a '1' to US_CR.RSTSTA. See Figure 36-15 for an example of Manchester error detection during data phase.

Figure 36-14. Preamble Pattern Mismatch

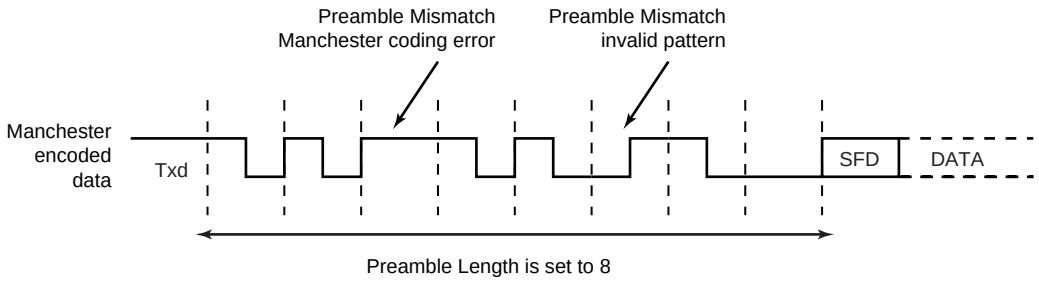
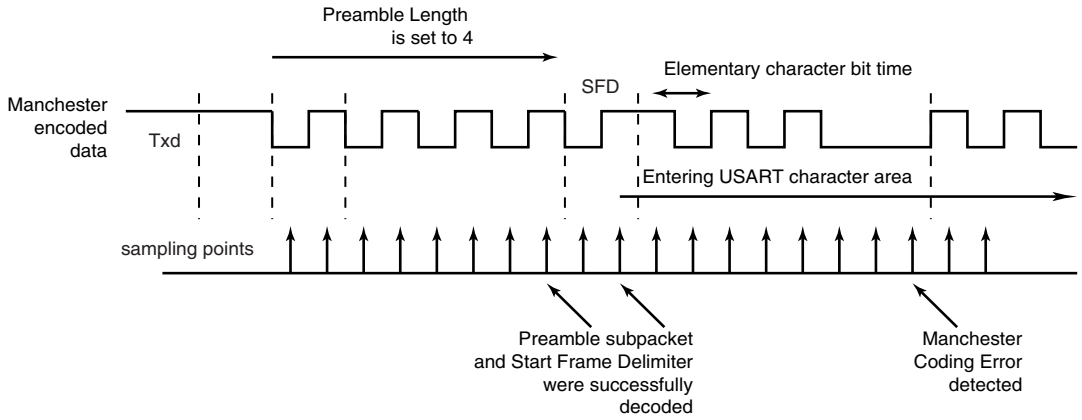


Figure 36-15. Manchester Error Flag



When the start frame delimiter is a sync pattern (US_MR.ONEBIT = 0), both command and data delimiter are supported. If a valid sync is detected, the received character is written in RXCHR in the Receive Holding register (US_RHR) and RXSYNH is updated. RXSYNH is set to '1' when the received character is a command, and to '0' if the received character is a data. This alleviates and simplifies the direct memory access as the character contains its own sync field in the same register.

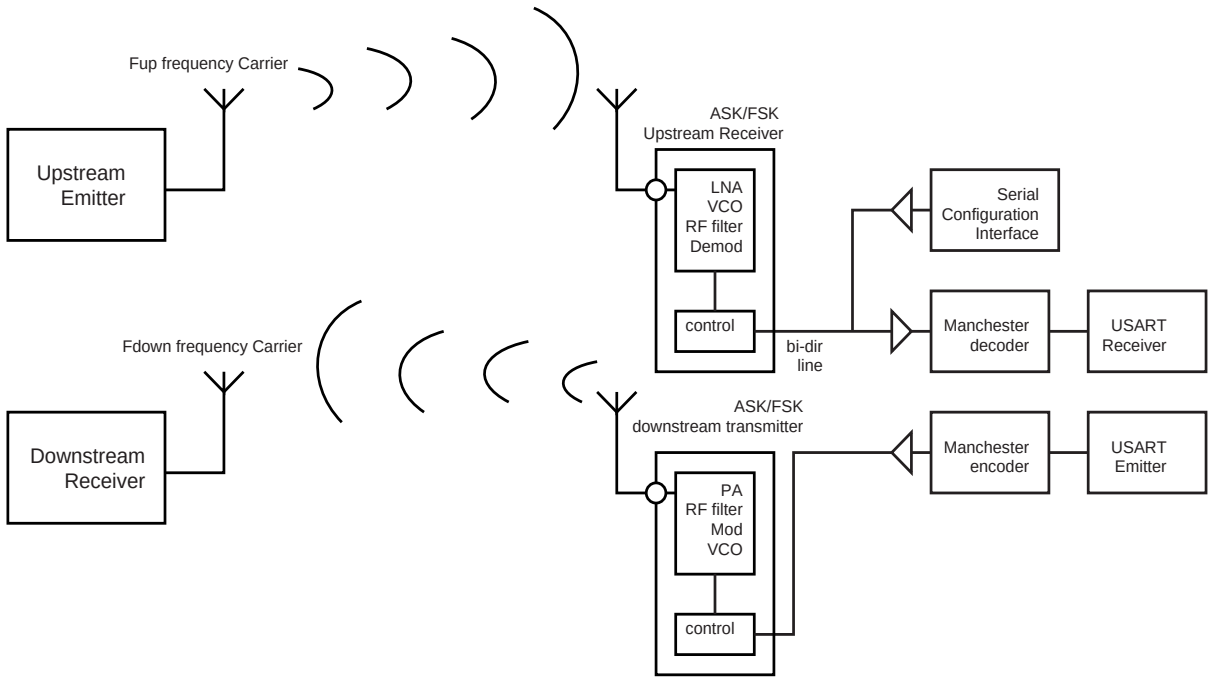
As the decoder is setup to be used in Unipolar mode, the first bit of the frame has to be a zero-to-one transition.

36.6.3.5 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 36-16.

Figure 36-16. 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 36-17 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 one is sent, the modulator outputs an RF signal at frequency F0 and switches to F1 if the data sent is a zero. See Figure 36-18.

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 36-17. ASK Modulator Output

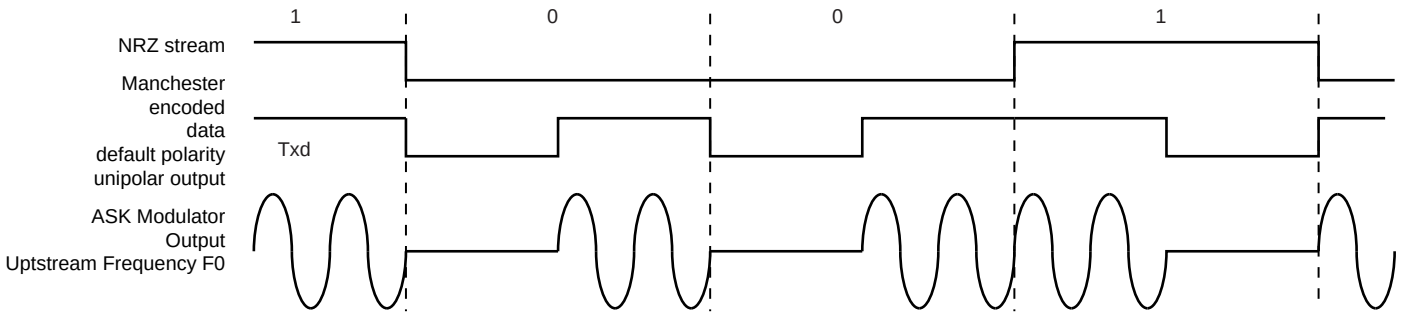
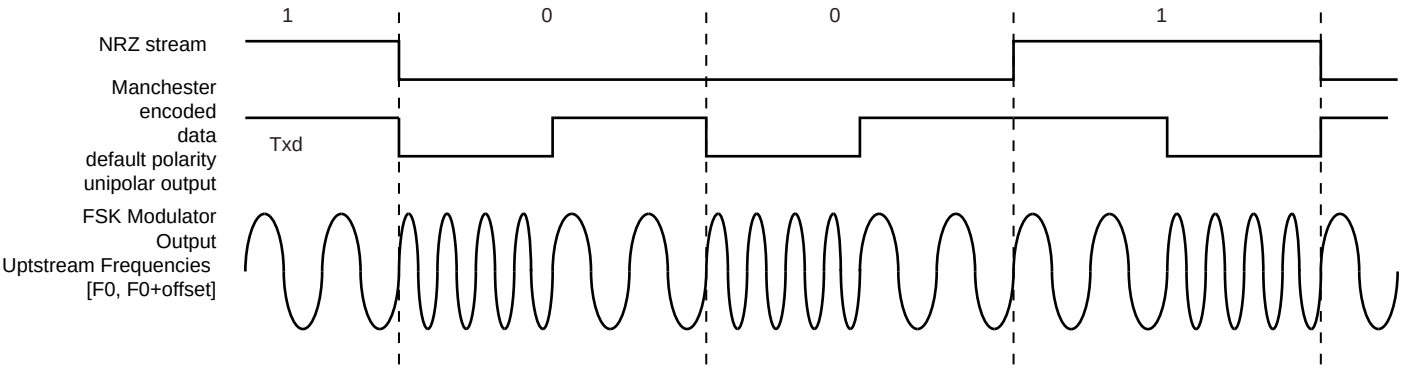


Figure 36-18. FSK Modulator Output



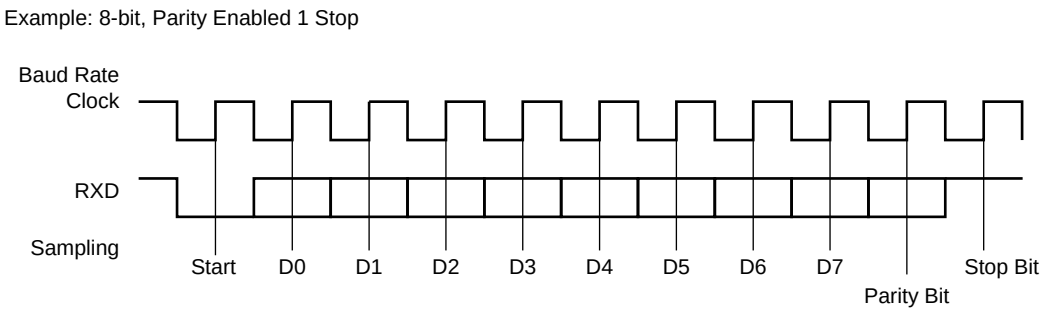
36.6.3.6 Synchronous Receiver

In Synchronous mode (US_MR.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 36-19 illustrates a character reception in Synchronous mode.

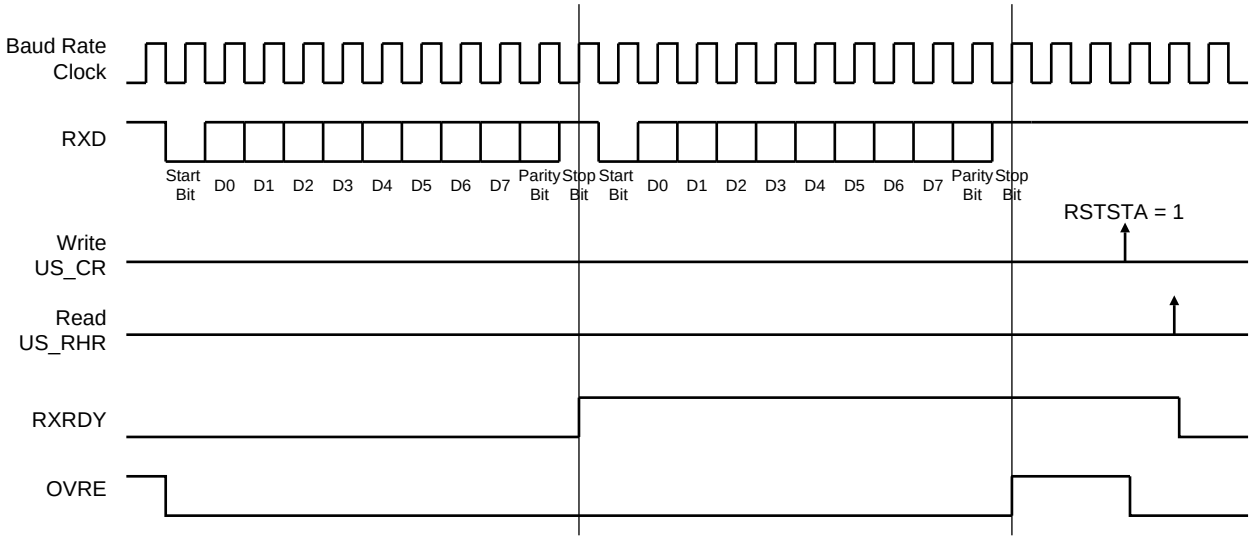
Figure 36-19. Synchronous Mode Character Reception



36.6.3.7 Receiver Operations

When a character reception is completed, it is transferred to the Receive Holding Register (US_RHR) and US_CSR.RXRDY 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 US_CR.RSTSTA.

Figure 36-20. Receiver Status



36.6.3.8 Parity

The USART supports five Parity modes. The PAR field also enables Multidrop mode, see Section 36.6.3.9 ”Multidrop Mode”. Even and odd parity bit generation and error detection are supported. The configuration is done in US_MR.PAR.

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.

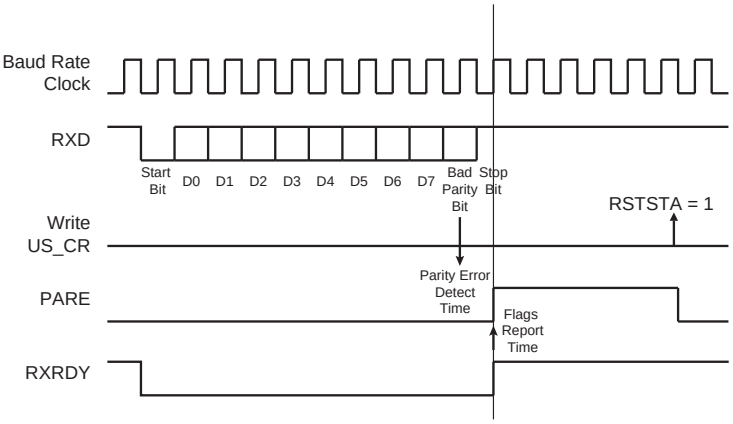
Table 36-8 shows an example of the parity bit for the character 0x41 (character ASCII “A”) depending on the configuration of the USART. Because there are two bits set to 1 in the character value, the parity bit is set to ‘1’ when the parity is odd, or configured to ‘0’ when the parity is even.

Table 36-8. 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 US_CSR.PARE (Parity Error). PARE can be cleared by writing a ‘1’ to the RSTSTA bit in the US_CR. Figure 36-21 illustrates the parity bit status setting and clearing.

Figure 36-21. Parity Error



36.6.3.9 Multidrop Mode

If the value 0x6 or 0x07 is written to US_MR.PAR, 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 PARE 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 US_CR.SENDA.

To handle parity error, PARE is cleared when a '1' is written to US_CR.RSTSTA.

The transmitter sends an address byte (parity bit set) when US_CR.SENDA = 1. In this case, the next byte written to US_THR is transmitted as an address. Any character written in the US_THR without having written SENDA is transmitted normally with the parity at 0.

36.6.3.10 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 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 '0', 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 36-22, 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 at 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 36-22. Timeguard Operations

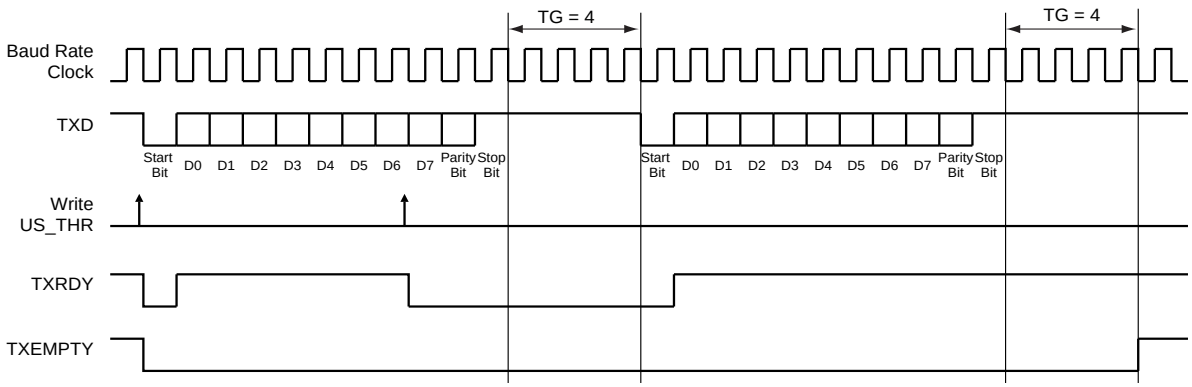


Table 36-9 indicates the maximum length of a timeguard period that the transmitter can handle depending on the baud rate.

Table 36-9. 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

36.6.3.11 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, US_CSR.TIMEOUT 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 TO is written to ‘0’, the Receiver Time-out is disabled and no time-out is detected. US_CSR.TIMEOUT remains at ‘0’. Otherwise, the receiver loads a 16-bit counter with the value programmed in US_RTOR.TO. This counter is decremented at each bit period and reloaded each time a new character is received. If the counter reaches 0, TIMEOUT rises. Then, the user can either:

- Stop the counter clock until a new character is received. This is performed by writing a ‘1’ to US_CR.STTTO. 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. In this case, the counter starts counting down immediately from the value TO. This generates of a periodic interrupt so that a user time-out can be handled, for example when no key is pressed on a keyboard.

Figure 36-23 shows the block diagram of the Receiver Time-out feature.

Figure 36-23. Receiver Time-out Block Diagram

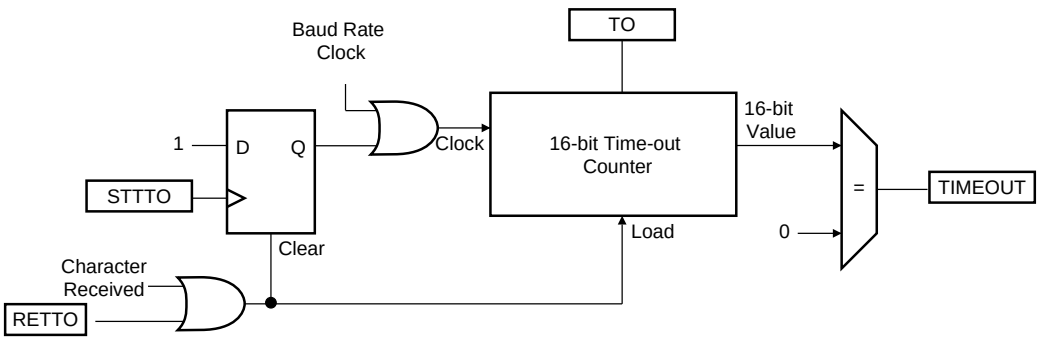


Table 36-10 gives the maximum time-out period for some standard baud rates.

Table 36-10. Maximum Time-out Period

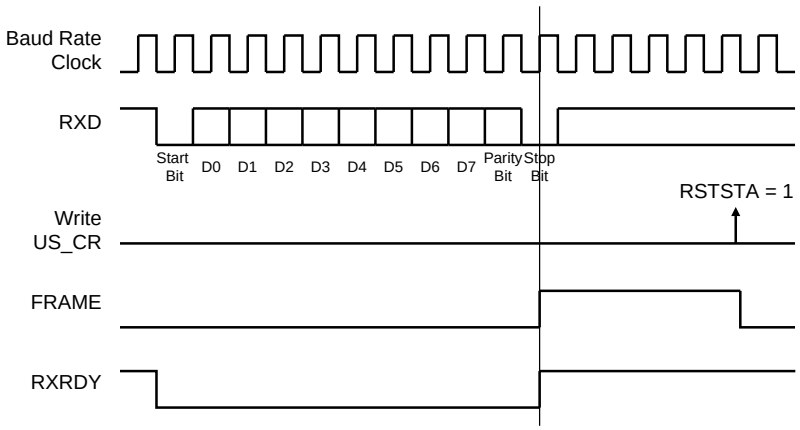
Baud Rate (Bit/s)	Bit Time (µs)	Time-out (ms)
600	1,667	109,225
1,200	833	54,613
2,400	417	27,306
4,800	208	13,653
9,600	104	6,827
14,400	69	4,551
19,200	52	3,413
28,800	35	2,276
38,400	26	1,704
56,000	18	1,170
57,600	17	1,138
200,000	5	328

36.6.3.12 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 in US_CSR.FRAME. FRAME 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 US_CR.RSTSTA.

Figure 36-24. Framing Error Status



36.6.3.13 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 US_CR.STTBRK. 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 US_CR.STPBRK. 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 US_CSR.TXRDY = 1 and the start of the break condition clears the TXRDY and TXEMPTY bits as if a character is processed.

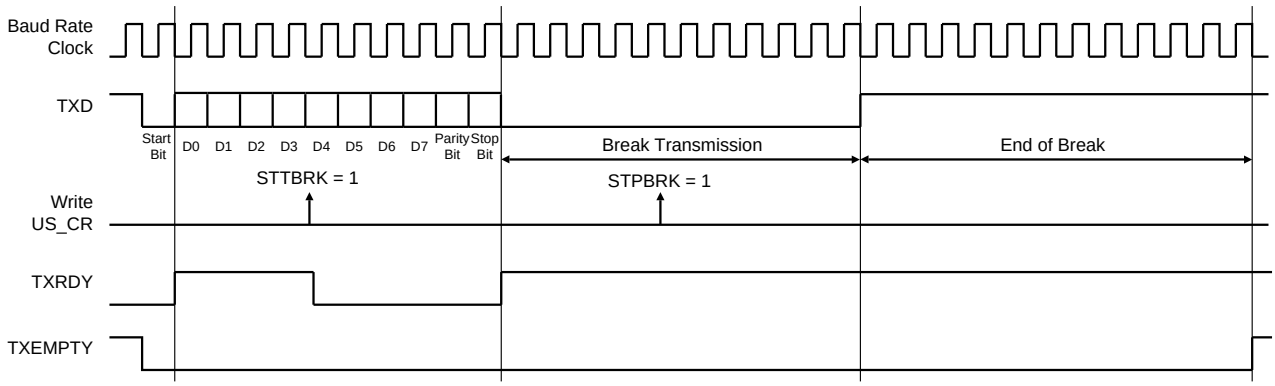
Writing US_CR with both STTBRK and STPBRK bits to ‘1’ can lead to an unpredictable result. All STPBRK commands requested without a previous STTBRK command are ignored. A byte written into US_THR while a break is pending, but not started, is ignored.

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 36-25 illustrates the effect of both the Start Break (STTBRK) and Stop Break (STPBRK) commands on the TXD line.

Figure 36-25. Break Transmission



36.6.3.14 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.

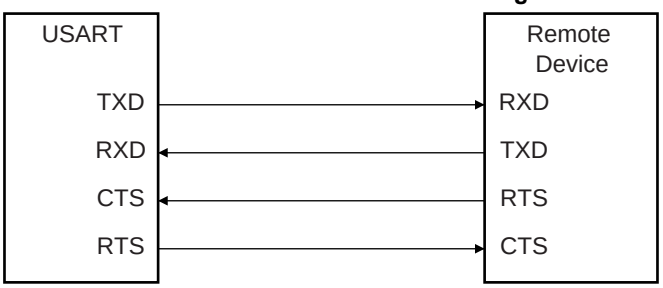
When the low stop bit is detected, the receiver asserts US_CSR.RXBRK. This bit may be cleared by writing a ‘1’ to US_CR.RSTSTA.

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 US_CSR.RXBRK.

36.6.3.15 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 36-26.

Figure 36-26. Connection with a Remote Device for Hardware Handshaking



Setting the USART to operate with hardware handshaking is performed by writing the value 0x2 to US_MR.USART_MODE.

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 figure below. Using this mode requires using the PDC channel for reception. The transmitter can handle hardware handshaking in any case.

Figure 36-27 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 36-27. Receiver Behavior when Operating with Hardware Handshaking

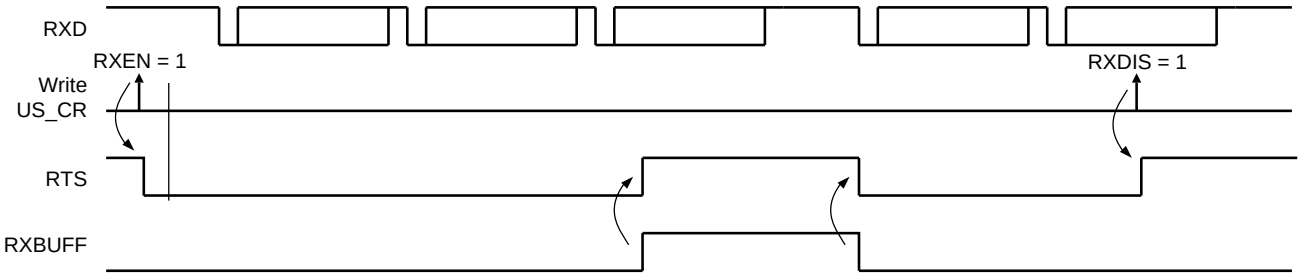
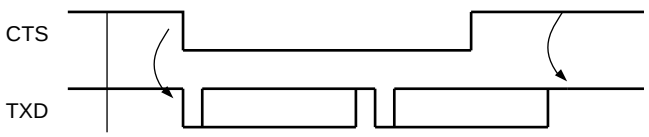


Figure 36-28 shows how the transmitter operates if hardware handshaking is enabled. The CTS pin disables the transmitter. If a character is being processed, the transmitter is disabled only after the completion of the current character and transmission of the next character occurs as soon as the pin CTS falls.

Figure 36-28. Transmitter Behavior when Operating with Hardware Handshaking



36.6.4 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.

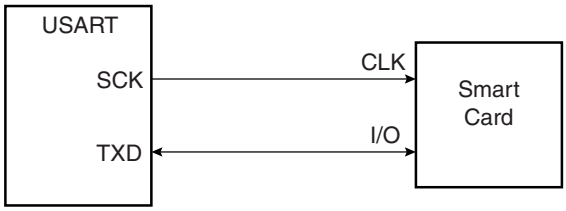
Setting the USART in ISO7816 mode is performed by writing US_MR.USART_MODE to the value 0x4 for protocol T = 0 and to the value 0x6 for protocol T = 1.

36.6.4.1 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 36-2 ”Baud Rate Generator”).

The USART connects to a smart card as shown in Figure 36-29. 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 36-29. 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 Mode register fields CHRL, MODE9, PAR and CHMODE. US_MR.MSBF can be used to transmit LSB or MSB first. US_MR.PAR can be used to transmit in normal or inverse mode. Refer to Section 36.7.3 ”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.

36.6.4.2 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 36-30.

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 36-31. This error bit, NACK, for Non Acknowledge. In this case, the character lasts one additional bit time, as the guard time does not change and is added to the error bit time, which lasts one bit time.

When the USART is the receiver and it detects an error, it does not load the erroneous character in US_RHR. It sets US_SR.PARE so that the software can handle the error.

Figure 36-30. T = 0 Protocol without Parity Error

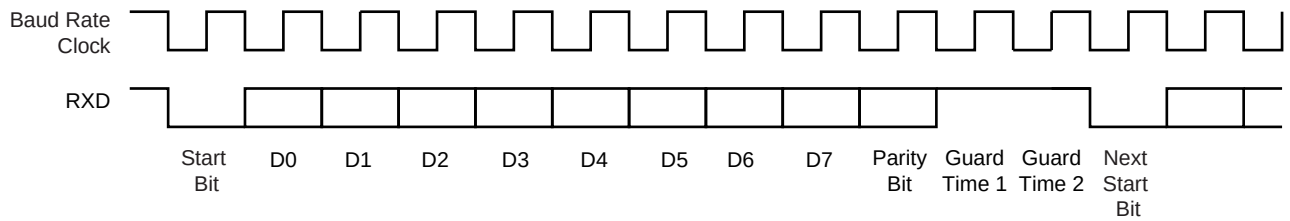
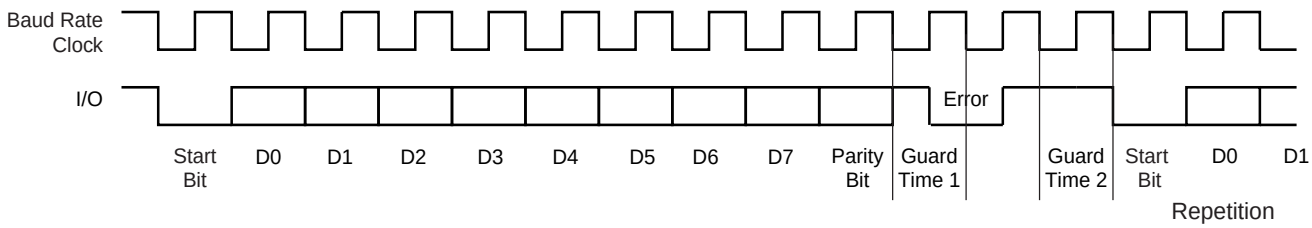


Figure 36-31. T = 0 Protocol with Parity Error



Receive Error Counter

The USART receiver also records the total number of errors. This can be read in the Number of Errors (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 be configured to inhibit an error. This is done by writing a ‘1’ to US_MR.INACK. In this case, no error signal is driven on the I/O line even if a parity bit is detected.

Moreover, if INACK = 1, the erroneous received character is stored in the Receive Holding Register as if no error occurred, and the RXRDY bit rises.

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 US_MR.MAX_ITERATION to a value greater 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 US_CSR.ITER is set. If the repetition of the character is acknowledged by the receiver, the repetitions are stopped and the iteration counter is cleared.

US_CSR.ITER can be cleared by writing a ‘1’ to US_CR.RSTIT.

Disable Successive Receive NACK

The receiver can limit the number of successive NACKs sent back to the remote transmitter. This is programmed by setting US_MR.DSNACK. The maximum number of NACKs transmitted is configured in US_MR.MAX_ITERATION. As soon as MAX_ITERATION is reached, no error signal is driven on the I/O line and US_CSR.ITER is set.

36.6.4.3 Protocol T = 1

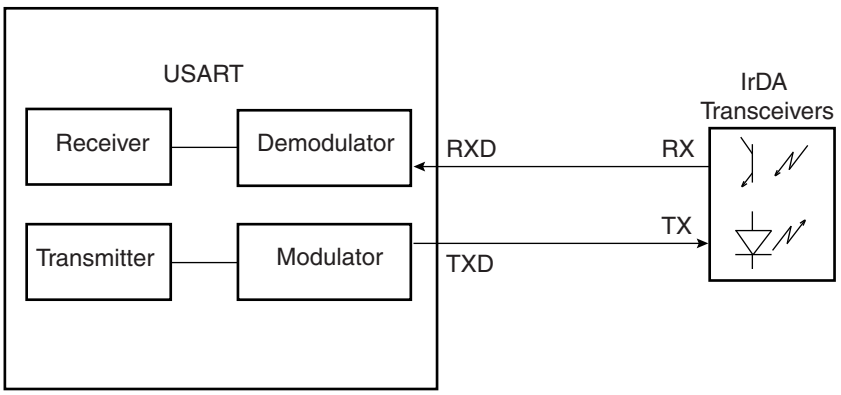
When operating in ISO7816 protocol T = 1, the transmission is similar to an asynchronous format with only one stop bit. The parity is generated when transmitting and checked when receiving. Parity error detection sets US_CSR.PARE.

36.6.5 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 36-32. 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 writing the value 0x8 to US_MR.USART_MODE. 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 36-32. 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.

36.6.5.1 IrDA Modulation

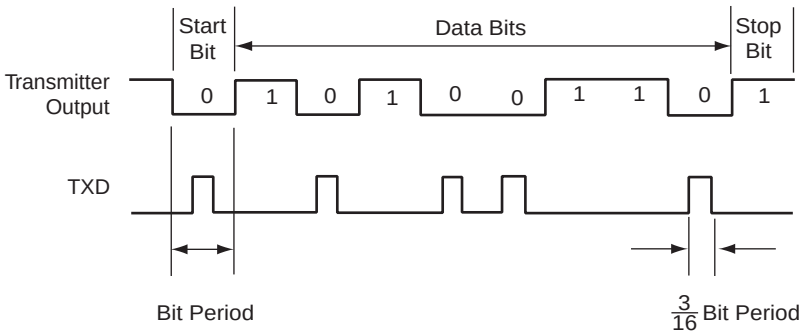
For baud rates up to and including 115.2 kbit/s, the RZI modulation scheme is used. “0” is represented by a light pulse of 3/16th of a bit time. Some examples of signal pulse duration are shown in Table 36-11.

Table 36-11. 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 36-33 shows an example of character transmission.

Figure 36-33. IrDA Modulation



36.6.5.2 IrDA Baud Rate

Table 36-12 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 36-12. 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

Table 36-12. IrDA Baud Rate Error (Continued)

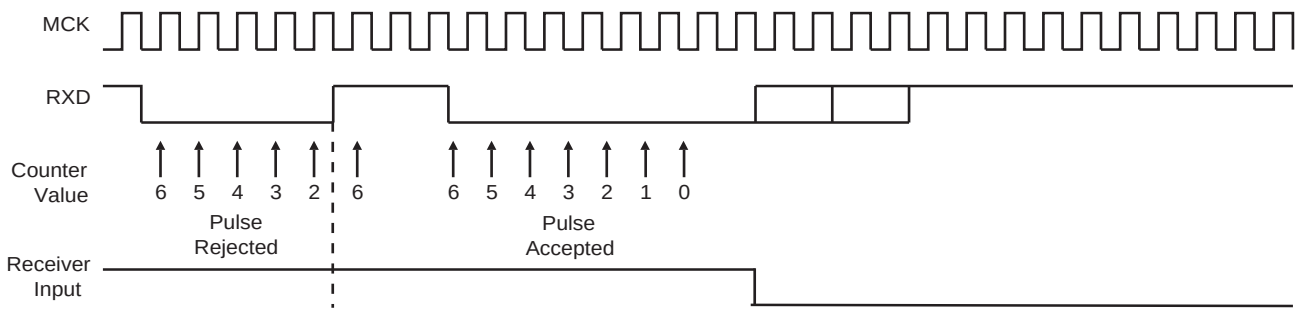
Peripheral Clock	Baud Rate (Bit/s)	CD	Baud Rate Error	Pulse Time (µs)
40,000,000	57,600	43	0.93%	3.26
3,686,400	38,400	6	0.00%	4.88
20,000,000	38,400	33	1.38%	4.88
32,768,000	38,400	53	0.63%	4.88
40,000,000	38,400	65	0.16%	4.88
3,686,400	19,200	12	0.00%	9.77
20,000,000	19,200	65	0.16%	9.77
32,768,000	19,200	107	0.31%	9.77
40,000,000	19,200	130	0.16%	9.77
3,686,400	9,600	24	0.00%	19.53
20,000,000	9,600	130	0.16%	19.53
32,768,000	9,600	213	0.16%	19.53
40,000,000	9,600	260	0.16%	19.53
3,686,400	2,400	96	0.00%	78.13
20,000,000	2,400	521	0.03%	78.13
32,768,000	2,400	853	0.04%	78.13

36.6.5.3 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 36-34 illustrates the operations of the IrDA demodulator.

Figure 36-34. IrDA Demodulator Operations



The programmed value in the US_IF register must always meet the following criterion:

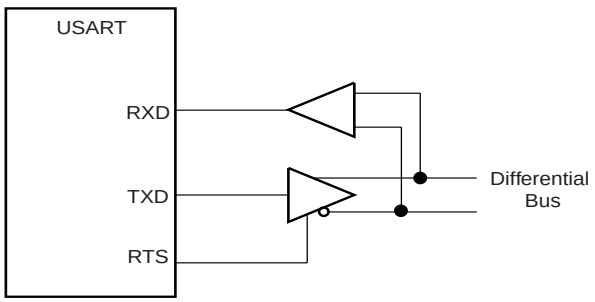
$$t_{peripheral\ clock} \times (IRDA\_FILTER + 3) < 1.41\ \mu s$$

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 ensure IrDA communications operate correctly.

36.6.6 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 36-35.

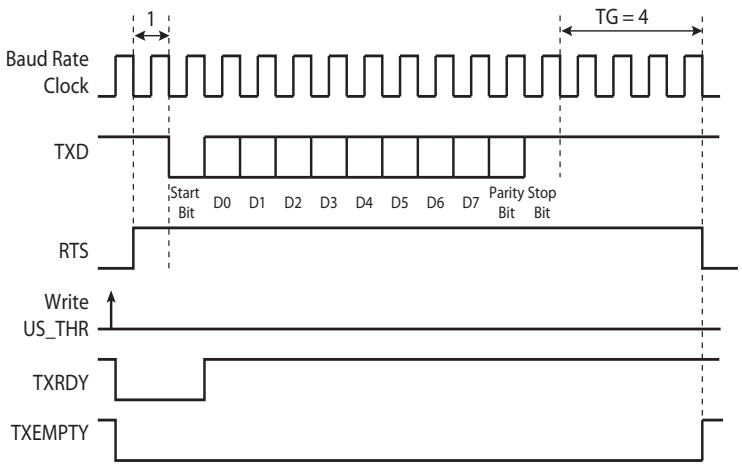
Figure 36-35. Typical Connection to a RS485 Bus



RS485 mode is enabled by writing the value 0x1 to the US_MR.USART_MODE.

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 36-36 gives an example of the RTS waveform during a character transmission when the timeguard is enabled.

Figure 36-36. Example of RTS Drive with Timeguard



36.6.7 SPI Mode

The Serial Peripheral Interface (SPI) mode is a synchronous serial data link that provides communication with external devices in Master or Slave mode. It also enables communication between processors if an external processor is connected to the system.

The Serial Peripheral Interface is 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.

36.6.7.1 Modes of Operation

The USART can operate in SPI Master mode or in SPI Slave mode.

SPI Master mode is enabled by writing 0xE to US_MR.USART_MODE. 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.

SPI Slave mode is enabled by writing 0xF to US_MR.USART_MODE. 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 36.6.7.4.

36.6.7.2 Baud Rate

In SPI mode, the baud rate generator operates in the same way as in USART Synchronous mode. See Section 36.6.1.3 ”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 US_MR.CLKO must be written to ‘1’, 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 US_BRGR.CD must be greater than 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 US_MR.USCLKS. 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.

36.6.7.3 Data Transfer

Up to nine data bits are successively shifted out on the TXD pin at each rising or falling edge (depending on 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 using US_MR.CHRL and US_MR.MODE9. The nine bits are selected by setting the MODE9 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 using US_MR.CPOL. The clock phase is programmed using US_MR.CPHA. 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 36-13. SPI Bus Protocol Mode

SPI Bus Protocol Mode	CPOL	CPHA
0	0	1
1	0	0
2	1	1
3	1	0

Figure 36-37. SPI Transfer Format (CPHA = 1, 8 bits per transfer)

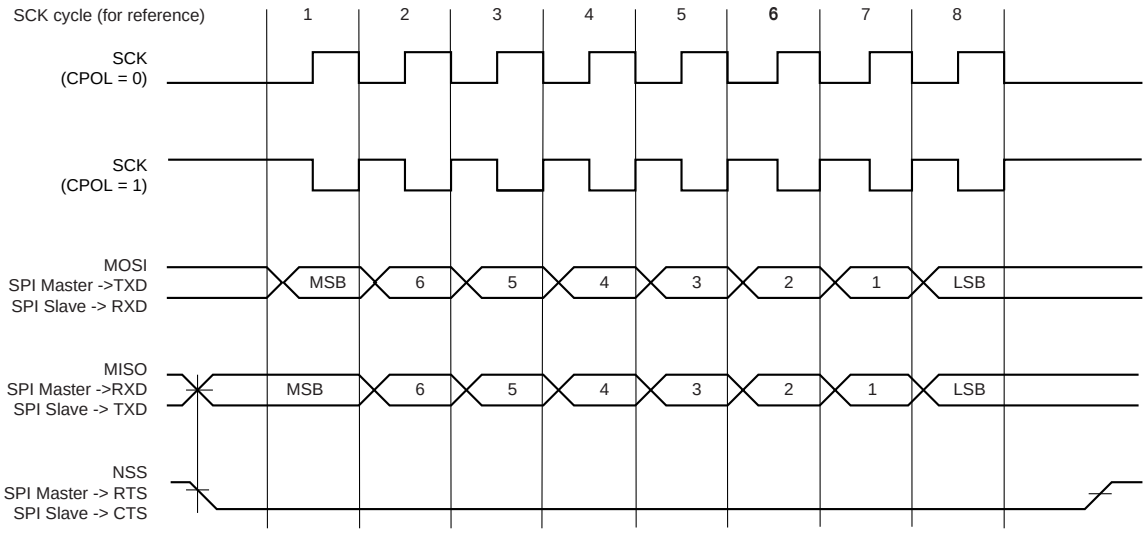
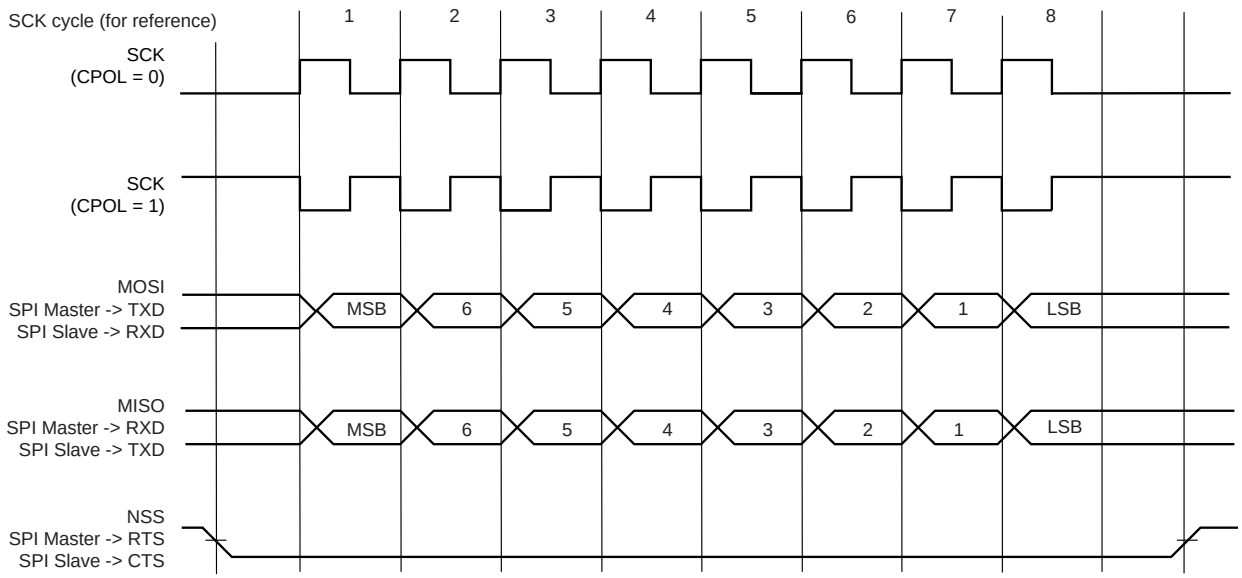


Figure 36-38. SPI Transfer Format (CPHA = 0, 8 bits per transfer)



36.6.7.4 Receiver and Transmitter Control

See Section 36.6.2 ”Receiver and Transmitter Control”.

36.6.7.5 Character Transmission

The characters are sent by writing in the US_THR. An additional condition for transmitting a character can be added when the USART is configured in SPI Master mode. In the USART_MR (SPI_MODE), the value of 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 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 deasserted 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 US_CR.FCS (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.

36.6.7.6 Character Reception

When a character reception is completed, it is transferred to US_RHR and US_CSR.RXRDY 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 US_CR.RSTSTA.

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.

36.6.7.7 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 value is in US_RTOR.TO.

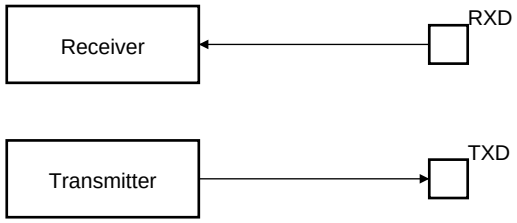
36.6.8 Test Modes

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.

36.6.8.1 Normal Mode

Normal mode connects the RXD pin on the receiver input and the transmitter output on the TXD pin.

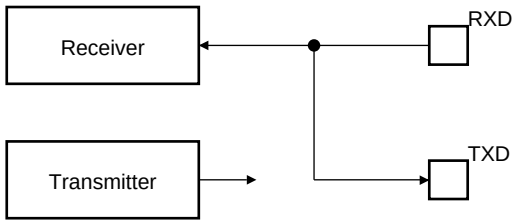
Figure 36-39. Normal Mode Configuration



36.6.8.2 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 36-40. 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.

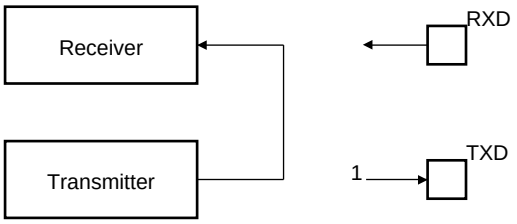
Figure 36-40. Automatic Echo Mode Configuration



36.6.8.3 Local Loopback Mode

Local loopback mode connects the output of the transmitter directly to the input of the receiver, as shown in Figure 36-41. 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.

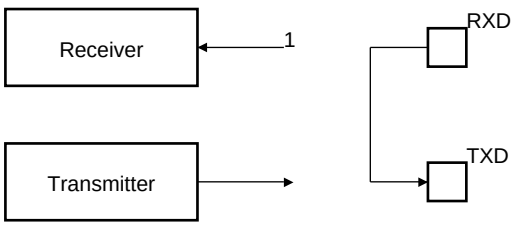
Figure 36-41. Local Loopback Mode Configuration



36.6.8.4 Remote Loopback Mode

Remote loopback mode directly connects the RXD pin to the TXD pin, as shown in Figure 36-42. The transmitter and the receiver are disabled and have no effect. This mode allows bit-by-bit retransmission.

Figure 36-42. Remote Loopback Mode Configuration



36.6.9 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 Manchester Configuration Register"

36.7 Universal Synchronous Asynchronous Receiver Transmitter (USART) User Interface

Table 36-14. 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	–	–	–

36.7.1 USART Control Register

Name: US_CR

Address: 0x40024000 (0), 0x40028000 (1), 0x4002C000 (2), 0x40030000 (3), 0x40034000 (4)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	RTSDIS	RTSEN	–	–
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 36.7.2 ”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.

- **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.

36.7.2 USART Control Register (SPI_MODE)

Name: US_CR (SPI_MODE)

Address: 0x40024000 (0), 0x40028000 (1), 0x4002C000 (2), 0x40030000 (3), 0x40034000 (4)

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).

36.7.3 USART Mode Register

Name: US_MR

Address: 0x40024004 (0), 0x40028004 (1), 0x4002C004 (2), 0x40030004 (3), 0x40034004 (4)

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 36.7.4 ”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	-	Reserved
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: 16X Oversampling.

1: 8X Oversampling.

- **INACK: Inhibit Non Acknowledge**

0: The NACK is generated.

1: The NACK is not generated.

- **DSNACK: Disable Successive NACK**

0: NACK is sent on the ISO line as soon as a parity error occurs in the received character (unless INACK is set).

1: Successive parity errors are counted up to the value specified in the MAX_ITERATION field. These parity errors generate a NACK on the ISO line. As soon as this value is reached, no additional NACK is sent on the ISO line. The flag 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 ISO7816 mode, 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.

36.7.4 USART Mode Register (SPI_MODE)

Name: US_MR (SPI_MODE)

Address: 0x40024004 (0), 0x40028004 (1), 0x4002C004 (2), 0x40030004 (3), 0x40034004 (4)

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 (SCK) 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 (SCK). 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).

36.7.5 USART Interrupt Enable Register

Name: US_IER

Address: 0x40024008 (0), 0x40028008 (1), 0x4002C008 (2), 0x40030008 (3), 0x40034008 (4)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	MANE
23	22	21	20	19	18	17	16
–	–	–	–	CTSIC	–	–	–
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 36.7.6 ”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**
- **CTSIC: Clear to Send Input Change Interrupt Enable**
- **MANE: Manchester Error Interrupt Enable**

36.7.6 USART Interrupt Enable Register (SPI_MODE)

Name: US_IER (SPI_MODE)

Address: 0x40024008 (0), 0x40028008 (1), 0x4002C008 (2), 0x40030008 (3), 0x40034008 (4)

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**

36.7.7 USART Interrupt Disable Register

Name: US_IDR

Address: 0x4002400C (0), 0x4002800C (1), 0x4002C00C (2), 0x4003000C (3), 0x4003400C (4)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	MANE
23	22	21	20	19	18	17	16
–	–	–	–	CTSIC	–	–	–
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 36.7.8 ”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 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 Disable**
- **FRAME: Framing Error Interrupt Disable**
- **PARE: Parity Error Interrupt Disable**
- **TIMEOUT: Time-out Interrupt Disable**
- **TXEMPTY: TXEMPTY Interrupt Disable**
- **ITER: Max Number of Repetitions Reached 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**
- **CTSIC: Clear to Send Input Change Interrupt Disable**
- **MANE: Manchester Error Interrupt Disable**

36.7.8 USART Interrupt Disable Register (SPI_MODE)

Name: US_IDR (SPI_MODE)

Address: 0x4002400C (0), 0x4002800C (1), 0x4002C00C (2), 0x4003000C (3), 0x4003400C (4)

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 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**

36.7.9 USART Interrupt Mask Register

Name: US_IMR

Address: 0x40024010 (0), 0x40028010 (1), 0x4002C010 (2), 0x40030010 (3), 0x40034010 (4)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	MANE
23	22	21	20	19	18	17	16
–	–	–	–	CTSIC	–	–	–
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 36.7.10 ”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**
- **CTSIC: Clear to Send Input Change Interrupt Mask**
- **MANE: Manchester Error Interrupt Mask**

36.7.10 USART Interrupt Mask Register (SPI_MODE)

Name: US_IMR (SPI_MODE)

Address: 0x40024010 (0), 0x40028010 (1), 0x4002C010 (2), 0x40030010 (3), 0x40034010 (4)

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**

36.7.11 USART Channel Status Register

Name: US_CSR

Address: 0x40024014 (0), 0x40028014 (1), 0x4002C014 (2), 0x40030014 (3), 0x40034014 (4)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	MANERR
23	22	21	20	19	18	17	16
CTS	–	–	–	CTSIC	–	–	–
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 36.7.12 "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.

- **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.

- **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.

36.7.12 USART Channel Status Register (SPI_MODE)

Name: US_CSR (SPI_MODE)

Address: 0x40024014 (0), 0x40028014 (1), 0x4002C014 (2), 0x40030014 (3), 0x40034014 (4)

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).

36.7.13 USART Receive Holding Register

Name: US_RHR

Address: 0x40024018 (0), 0x40028018 (1), 0x4002C018 (2), 0x40030018 (3), 0x40034018 (4)

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.

36.7.14 USART Transmit Holding Register

Name: US_THR

Address: 0x4002401C (0), 0x4002801C (1), 0x4002C01C (2), 0x4003001C (3), 0x4003401C (4)

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.

36.7.15 USART Baud Rate Generator Register

Name: US_BRGR

Address: 0x40024020 (0), 0x40028020 (1), 0x4002C020 (2), 0x40030020 (3), 0x40034020 (4)

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**

CD	USART_MODE ≠ ISO7816			USART_MODE = ISO7816
	SYNC = 0		SYNC = 1 or USART_MODE = SPI (Master or Slave)	
	OVER = 0	OVER = 1		
0	Baud Rate Clock Disabled			
1 to 65535	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 nonconstant 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.

36.7.16 USART Receiver Time-out Register

Name: US_RTOR

Address: 0x40024024 (0), 0x40028024 (1), 0x4002C024 (2), 0x40030024 (3), 0x40034024 (4)

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.

36.7.17 USART Transmitter Timeguard Register

Name: US_TTGR

Address: 0x40024028 (0), 0x40028028 (1), 0x4002C028 (2), 0x40030028 (3), 0x40034028 (4)

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.

36.7.18 USART FI DI RATIO Register

Name: US_FIDI

Address: 0x40024040 (0), 0x40028040 (1), 0x4002C040 (2), 0x40030040 (3), 0x40034040 (4)

Access: Read/Write

Reset: 0x174

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

This register can only be written if the WPEN bit is cleared in 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.

36.7.19 USART Number of Errors Register

Name: US_NER

Address: 0x40024044 (0), 0x40028044 (1), 0x4002C044 (2), 0x40030044 (3), 0x40034044 (4)

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.

36.7.20 USART IrDA Filter Register

Name: US_IF

Address: 0x4002404C (0), 0x4002804C (1), 0x4002C04C (2), 0x4003004C (3), 0x4003404C (4)

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:

$$t_{\text{peripheral clock}} \times (\text{IRDA\_FILTER} + 3) < 1.41\ \mu s$$

36.7.21 USART Manchester Configuration Register

Name: US_MAN

Address: 0x40024050 (0), 0x40028050 (1), 0x4002C050 (2), 0x40030050 (3), 0x40034050 (4)

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 x 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 x Bit Period.

- **RX_PP: Receiver Preamble Pattern detected**

The following values assume that RX_MPOL field is not set:

Value	Name	Description
00	ALL_ONE	The preamble is composed of ‘1’s
01	ALL_ZERO	The preamble is composed of ‘0’s
10	ZERO_ONE	The preamble is composed of ‘01’s
11	ONE_ZERO	The preamble is composed of ‘10’s

- **RX_MPOL: Receiver Manchester Polarity**

0: Logic zero is coded as a zero-to-one transition, Logic one is coded as a one-to-zero transition.

1: Logic zero is coded as a one-to-zero transition, Logic one is coded as a zero-to-one transition.

- **ONE: Must Be Set to 1**

Bit 29 must always be set to 1 when programming the US_MAN register.

- **DRIFT: Drift Compensation**

0: The USART cannot recover from an important clock drift.

1: The USART can recover from clock drift. The 16X clock mode must be enabled.

36.7.22 USART Write Protection Mode Register

Name: US_WPMR

Address: 0x400240E4 (0), 0x400280E4 (1), 0x4002C0E4 (2), 0x400300E4 (3), 0x400340E4 (4)

Access: Read/Write

Reset: See Table 36-14

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 36.6.9 ”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.

36.7.23 USART Write Protection Status Register

Name: US_WPSR

Address: 0x400240E8 (0), 0x400280E8 (1), 0x4002C0E8 (2), 0x400300E8 (3), 0x400340E8 (4)

Access: Read-only

Reset: See Table 36-14

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.

37. Timer Counter (TC)

37.1 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 multipurpose 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.

37.2 Embedded Characteristics

- Total number of TC channels implemented on this device: three
- TC channel size: 16-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 multipurpose input/output signals acting as trigger event
- Internal interrupt signal
- Register Write Protection

37.3 Block Diagram

Table 37-1. 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 37-1. Timer Counter Block Diagram

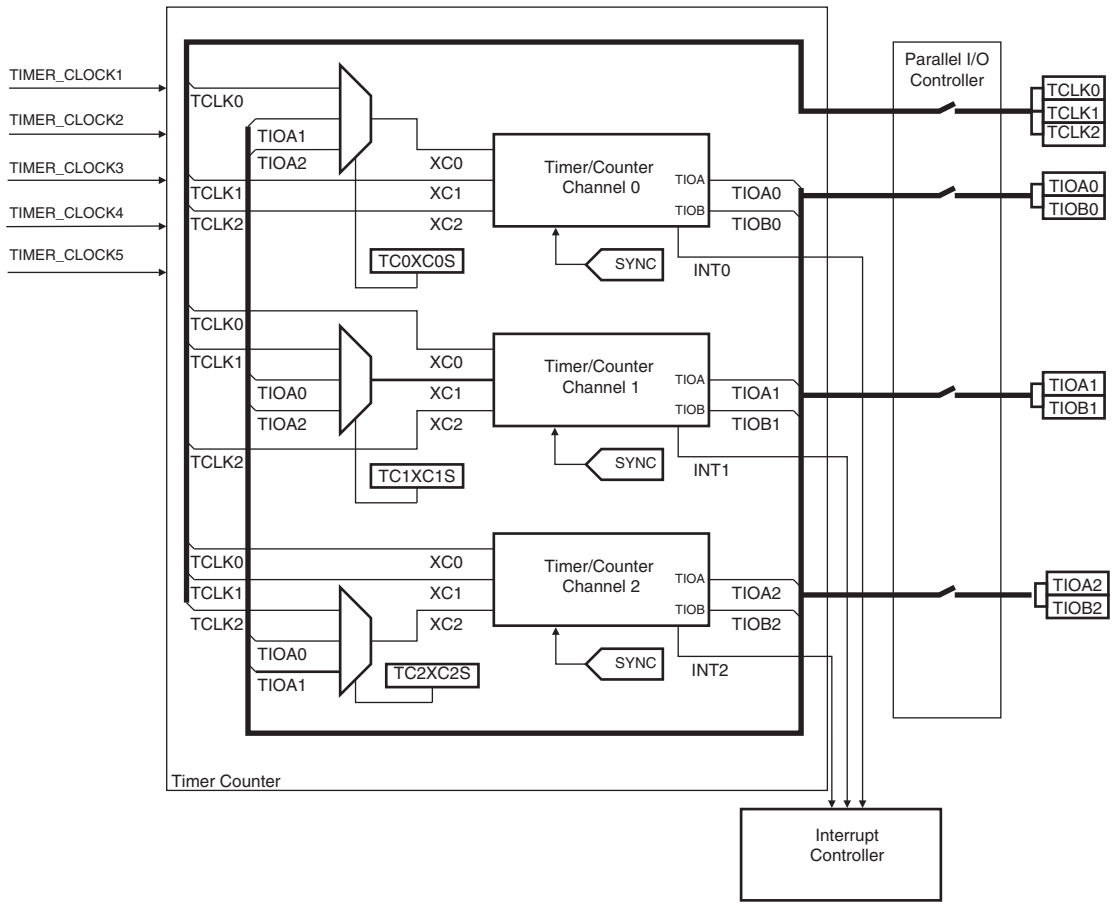


Table 37-2. Channel Signal Description

Signal Name	Description
XC0, XC1, XC2	External Clock Inputs
TIOAx	Capture Mode: Timer Counter Input Waveform Mode: Timer Counter Output
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)

37.4 Pin List

Table 37-3. TC Pin List

Pin Name	Description	Type
TCLK0 - TCLK2	External Clock Input	Input
TIOA0 - TIOA2	I/O Line A	I/O
TIOB0 - TIOB2	I/O Line B	I/O

37.5 Product Dependencies

37.5.1 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 37-4. I/O Lines

Instance	Signal	I/O Line	Peripheral
TC0	TCLK0	PB4	B
TC0	TCLK1	PB9	A
TC0	TCLK2	PB12	A
TC0	TIOA0	PA13	B
TC0	TIOA1	PB7	A
TC0	TIOA2	PB10	A
TC0	TIOB0	PA14	B
TC0	TIOB1	PB8	A
TC0	TIOB2	PB11	A
TC1	TCLK3	PB26	A
TC1	TCLK4	PA17	B
TC1	TCLK5	PA19	B
TC1	TIOA3	PB24	A
TC1	TIOA4	PA15	B
TC1	TIOA5	PA18	B
TC1	TIOB3	PB25	A
TC1	TIOB4	PA16	B
TC1	TIOB5	PA20	B

37.5.2 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.

37.5.3 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 37-5. Peripheral IDs

Instance	ID
TC0	23
TC1	24

37.6 Functional Description

37.6.1 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 37-6.

37.6.2 16-bit Counter

Each 16-bit channel is organized around a 16-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^{16} - 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.

37.6.3 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 37-2 ”Clock Chaining Selection”.

Each channel can independently select an internal or external clock source for its counter:

- External clock signals(2): 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 TC_CMR.CLKI. 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 37-3 ”Clock Selection”.

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 37-2. Clock Chaining Selection

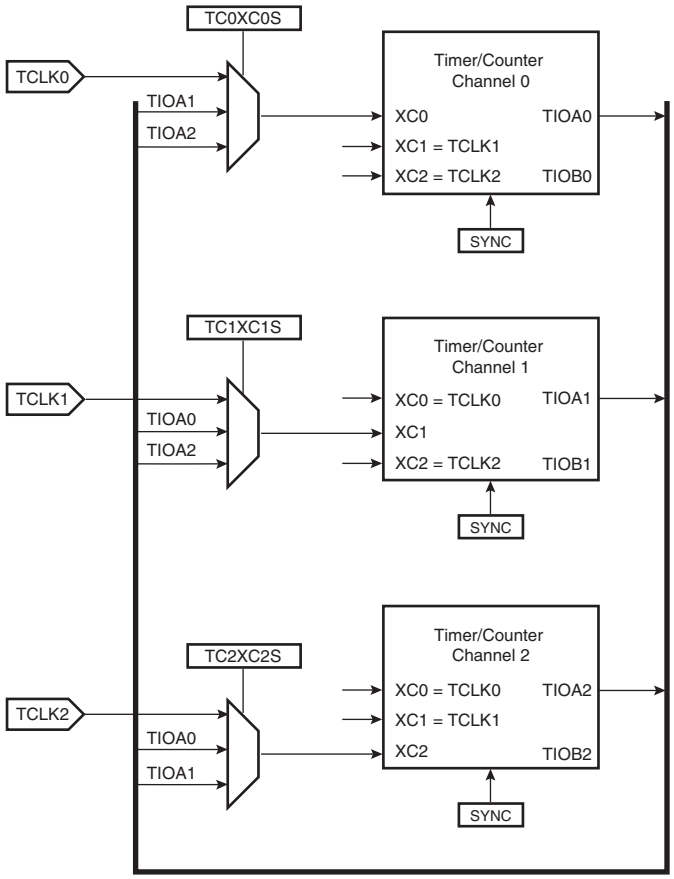
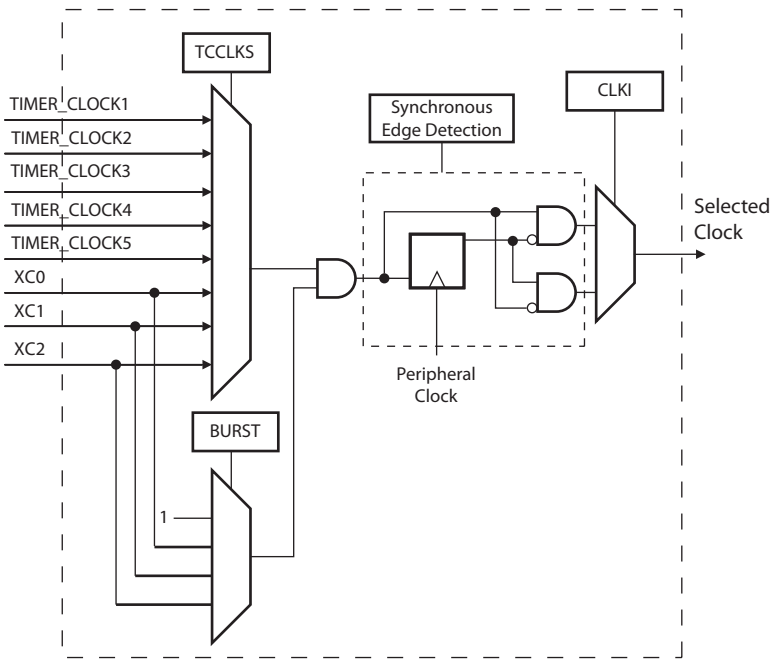


Figure 37-3. Clock Selection

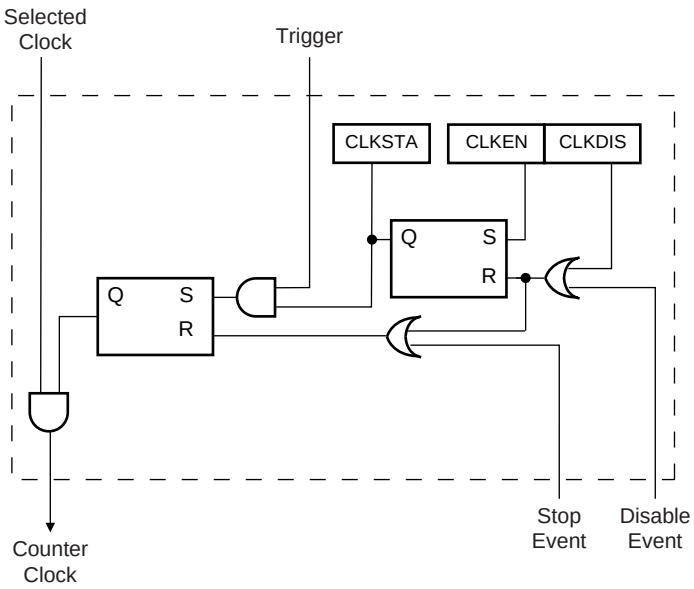


37.6.4 Clock Control

The clock of each counter can be controlled in two different ways: it can be enabled/disabled and started/stopped. See Figure 37-4.

- 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 TC_CMR.LDBDIS is set to 1. In Waveform mode, it can be disabled by an RC Compare event if TC_CMR.CPCDIS is set to 1. When disabled, the start or the stop actions have no effect: only a CLKEN command in the TC_CCR can reenable the clock. When the clock is enabled, TC_SR.CLKSTA is set.
- 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 37-4. Clock Control



37.6.5 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 TC_CMR.WAVE.

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.

37.6.6 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 TC_CCR.SWTRG.
- 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 TC_CMR.CPCTRG is set.

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 TC_CMR.ENETRG.

If an external trigger is used, the duration of the pulses must be longer than the peripheral clock period in order to be detected.

37.6.7 Capture Mode

Capture mode is entered by clearing TC_CMR.WAVE.

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 37-5 shows the configuration of the TC channel when programmed in Capture mode.

37.6.8 Capture Registers A and B

Registers A and B (TC_RA and TC_RB) are used as capture registers. They can be loaded with the counter value when a programmable event occurs on the signal TIOAx.

TC_CMR.LDRA defines the TIOAx selected edge for the loading of TC_RA, and TC_CMR.LDRB defines the TIOAx selected edge for the loading of TC_RB.

TC_RA is loaded only if it has not been loaded since the last trigger or if TC_RB has been loaded since the last loading of TC_RA.

TC_RB is loaded only if TC_RA has been loaded since the last trigger or the last loading of TC_RB.

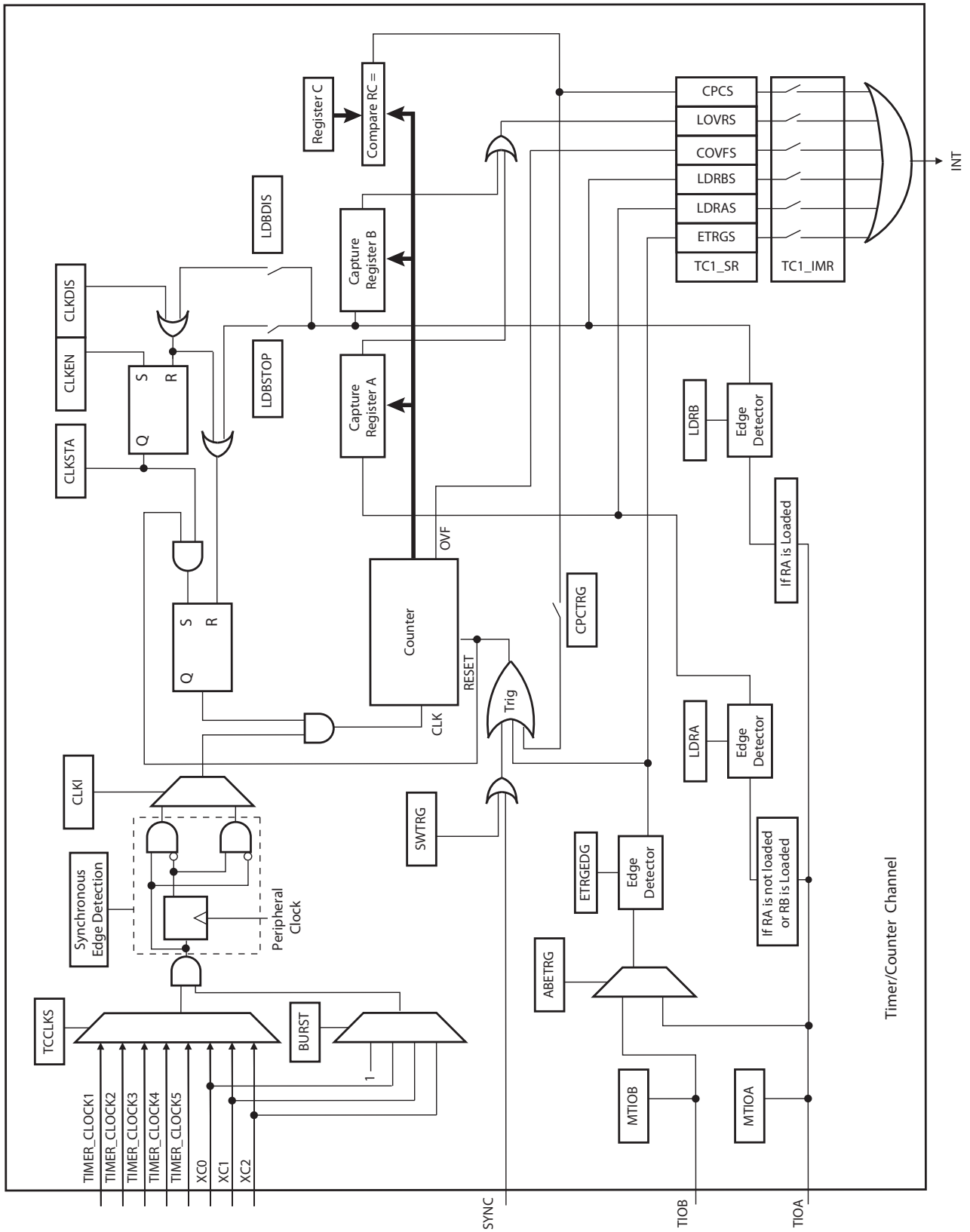
Loading TC_RA or TC_RB before the read of the last value loaded sets the Overrun Error Flag (LOVRS bit) in the TC_SR. In this case, the old value is overwritten.

37.6.9 Trigger Conditions

In addition to the SYNC signal, the software trigger and the RC compare trigger, an external trigger can be defined.

The ABETRG bit in the TC_CMR selects TIOAx or TIOBx input signal as an external trigger. The External Trigger Edge Selection parameter (ETRGEDG field in TC_CMR) defines the edge (rising, falling or both) detected to generate an external trigger. If ETRGEDG = 0 (none), the external trigger is disabled.

Figure 37-5. Capture Mode



37.6.10 Waveform Mode

Waveform mode is entered by setting the TC_CMRx.WAVE bit.

In Waveform mode, the TC channel generates one or two PWM signals with the same frequency and independently programmable duty cycles, or generates different types of one-shot or repetitive pulses.

In this mode, 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 37-6 shows the configuration of the TC channel when programmed in Waveform operating mode.

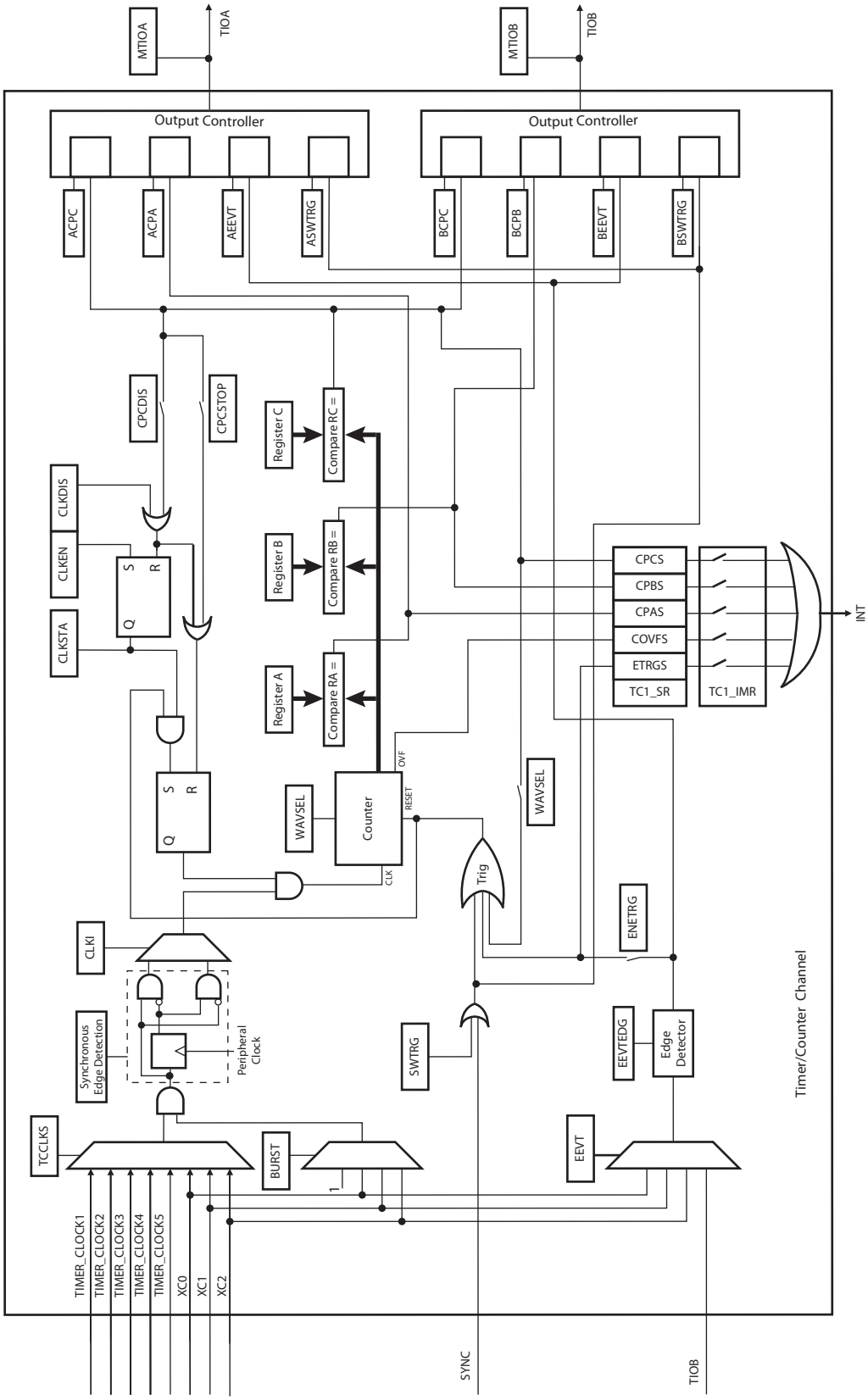
37.6.11 Waveform Selection

Depending on the WAVSEL parameter in TC_CMR, the behavior of TC_CV varies.

With any selection, TC_RA, TC_RB and TC_RC can all be used as compare registers.

RA Compare is used to control the 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.

Figure 37-6. Waveform Mode



37.6.11.1 WAVSEL = 00

When WAVSEL = 00, the value of TC_CV is incremented from 0 to 2^{16} - 1. Once 2^{16} - 1 has been reached, the value of TC_CV is reset. Incrementation of TC_CV starts again and the cycle continues. See Figure 37-7.

An external event trigger or a software trigger can reset the value of TC_CV. It is important to note that the trigger may occur at any time. See Figure 37-8.

RC Compare cannot be programmed to generate a trigger in this configuration. At the same time, RC Compare can stop the counter clock (CPCSTOP = 1 in TC_CMR) and/or disable the counter clock (CPCDIS = 1 in TC_CMR).

Figure 37-7. WAVSEL = 00 without Trigger

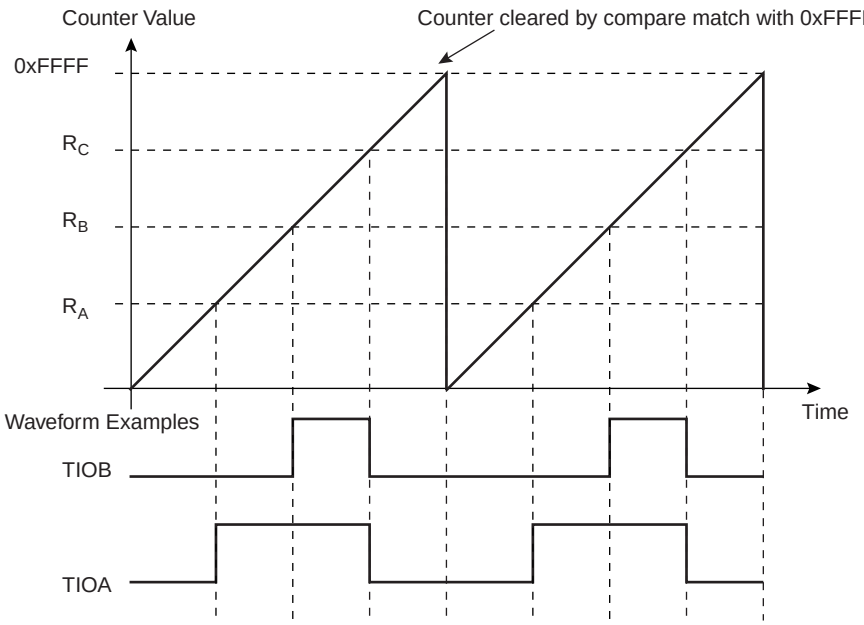
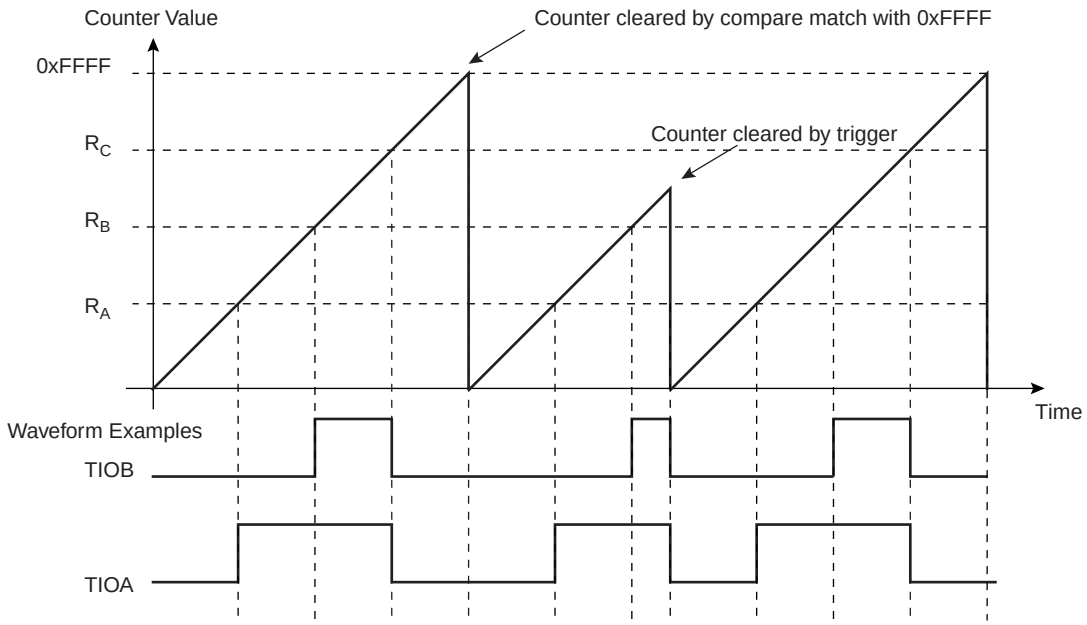


Figure 37-8. WAVSEL = 00 with Trigger



37.6.11.2 WAVSEL = 10

When WAVSEL = 10, the value of TC_CV is incremented from 0 to the value of RC, then automatically reset on a RC Compare. Once the value of TC_CV has been reset, it is then incremented and so on. See Figure 37-9.

It is important to note that TC_CV can be reset at any time by an external event or a software trigger if both are programmed correctly. See Figure 37-10.

In addition, RC Compare can stop the counter clock (CPCSTOP = 1 in TC_CMR) and/or disable the counter clock (CPCDIS = 1 in TC_CMR).

Figure 37-9. WAVSEL = 10 without Trigger

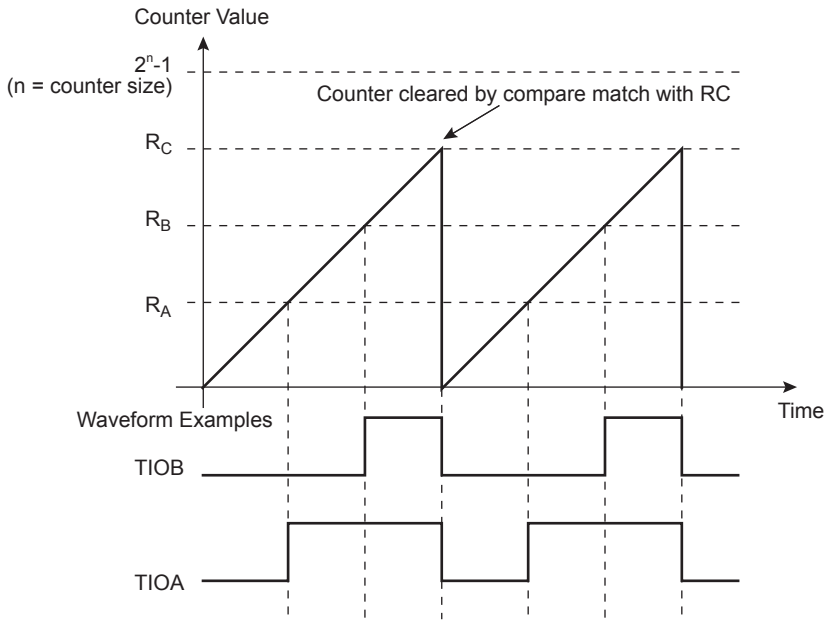
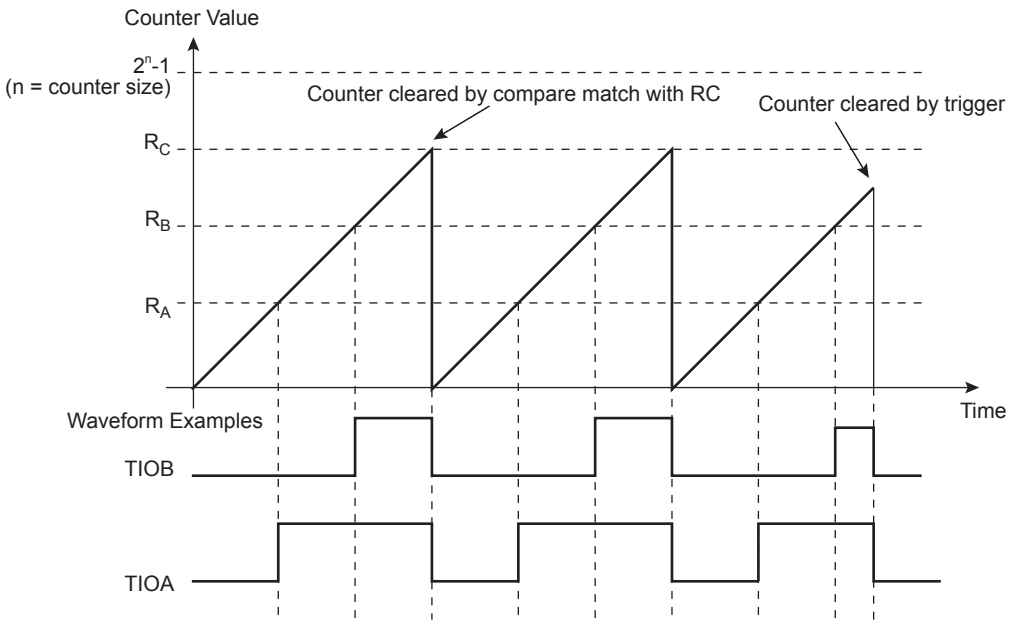


Figure 37-10. WAVSEL = 10 with Trigger



37.6.11.3 WAVSEL = 01

When WAVSEL = 01, the value of TC_CV is incremented from 0 to 2^{16} - 1. Once 2^{16} - 1 is reached, the value of TC_CV is decremented to 0, then reincremented to 2^{16} - 1 and so on. See Figure 37-11.

A trigger such as an external event or a software trigger can modify TC_CV at any time. If a trigger occurs while TC_CV is incrementing, TC_CV then decrements. If a trigger is received while TC_CV is decrementing, TC_CV then increments. See Figure 37-12.

RC Compare cannot be programmed to generate a trigger in this configuration.

At the same time, RC Compare can stop the counter clock (CPCSTOP = 1) and/or disable the counter clock (CPCDIS = 1).

Figure 37-11. WAVSEL = 01 without Trigger

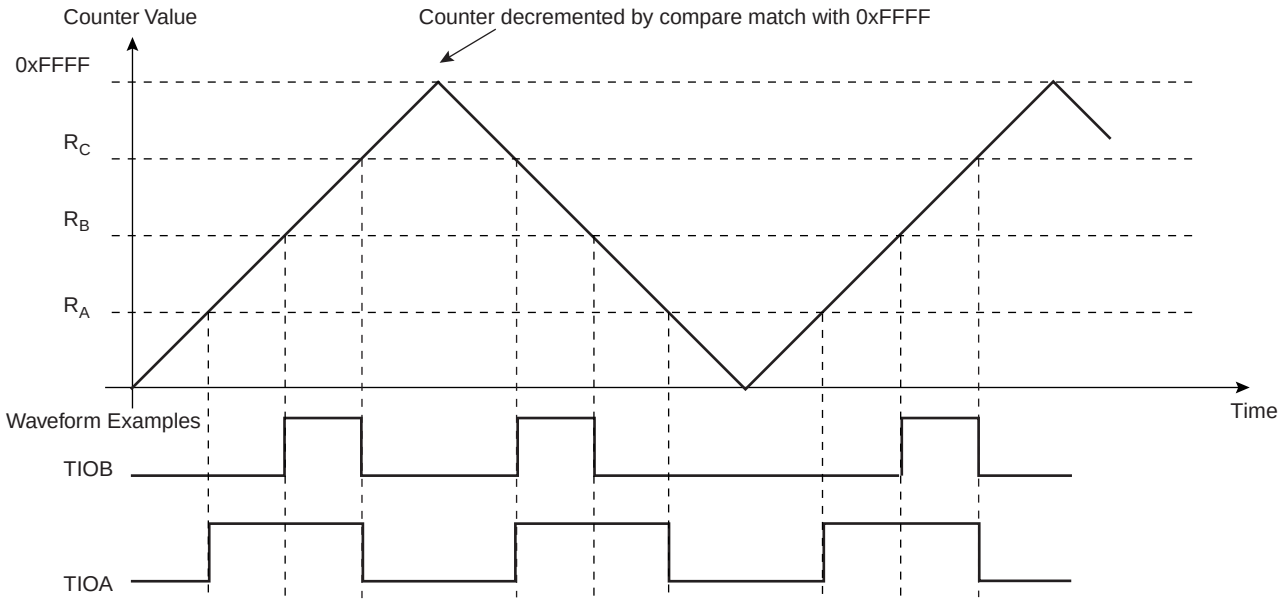
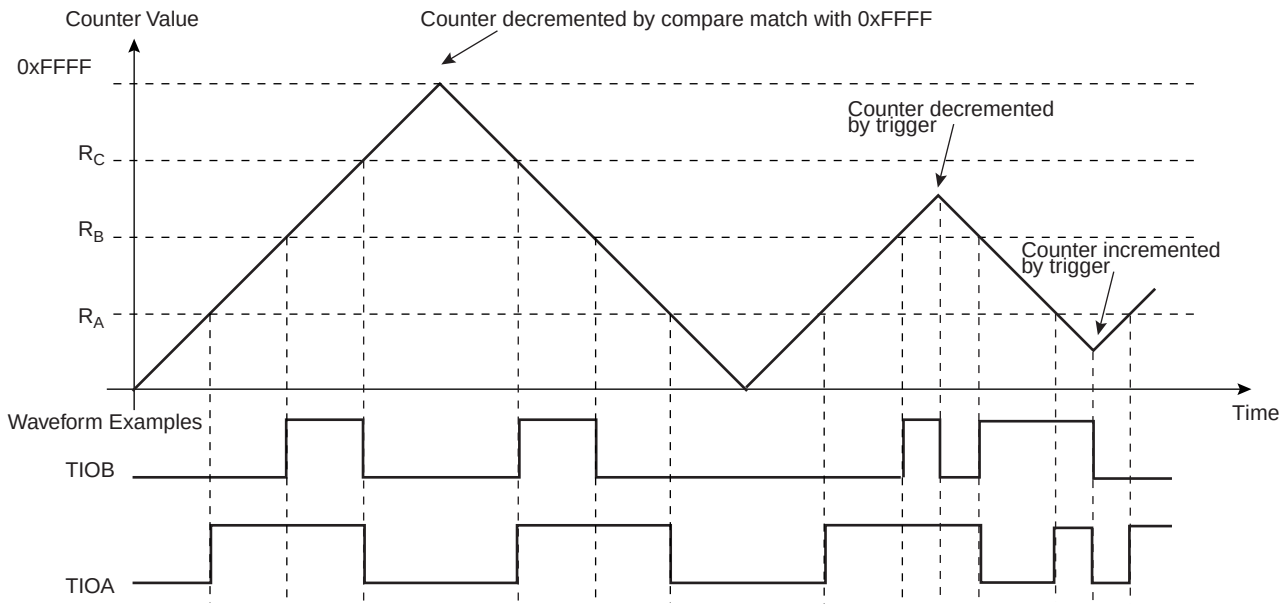


Figure 37-12. WAVSEL = 01 with Trigger



37.6.11.4 WAVSEL = 11

When WAVSEL = 11, the value of TC_CV is incremented from 0 to RC. Once RC is reached, the value of TC_CV is decremented to 0, then reincremented to RC and so on. See Figure 37-13.

A trigger such as an external event or a software trigger can modify TC_CV at any time. If a trigger occurs while TC_CV is incrementing, TC_CV then decrements. If a trigger is received while TC_CV is decrementing, TC_CV then increments. See Figure 37-14.

RC Compare can stop the counter clock (CPCSTOP = 1) and/or disable the counter clock (CPCDIS = 1).

Figure 37-13. WAVSEL = 11 without Trigger

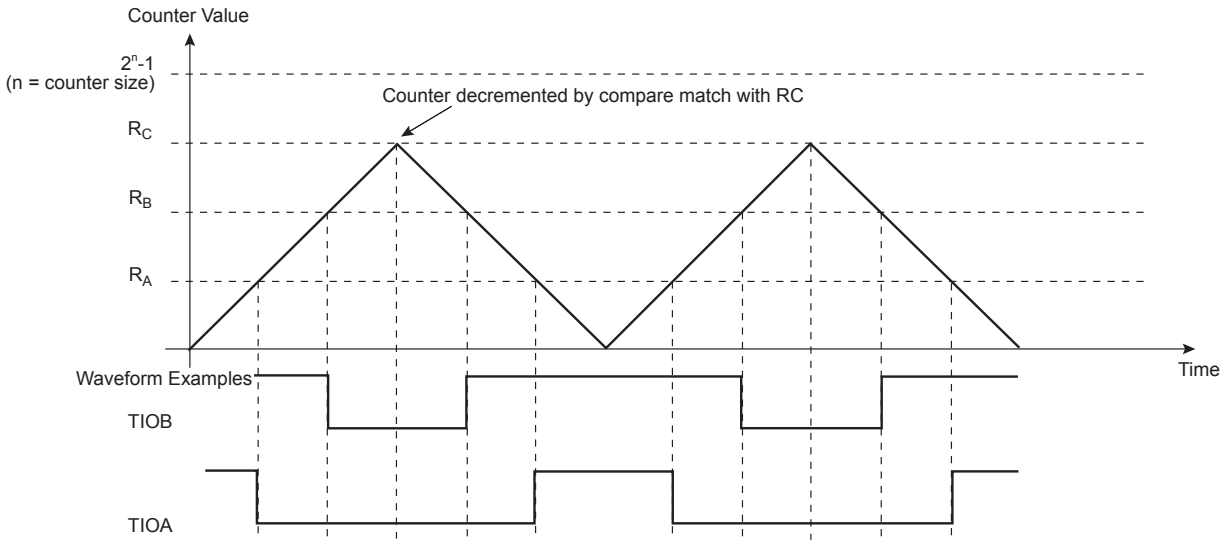
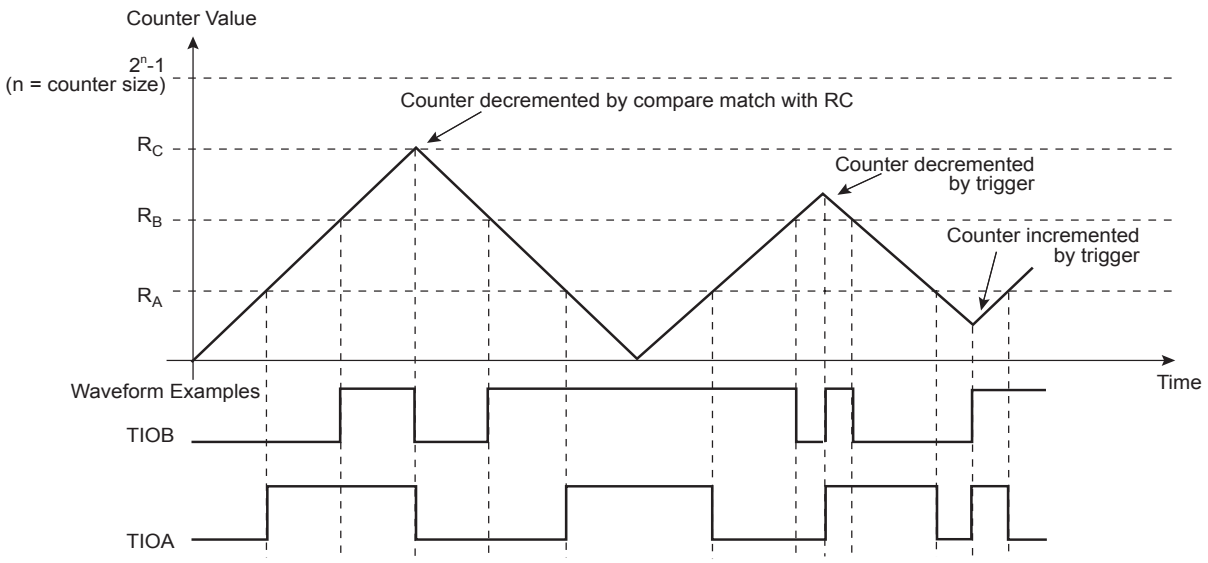


Figure 37-14. WAVSEL = 11 with Trigger



37.6.12 External Event/Trigger Conditions

An external event can be programmed to be detected on one of the clock sources (XC0, XC1, XC2) or TIOBx. The external event selected can then be used as a trigger.

The event trigger is selected using TC_CMR.EEVT. The trigger edge (rising, falling or both) for each of the possible external triggers is defined in TC_CMR.EEVTEDG. 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 TC_CMR.ENETRG.

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.

37.6.13 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.

37.6.14 Quadrature Decoder

37.6.14.1 Description

The quadrature decoder (QDEC) is driven by TIOA0, TIOB0 and TIOB1 input pins and drives the timer counter of channel 0 and channel 1. Channel 2 can be used as a time base in case of speed measurement requirements (refer to Figure 37-15 ”Predefined Connection of the Quadrature Decoder with Timer Counters”).

When writing a 0 to TC_BMR.QDEN, 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.

TC_CMRx.TCCLKS 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.

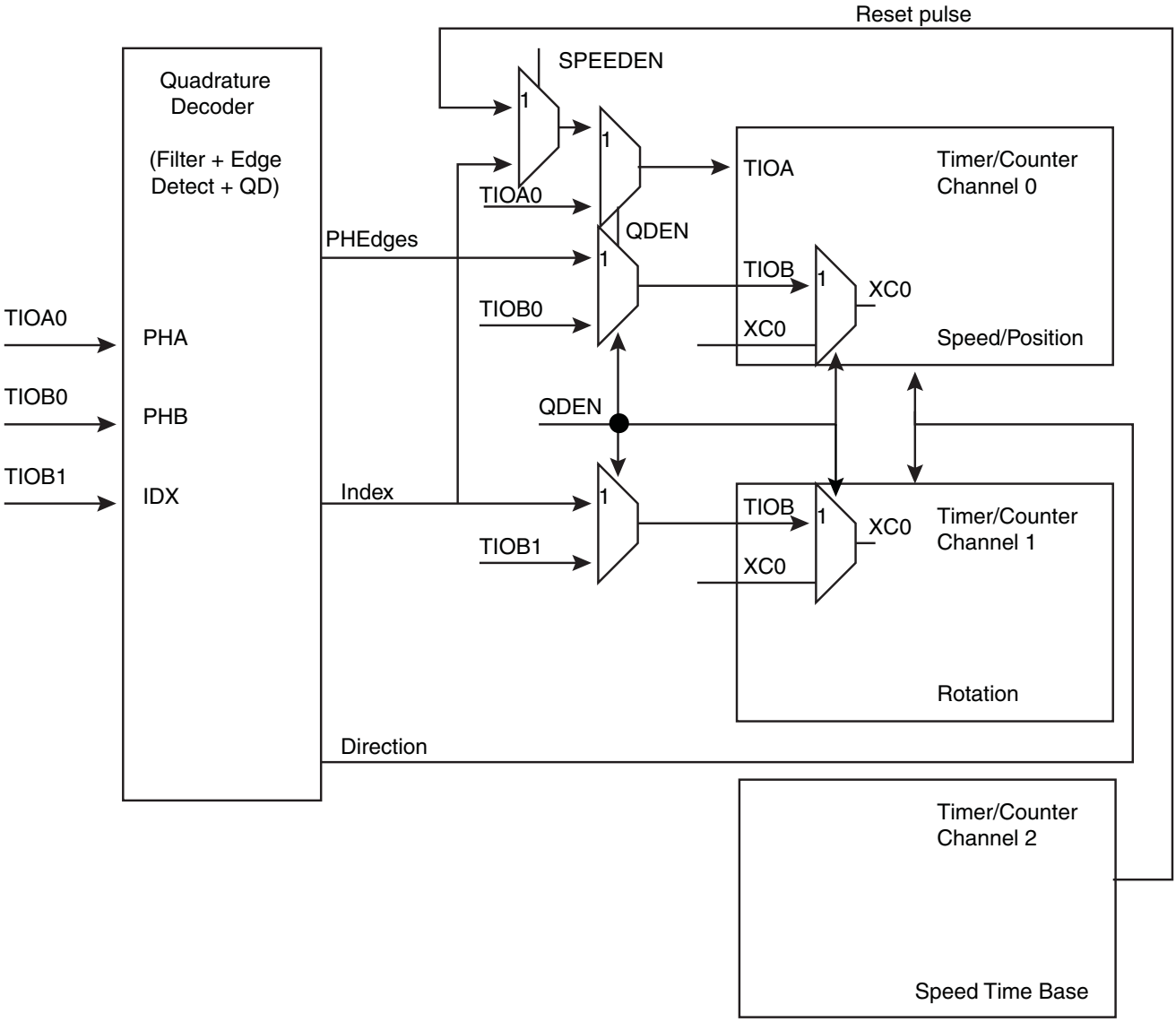
In Speed mode, position cannot be measured but revolution can be measured.

Inputs from the rotary sensor can be filtered prior to downstream 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 TC_SRx.CPCS.

Figure 37-15. Predefined Connection of the Quadrature Decoder with Timer Counters



37.6.14.2 Input Preprocessing

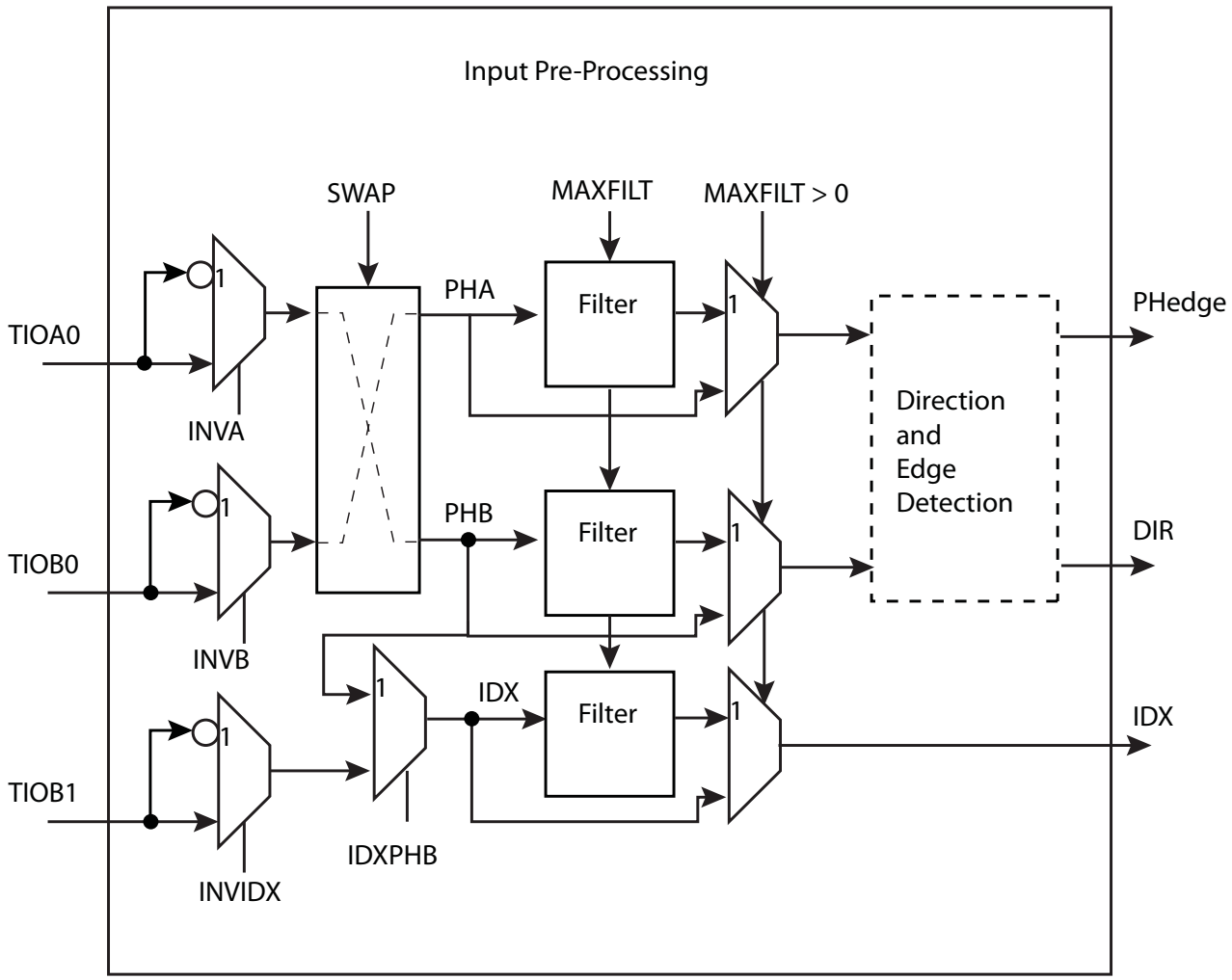
Input preprocessing consists of capabilities to take into account rotary sensor factors such as polarities and phase definition followed by configurable digital filtering.

Each input can be negated and swapping PHA, PHB is also configurable.

TC_BMR.MAXFILT 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 * $t_{peripheral\ clock}$ are not passed to downstream logic.

The value of (MAXFILT +1) * $t_{peripheral\ clock}$ must not be greater than 10% of the minimum pulse on PHA, PHB or index when the rotary encoder speed is at its maximun. This speed depends on the application.

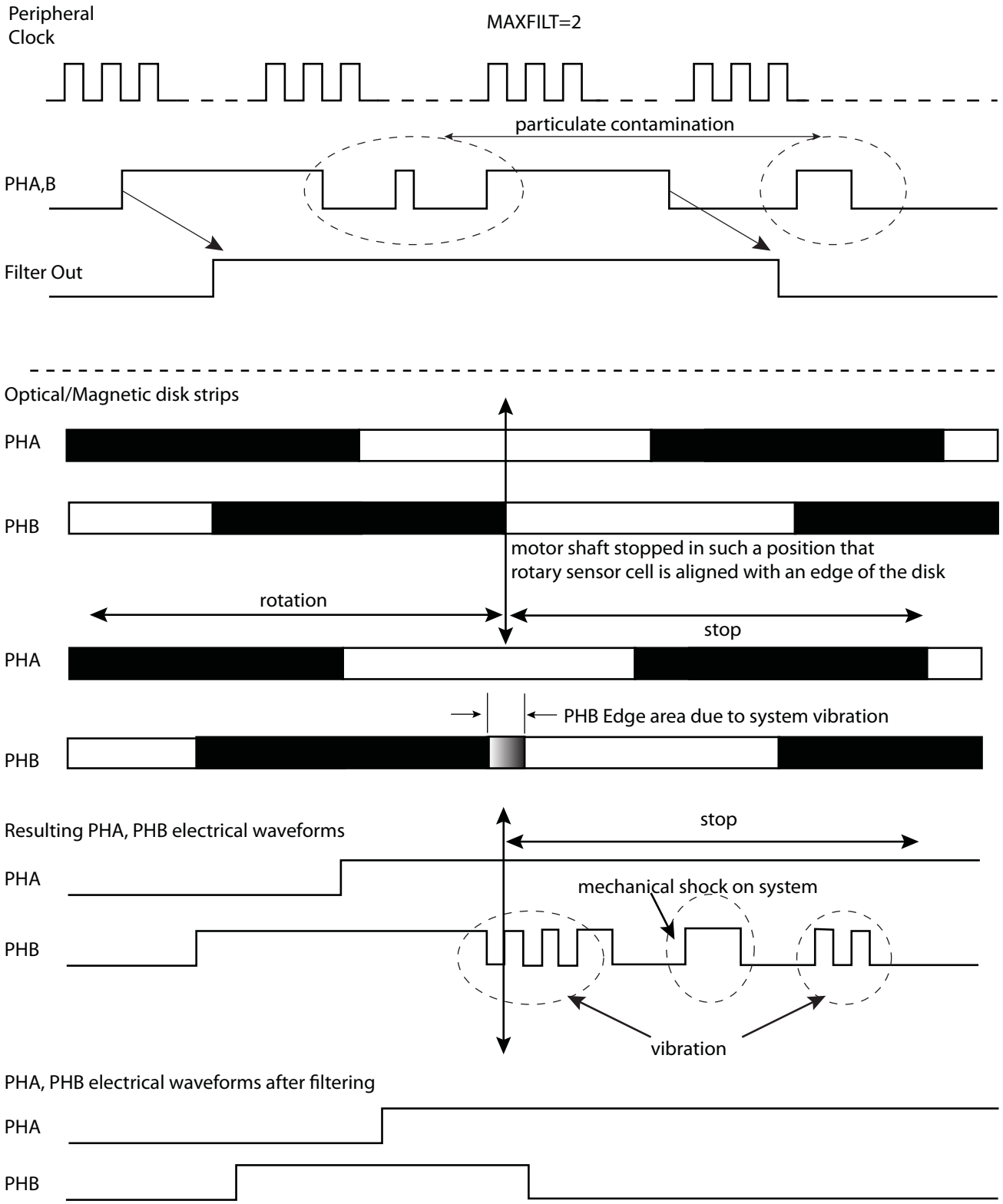
Figure 37-16. Input Stage



Input filtering can efficiently remove spurious pulses that might be generated by the presence of particulate contamination on the optical or magnetic disk of the rotary sensor.

Spurious pulses can also occur in environments with high levels of electromagnetic 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 37-17. Filtering Examples



37.6.14.3 Direction Status and Change Detection

After filtering, the quadrature signals are analyzed to extract the rotation direction and edges of the two quadrature signals detected in order to be counted by TC logic downstream.

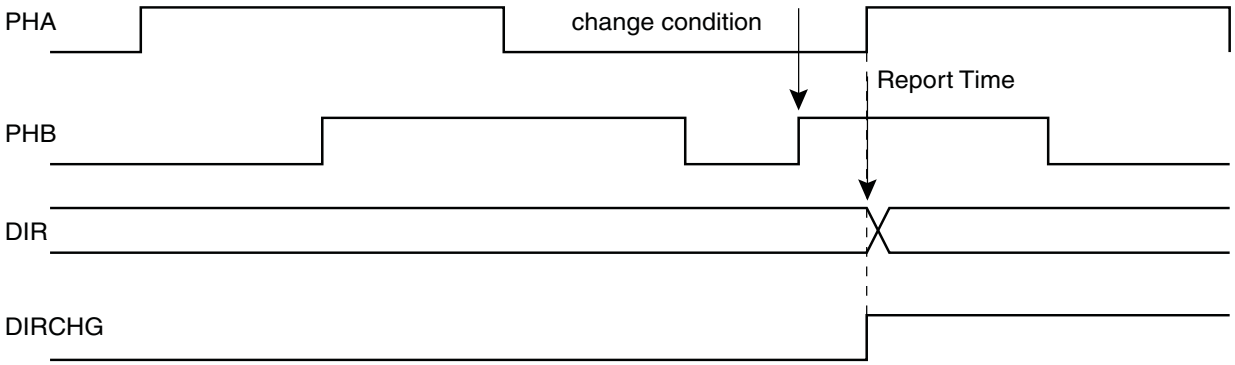
The direction status can be directly read at anytime in the TC_QISR. The polarity of the direction flag status depends on the configuration written in TC_BMR. INVA, INVB, INVIDX, SWAP modify the polarity of DIR flag.

Any change in rotation direction is reported in the TC_QISR and can generate an interrupt.

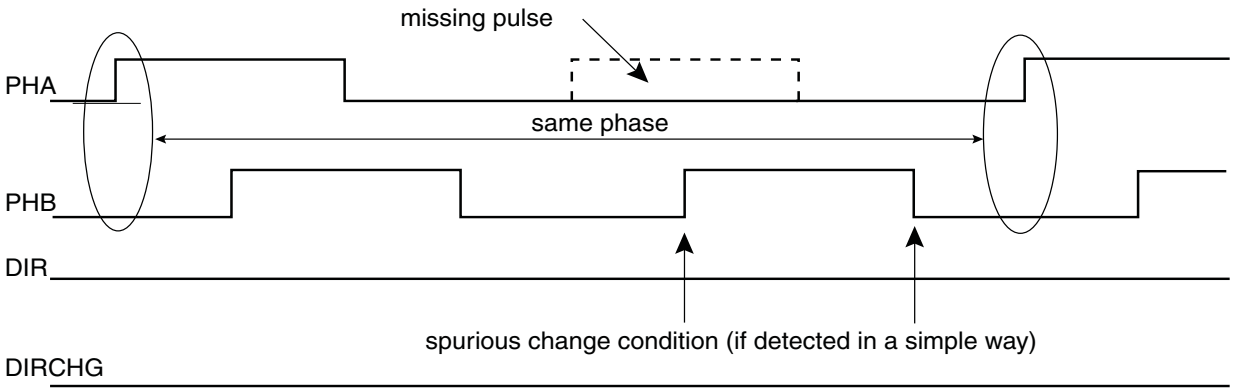
The direction change condition is reported as soon as two consecutive edges on a phase signal have sampled the same value on the other phase signal and there is an edge on the other signal. The two consecutive edges of one phase signal sampling the same value on other phase signal is not sufficient to declare a direction change, as particulate contamination may mask one or more reflective bars on the optical or magnetic disk of the sensor. Refer to Figure 37-18 ”Rotation Change Detection” for waveforms.

Figure 37-18. Rotation Change Detection

Direction Change under normal conditions



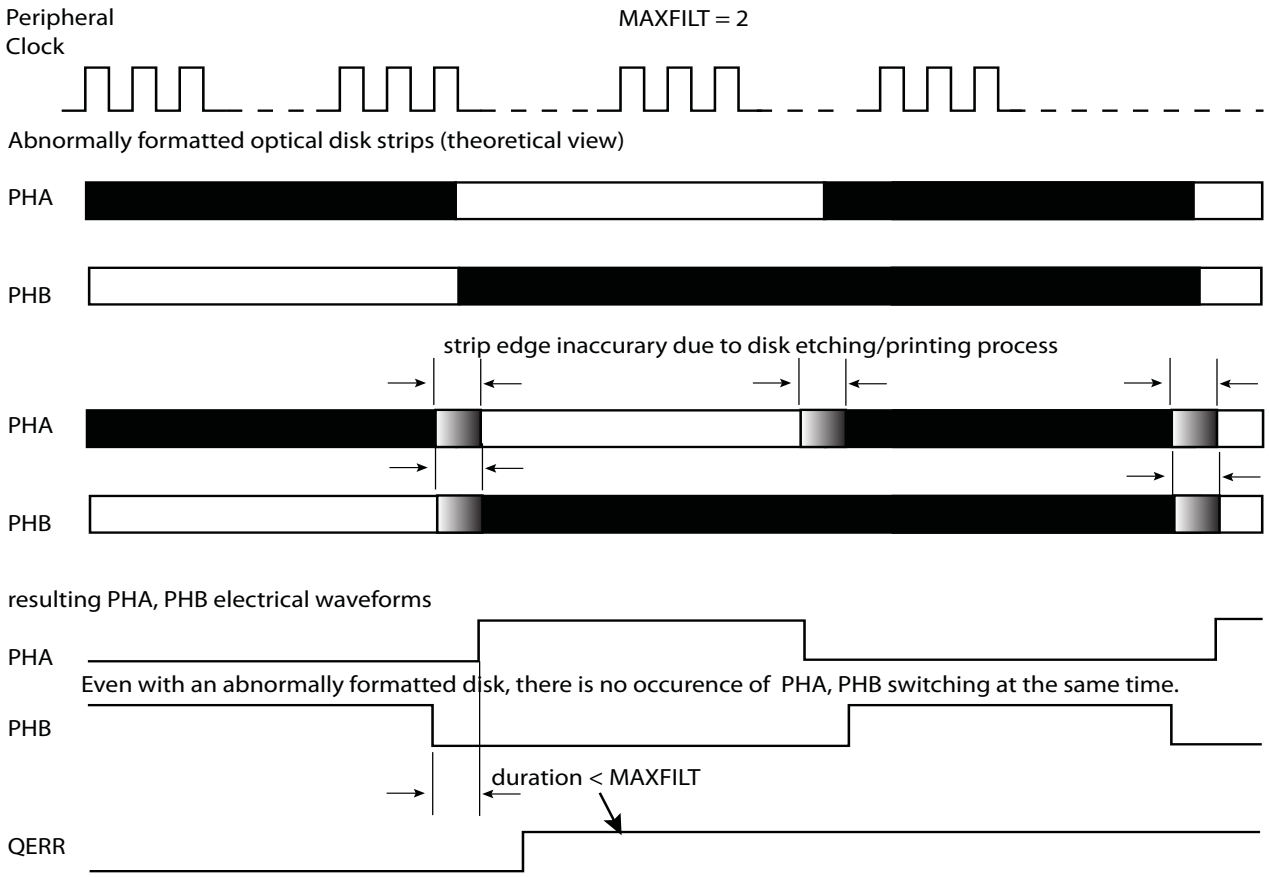
No direction change due to particulate contamination masking a reflective bar



The direction change detection is disabled when TC_BMR.QDTRANS is set. In this case, the DIR flag report must not be used.

A quadrature error is also reported by the QDEC via TC_QISR.QERR. 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 37-19. Quadrature Error Detection



MAXFILT must be tuned according to several factors such as the peripheral clock frequency, type of rotary sensor and rotation speed to be achieved.

37.6.14.4 Position and Rotation Measurement

When TC_BMR.POSEN is set, 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). The process must be started by configuring TC_CCR.CLKEN and TC_CCR.SWTRG.

In parallel, the number of edges are accumulated on TC 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 TC 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 TC channels 0 and 1. The direction status is reported on TC_QISR.

37.6.14.5 Speed Measurement

When TC_BMR.SPEEDEN is set, 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). TC_CMR0.ABETRG 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.

37.6.14.6 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=1026).

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.

The missing index pulse detection is only valid if the bit TC_QISR.DIRCHG = 0.

37.6.14.7 Detecting Contamination/Dust at Rotary Encoder Low Speed

The contamination/dust that can be filtered when the rotary encoder speed is high may not be filtered at low speed, thus creating unsollicited direction change, etc.

At low speed, even a minor contamination may appear as a long pulse, and thus not filtered and processed as a standard quadrature encoder pulse.

This contamination can be detected by using the similar method as the missing index detection.

A contamination exists on a phase line if TC_SR.CPCS = 1 and TC_QISR.DIRCHG = 1 when there is no sollicited change of direction.

37.6.15 2-bit Gray Up/Down Counter for Stepper Motor

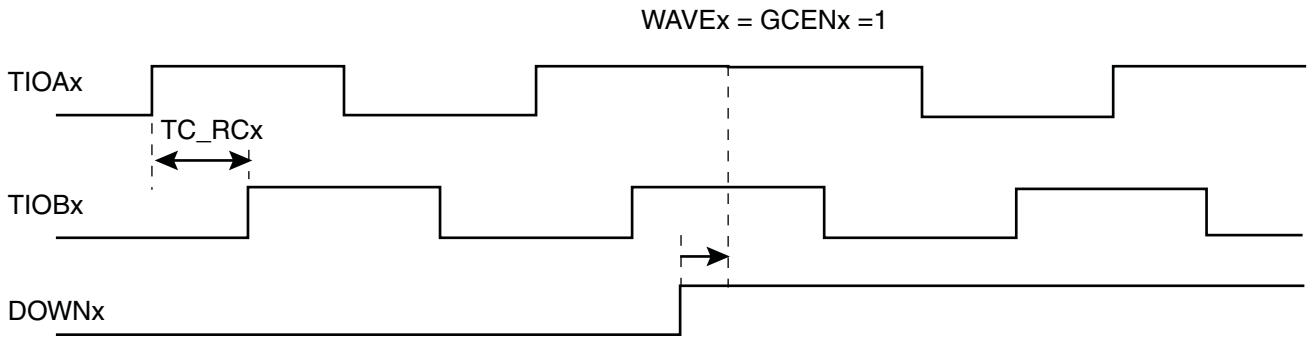
Each channel can be independently configured to generate a 2-bit Gray count waveform on corresponding TIOAx, TIOBx outputs by means of TC_SMMRx.GCEN.

Up or Down count can be defined by writing TC_SMMRx.DOWN.

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 37-20. 2-bit Gray Up/Down Counter



37.6.16 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 Stepper Motor Mode Register"
- "TC Register A"
- "TC Register B"
- "TC Register C"

37.7 Timer Counter (TC) User Interface

Table 37-6. Register Mapping

Offset[1]	Register	Name	Access	Reset
0x00 + channel * 0x40 + 0x00	Channel Control Register	TC_CCR	Write-only	–
0x00 + channel * 0x40 + 0x04	Channel Mode Register	TC_CMR	Read/Write	0
0x00 + channel * 0x40 + 0x08	Stepper Motor Mode Register	TC_SMMR	Read/Write	0
0x00 + channel * 0x40 + 0x0C	Reserved	–	–	–
0x00 + channel * 0x40 + 0x10	Counter Value	TC_CV	Read-only	0
0x00 + channel * 0x40 + 0x14	Register A	TC_RA	Read/Write[2]	0
0x00 + channel * 0x40 + 0x18	Register B	TC_RB	Read/Write[2]	0
0x00 + channel * 0x40 + 0x1C	Register C	TC_RC	Read/Write	0
0x00 + channel * 0x40 + 0x20	Status Register	TC_SR	Read-only	0
0x00 + channel * 0x40 + 0x24	Interrupt Enable Register	TC_IER	Write-only	–
0x00 + channel * 0x40 + 0x28	Interrupt Disable Register	TC_IDR	Write-only	–
0x00 + channel * 0x40 + 0x2C	Interrupt Mask Register	TC_IMR	Read-only	0
0xC0	Block Control Register	TC_BCR	Write-only	–
0xC4	Block Mode Register	TC_BMR	Read/Write	0
0xC8	QDEC Interrupt Enable Register	TC_QIER	Write-only	–
0xCC	QDEC Interrupt Disable Register	TC_QIDR	Write-only	–
0xD0	QDEC Interrupt Mask Register	TC_QIMR	Read-only	0
0xD4	QDEC Interrupt Status Register	TC_QISR	Read-only	0
0xD8	Reserved	–	–	–
0xE4	Write Protection Mode Register	TC_WPMR	Read/Write	0
0xE8 - 0xFC	Reserved	–	–	–

Notes: 1. Channel index ranges from 0 to 2.
2. Read-only if TC_CMRx.WAVE = 0.

37.7.1 TC Channel Control Register

Name: TC_CCRx [x=0..2]

Address: 0x40010000 (0)[0], 0x40010040 (0)[1], 0x40010080 (0)[2], 0x40014000 (1)[0], 0x40014040 (1)[1], 0x40014080 (1)[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.

37.7.2 TC Channel Mode Register: Capture Mode

Name: TC_CMRx [x=0..2] (CAPTURE_MODE)

Address: 0x40010004 (0)[0], 0x40010044 (0)[1], 0x40010084 (0)[2], 0x40014004 (1)[0], 0x40014044 (1)[1], 0x40014084 (1)[2]

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	LDRB		LDRA	
15	14	13	12	11	10	9	8
WAVE	CPCTRG	–	–	–	ABETRG	ETRGEDG	
7	6	5	4	3	2	1	0
LDBDIS	LDBSTOP	BURST		CLKI	TCCLKS		

This register can only be written if the WPEN bit is cleared in the "TC Write Protection Mode Register" .

- **TCCLKS: Clock Selection**

Value	Name	Description
0	TIMER_CLOCK1	Clock selected: internal MCK/2 clock signal (from PMC).
1	TIMER_CLOCK2	Clock selected: internal MCK/8 clock signal (from PMC).
2	TIMER_CLOCK3	Clock selected: internal MCK/32 clock signal (from PMC).
3	TIMER_CLOCK4	Clock selected: internal MCK/128 clock signal (from PMC).
4	TIMER_CLOCK5	Clock selected: internal SLCK clock signal (from PMC).
5	XC0	Clock selected: XC0.
6	XC1	Clock selected: XC1.
7	XC2	Clock selected: XC2.

- **CLKI: Clock Invert**

0 = Counter is incremented on rising edge of the clock.

1 = Counter is incremented on falling edge of the clock.

- **BURST: Burst Signal Selection**

Value	Name	Description
0	NONE	The clock is not gated by an external signal.
1	XC0	XC0 is ANDed with the selected clock.
2	XC1	XC1 is ANDed with the selected clock.
3	XC2	XC2 is ANDed with the selected clock.

- **LDBSTOP: Counter Clock Stopped with RB Loading**

0 = Counter clock is not stopped when RB loading occurs.

1 = Counter clock is stopped when RB loading occurs.

- **LDBDIS: Counter Clock Disable with RB Loading**

0 = Counter clock is not disabled when RB loading occurs.

1 = Counter clock is disabled when RB loading occurs.

- **ETRGEDG: External Trigger Edge Selection**

Value	Name	Description
0	NONE	The clock is not gated by an external signal.
1	RISING	Rising edge.
2	FALLING	Falling edge.
3	EDGE	Each edge.

- **ABETRG: 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.

37.7.3 TC Channel Mode Register: Waveform Mode

Name: TC_CMRx [x=0..2] (WAVEFORM_MODE)

Access: Read/Write

31	30	29	28	27	26	25	24
BSWTRG		BEEVT		BCPC		BCPB	

23	22	21	20	19	18	17	16
ASWTRG		AEEVT		ACPC		ACPA	

15	14	13	12	11	10	9	8
WAVE	WAVSEL		ENETRG	EEVT		EEVTEDG	

7	6	5	4	3	2	1	0
CPCDIS	CPCSTOP	BURST		CLKI	TCCLKS		

This register can only be written if the WPEN bit is cleared in the “TC Write Protection Mode Register” .

- **TCCLKS: Clock Selection**

Value	Name	Description
0	TIMER_CLOCK1	Clock selected: internal MCK/2 clock signal (from PMC).
1	TIMER_CLOCK2	Clock selected: internal MCK/8 clock signal (from PMC).
2	TIMER_CLOCK3	Clock selected: internal MCK/32 clock signal (from PMC).
3	TIMER_CLOCK4	Clock selected: internal MCK/128 clock signal (from PMC).
4	TIMER_CLOCK5	Clock selected: internal SLCK clock signal (from PMC).
5	XC0	Clock selected: XC0.
6	XC1	Clock selected: XC1.
7	XC2	Clock selected: XC2.

- **CLKI: Clock Invert**

0 = Counter is incremented on rising edge of the clock.

1 = Counter is incremented on falling edge of the clock.

- **BURST: Burst Signal Selection**

Value	Name	Description
0	NONE	The clock is not gated by an external signal.
1	XC0	XC0 is ANDed with the selected clock.
2	XC1	XC1 is ANDed with the selected clock.
3	XC2	XC2 is ANDed with the selected clock.

- **CPCSTOP: Counter Clock Stopped with RC Compare**

0 = Counter clock is not stopped when counter reaches RC.

1 = Counter clock is stopped when counter reaches RC.

- **CPCDIS: Counter Clock Disable with RC Compare**

0 = Counter clock is not disabled when counter reaches RC.

1 = Counter clock is disabled when counter reaches RC.

- **EEVTEDG: External Event Edge Selection**

Value	Name	Description
0	NONE	None
1	RISING	Rising edge
2	FALLING	Falling edge
3	EDGE	Each edge

- **EEVT: External Event Selection**

Signal selected as external event.

Value	Name	Description	TIOB Direction
0	TIOB	TIOB(1)	Input
1	XC0	XC0	Output
2	XC1	XC1	Output
3	XC2	XC2	Output

Notes: 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

37.7.4 TC Stepper Motor Mode Register

Name: TC_SMMRx [x=0..2]

Address: 0x40010008 (0)[0], 0x40010048 (0)[1], 0x40010088 (0)[2], 0x40014008 (1)[0], 0x40014048 (1)[1], 0x40014088 (1)[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.

37.7.5 TC Counter Value Register

Name: TC_CVx [x=0..2]

Address: 0x40010010 (0)[0], 0x40010050 (0)[1], 0x40010090 (0)[2], 0x40014010 (1)[0], 0x40014050 (1)[1], 0x40014090 (1)[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.

IMPORTANT: For 16-bit channels, CV field size is limited to register bits 15:0.

37.7.6 TC Register A

Name: TC_RAx [x=0..2]

Address: 0x40010014 (0)[0], 0x40010054 (0)[1], 0x40010094 (0)[2], 0x40014014 (1)[0], 0x40014054 (1)[1], 0x40014094 (1)[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.

IMPORTANT: For 16-bit channels, RA field size is limited to register bits 15:0.

37.7.7 TC Register B

Name: TC_RBx [x=0..2]

Address: 0x40010018 (0)[0], 0x40010058 (0)[1], 0x40010098 (0)[2], 0x40014018 (1)[0], 0x40014058 (1)[1], 0x40014098 (1)[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.

IMPORTANT: For 16-bit channels, RB field size is limited to register bits 15:0.

37.7.8 TC Register C

Name: TC_RCx [x=0..2]

Address: 0x4001001C (0)[0], 0x4001005C (0)[1], 0x4001009C (0)[2], 0x4001401C (1)[0], 0x4001405C (1)[1], 0x4001409C (1)[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.

IMPORTANT: For 16-bit channels, RC field size is limited to register bits 15:0.

37.7.9 TC Status Register

Name: TC_SRx [x=0..2]

Address: 0x40010020 (0)[0], 0x40010060 (0)[1], 0x400100A0 (0)[2], 0x40014020 (1)[0], 0x40014060 (1)[1], 0x400140A0 (1)[2]

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	MTIOB	MTIOA	CLKSTA
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow Status (cleared on read)**

0 = No counter overflow has occurred since the last read of the Status Register.

1 = A counter overflow has occurred since the last read of the Status Register.

- **LOVRS: Load Overrun Status (cleared on read)**

0 = Load overrun has not occurred since the last read of the Status Register or TC_CMRx.WAVE = 1.

1 = RA or RB have been loaded at least twice without any read of the corresponding register since the last read of the Status Register, if TC_CMRx.WAVE = 0.

- **CPAS: RA Compare Status (cleared on read)**

0 = RA Compare has not occurred since the last read of the Status Register or TC_CMRx.WAVE = 0.

1 = RA Compare has occurred since the last read of the Status Register, if TC_CMRx.WAVE = 1.

- **CPBS: RB Compare Status (cleared on read)**

0 = RB Compare has not occurred since the last read of the Status Register or TC_CMRx.WAVE = 0.

1 = RB Compare has occurred since the last read of the Status Register, if TC_CMRx.WAVE = 1.

- **CPCS: RC Compare Status (cleared on read)**

0 = RC Compare has not occurred since the last read of the Status Register.

1 = RC Compare has occurred since the last read of the Status Register.

- **LDRAS: RA Loading Status (cleared on read)**

0 = RA Load has not occurred since the last read of the Status Register or TC_CMRx.WAVE = 1.

1 = RA Load has occurred since the last read of the Status Register, if TC_CMRx.WAVE = 0.

- **LDRBS: RB Loading Status (cleared on read)**

0 = RB Load has not occurred since the last read of the Status Register or TC_CMRx.WAVE = 1.

1 = RB Load has occurred since the last read of the Status Register, if TC_CMRx.WAVE = 0.

- **ETRGS: External Trigger Status (cleared on read)**

0 = External trigger has not occurred since the last read of the Status Register.

1 = External trigger has occurred since the last read of the Status Register.

- **CLKSTA: Clock Enabling Status**

0 = Clock is disabled.

1 = Clock is enabled.

- **MTIOA: TIOAx Mirror**

0 = TIOAx is low. If TC_CMRx.WAVE = 0, TIOAx pin is low. If TC_CMRx.WAVE = 1, TIOAx is driven low.

1 = TIOAx is high. If TC_CMRx.WAVE = 0, TIOAx pin is high. If TC_CMRx.WAVE = 1, TIOAx is driven high.

- **MTIOB: TIOBx Mirror**

0 = TIOBx is low. If TC_CMRx.WAVE = 0, TIOBx pin is low. If TC_CMRx.WAVE = 1, TIOBx is driven low.

1 = TIOBx is high. If TC_CMRx.WAVE = 0, TIOBx pin is high. If TC_CMRx.WAVE = 1, TIOBx is driven high.

37.7.10 TC Interrupt Enable Register

Name: TC_IERx [x=0..2]

Address: 0x40010024 (0)[0], 0x40010064 (0)[1], 0x400100A4 (0)[2], 0x40014024 (1)[0], 0x40014064 (1)[1], 0x400140A4 (1)[2]

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow**

0 = No effect.

1 = Enables the Counter Overflow Interrupt.

- **LOVRS: Load Overrun**

0 = No effect.

1 = Enables the Load Overrun Interrupt.

- **CPAS: RA Compare**

0 = No effect.

1 = Enables the RA Compare Interrupt.

- **CPBS: RB Compare**

0 = No effect.

1 = Enables the RB Compare Interrupt.

- **CPCS: RC Compare**

0 = No effect.

1 = Enables the RC Compare Interrupt.

- **LDRAS: RA Loading**

0 = No effect.

1 = Enables the RA Load Interrupt.

- **LDRBS: RB Loading**

0 = No effect.

1 = Enables the RB Load Interrupt.

- **ETRGS: External Trigger**

0 = No effect.

1 = Enables the External Trigger Interrupt.

37.7.11 TC Interrupt Disable Register

Name: TC_IDRx [x=0..2]

Address: 0x40010028 (0)[0], 0x40010068 (0)[1], 0x400100A8 (0)[2], 0x40014028 (1)[0], 0x40014068 (1)[1], 0x400140A8 (1)[2]

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow**

0 = No effect.

1 = Disables the Counter Overflow Interrupt.

- **LOVRS: Load Overrun**

0 = No effect.

1 = Disables the Load Overrun Interrupt (if TC_CMRx.WAVE = 0).

- **CPAS: RA Compare**

0 = No effect.

1 = Disables the RA Compare Interrupt (if TC_CMRx.WAVE = 1).

- **CPBS: RB Compare**

0 = No effect.

1 = Disables the RB Compare Interrupt (if TC_CMRx.WAVE = 1).

- **CPCS: RC Compare**

0 = No effect.

1 = Disables the RC Compare Interrupt.

- **LDRAS: RA Loading**

0 = No effect.

1 = Disables the RA Load Interrupt (if TC_CMRx.WAVE = 0).

- **LDRBS: RB Loading**

0 = No effect.

1 = Disables the RB Load Interrupt (if TC_CMRx.WAVE = 0).

- **ETRGS: External Trigger**

0 = No effect.

1 = Disables the External Trigger Interrupt.

37.7.12 TC Interrupt Mask Register

Name: TC_IMRx [x=0..2]

Address: 0x4001002C (0)[0], 0x4001006C (0)[1], 0x400100AC (0)[2], 0x4001402C (1)[0], 0x4001406C (1)[1], 0x400140AC (1)[2]

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow**

0 = The Counter Overflow Interrupt is disabled.

1 = The Counter Overflow Interrupt is enabled.

- **LOVRS: Load Overrun**

0 = The Load Overrun Interrupt is disabled.

1 = The Load Overrun Interrupt is enabled.

- **CPAS: RA Compare**

0 = The RA Compare Interrupt is disabled.

1 = The RA Compare Interrupt is enabled.

- **CPBS: RB Compare**

0 = The RB Compare Interrupt is disabled.

1 = The RB Compare Interrupt is enabled.

- **CPCS: RC Compare**

0 = The RC Compare Interrupt is disabled.

1 = The RC Compare Interrupt is enabled.

- **LDRAS: RA Loading**

0 = The Load RA Interrupt is disabled.

1 = The Load RA Interrupt is enabled.

- **LDRBS: RB Loading**

0 = The Load RB Interrupt is disabled.

1 = The Load RB Interrupt is enabled.

- **ETRGS: External Trigger**

0 = The External Trigger Interrupt is disabled.

1 = The External Trigger Interrupt is enabled.

37.7.13 TC Block Control Register

Name: TC_BCR

Address: 0x400100C0 (0), 0x400140C0 (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
–	–	–	–	–	–	–	SYNC

- **SYNC: Synchro Command**

0 = No effect.

1 = Asserts the SYNC signal which generates a software trigger simultaneously for each of the channels.

37.7.14 TC Block Mode Register

Name: TC_BMR

Address: 0x400100C4 (0), 0x400140C4 (1)

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.For more details on MAXFILT constraints, see Section 37.6.14.2 ”Input Preprocessing”.

37.7.15 TC QDEC Interrupt Enable Register

Name: TC_QIER

Address: 0x400100C8 (0), 0x400140C8 (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
–	–	–	–	–	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.

37.7.16 TC QDEC Interrupt Disable Register

Name: TC_QIDR

Address: 0x400100CC (0), 0x400140CC (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
–	–	–	–	–	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.

37.7.17 TC QDEC Interrupt Mask Register

Name: TC_QIMR

Address: 0x400100D0 (0), 0x400140D0 (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
–	–	–	–	–	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.

37.7.18 TC QDEC Interrupt Status Register

Name: TC_QISR

Address: 0x400100D4 (0), 0x400140D4 (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
–	–	–	–	–	–	–	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.

37.7.19 TC Write Protection Mode Register

Name: TC_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 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 37.6.16 ”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.

38. Pulse Width Modulation Controller (PWM)

38.1 Description

The Pulse Width Modulation Controller (PWM) controls several channels independently. Each channel controls one square output waveform. Characteristics of the output waveform such as period, duty-cycle and polarity are configurable 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 macrocell master clock.

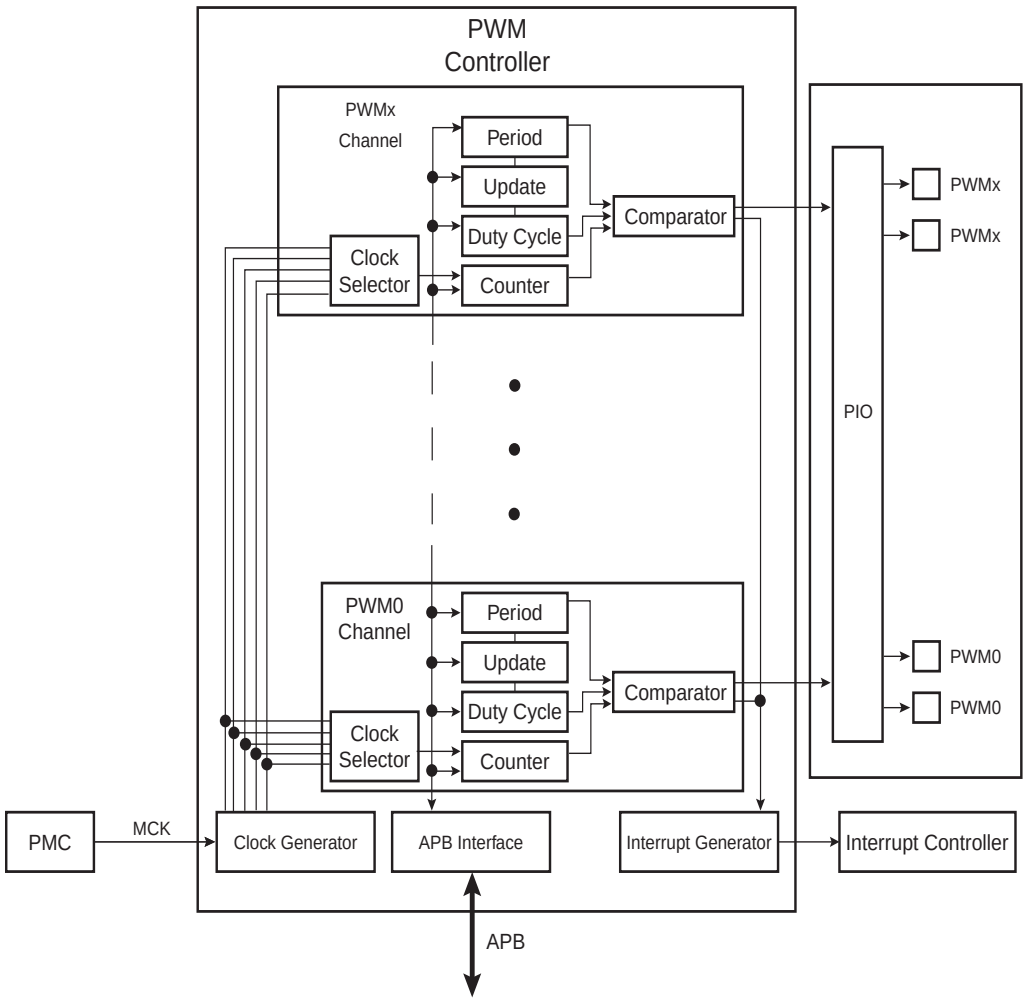
Channels can be synchronized, to generate non overlapped waveforms. All channels integrate a double buffering system in order to prevent an unexpected output waveform while modifying the period or the duty-cycle.

38.2 Embedded Characteristics

- 4 Channels
- One 16-bit Counter Per Channel
- 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 Enable Disable Command for Each Channel
  - Independent Clock Selection for Each Channel
  - Independent Period and Duty Cycle for Each Channel
  - Double Buffering of Period or Duty Cycle for Each Channel
  - Programmable Selection of The Output Waveform Polarity for Each Channel
  - Programmable Center or Left Aligned Output Waveform for Each Channel Block Diagram

38.3 Block Diagram

Figure 38-1. Pulse Width Modulation Controller Block Diagram



38.4 I/O Lines Description

Each channel outputs one waveform on one external I/O line.

Table 38-1. I/O Line Description

Name	Description	Type
PWMx	PWM Waveform Output for channel x	Output

38.5 Product Dependencies

38.5.1 I/O Lines

The pins used for interfacing the PWM may be multiplexed with PIO lines. The programmer must first program the PIO controller to assign the desired PWM pins to their peripheral function. If I/O lines of the PWM are not used by the application, they can be used for other purposes by the PIO controller.

All of the PWM outputs may or may not be enabled. If an application requires only four channels, then only four PIO lines will be assigned to PWM outputs.

Table 38-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
PWM	PWM0	PC0	B
PWM	PWM0	PC6	A
PWM	PWM1	PC1	B
PWM	PWM1	PC7	A
PWM	PWM2	PC2	B
PWM	PWM2	PC8	A
PWM	PWM3	PC3	B
PWM	PWM3	PC9	A

38.5.2 Power Management

The PWM is not continuously clocked. The programmer must first enable the PWM clock in the Power Management Controller (PMC) before using the PWM. However, if the application does not require PWM operations, the PWM clock can be stopped when not needed and be restarted later. In this case, the PWM will resume its operations where it left off.

All the PWM registers except PWM Channel Duty Cycle Register (PWM_CDTYx) and PWM Channel Period Register (PWM_CPRDx) can be read without the PWM peripheral clock enabled. All the registers can be written without the peripheral clock enabled.

38.5.3 Interrupt Sources

The PWM interrupt line is connected on one of the internal sources of the Interrupt Controller. Using the PWM interrupt requires the Interrupt Controller to be programmed first. Note that it is not recommended to use the PWM interrupt line in edge sensitive mode.

Table 38-3. Peripheral IDs

Instance	ID
PWM	41

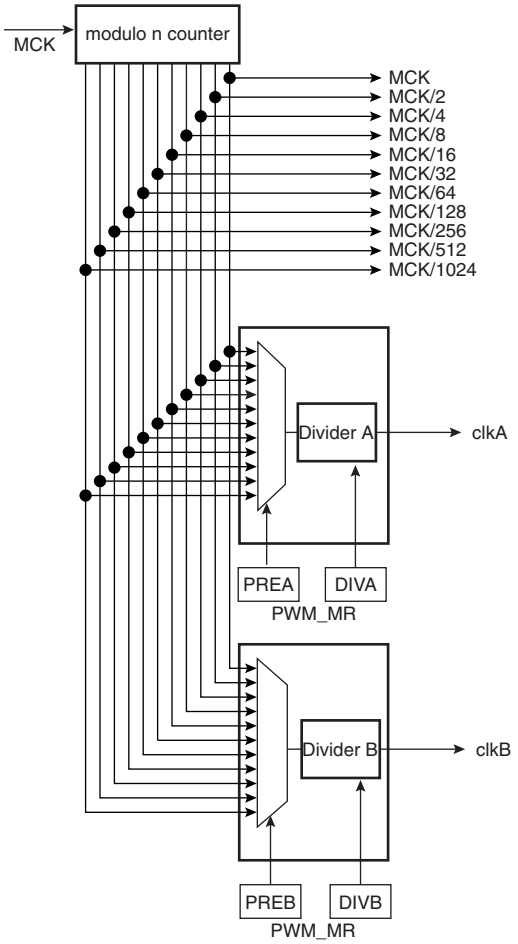
38.6 Functional Description

The PWM macrocell is primarily composed of a clock generator module and 4 channels.

- Clocked by the system clock, MCK, the clock generator module provides 13 clocks.
- Each channel can independently choose one of the clock generator outputs.
- Each channel generates an output waveform with attributes that can be defined independently for each channel through the user interface registers.

38.6.1 PWM Clock Generator

Figure 38-2. Functional View of the Clock Generator Block Diagram



Caution: Before using the PWM macrocell, the programmer must first enable the PWM clock in the Power Management Controller (PMC).

The PWM macrocell master clock, MCK, is divided in the clock generator module to provide different clocks available for all channels. Each channel can independently select one of the divided clocks.
The clock generator is divided in three blocks:

- A modulo n counter which provides 11 clocks: f_{MCK}, $f_{MCK}/2$, $f_{MCK}/4$, $f_{MCK}/8$, $f_{MCK}/16$, $f_{MCK}/32$, $f_{MCK}/64$, $f_{MCK}/128$, $f_{MCK}/256$, $f_{MCK}/512$, $f_{MCK}/1024$.
- Two linear dividers (1, 1/2, 1/3,... 1/255) that provide two separate clocks: clkA and clkB.

Each linear divider can independently divide one of the clocks of the modulo n counter. The selection of the clock to be divided is made according to the PREA (PREB) field of the PWM Mode Register (PWM_MR). The resulting clock clkA (clkB) is the clock selected divided by DIVA (DIVB) field value in PWM_MR.

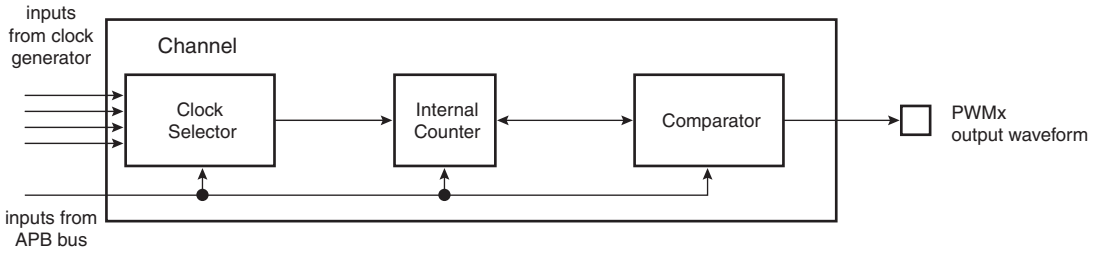
After a reset of the PWM controller, DIVA (DIVB) and PREA (PREB) in PWM_MR are set to 0. This implies that after reset clkA (clkB) are turned off.

At reset, all clocks provided by the modulo n counter are turned off except clock “clk”. This situation is also true when the PWM master clock is turned off through the Power Management Controller.

38.6.2 PWM Channel

38.6.2.1 Block Diagram

Figure 38-3. Functional View of the Channel Block Diagram



Each of the 4 channels is composed of three blocks:

- A clock selector which selects one of the clocks provided by the clock generator described in Section 38.6.1 “PWM Clock Generator” on page 769.
- An internal counter clocked by the output of the clock selector. This internal counter is incremented or decremented according to the channel configuration and comparators events. The size of the internal counter is 16 bits.
- A comparator used to generate events according to the internal counter value. It also computes the PWMx output waveform according to the configuration.

38.6.2.2 Waveform Properties

The different properties of output waveforms are:

- the ***internal clock selection***. The internal 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 PWM_CPRDx.
  If the waveform is left aligned, then the output waveform period depends on the counter source clock and can be calculated:
  By using the Master Clock (MCK) divided by an X given prescaler value (with X being 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024), the resulting period formula will be:

  $$\frac{(X*CPRD)}{MCK}$$

  By using a Master Clock divided by one of both DIVA or DIVB divider, the formula becomes, respectively:

  $$\frac{(X*CPRD*DIVA)}{MCK} \quad or \quad \frac{(X*CPRD*DIVB)}{MCK}$$

  If the waveform is center aligned then the output waveform period depends on the counter source clock and can be calculated:
  By using the Master Clock (MCK) divided by an X given prescaler value (with X being 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024). The resulting period formula will be:

  $$\frac{(2*X*CPRD)}{MCK}$$

  By using a Master Clock divided by one of both DIVA or DIVB divider, the formula becomes, respectively:

  $$\frac{(2*X*CPRD*DIVA)}{MCK} \quad or \quad \frac{(2*X*CPRD*DIVB)}{MCK}$$

- the ***waveform duty cycle***. This channel parameter is defined in the CDTY field of PWM_CDTYx.
  If the waveform is left aligned then:

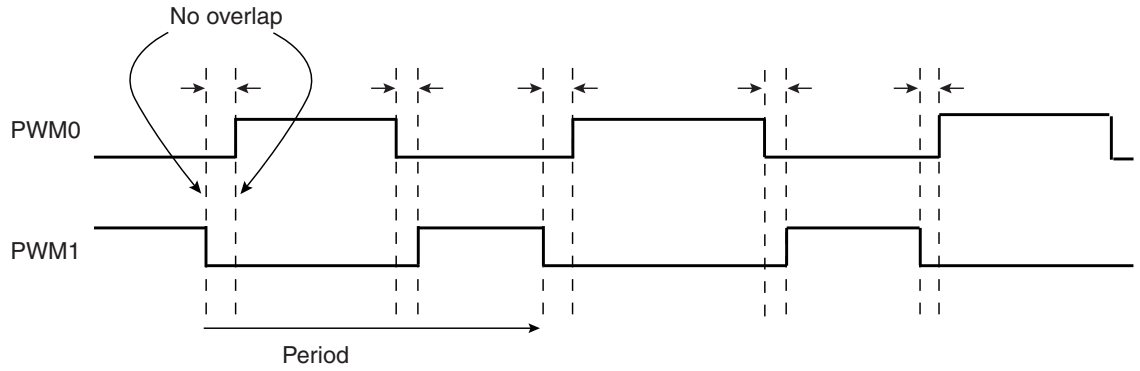
$$\text{duty cycle} = \frac{(period - 1/\text{fchannel\_x\_clock} \times CDTY)}{period}$$

  If the waveform is center aligned, then:

$$\text{duty cycle} = \frac{((period/2) - 1/\text{fchannel\_x\_clock} \times CDTY)}{(period/2)}$$

- the ***waveform polarity.*** At the beginning of the period, the signal can be at high or low level. This property is defined in the CPOL field of the PWM_CMRx. By default the signal starts by a low level.
- the ***waveform alignment***. The output waveform can be left or center aligned. Center aligned waveforms can be used to generate non overlapped waveforms. This property is defined in the CALG field of the PWM_CMRx. The default mode is left aligned.

Figure 38-4. Non Overlapped Center Aligned Waveforms



Note: 1. See Figure 38-5 for a detailed description of center aligned waveforms.

When center aligned, the internal channel counter increases up to CPRD and.decreases down to 0. This ends the period.

When left aligned, the internal channel counter increases up to CPRD and is reset. This ends the period.

Thus, for the same CPRD value, the period for a center aligned channel is twice the period for a left aligned channel.

Waveforms are fixed at 0 when:

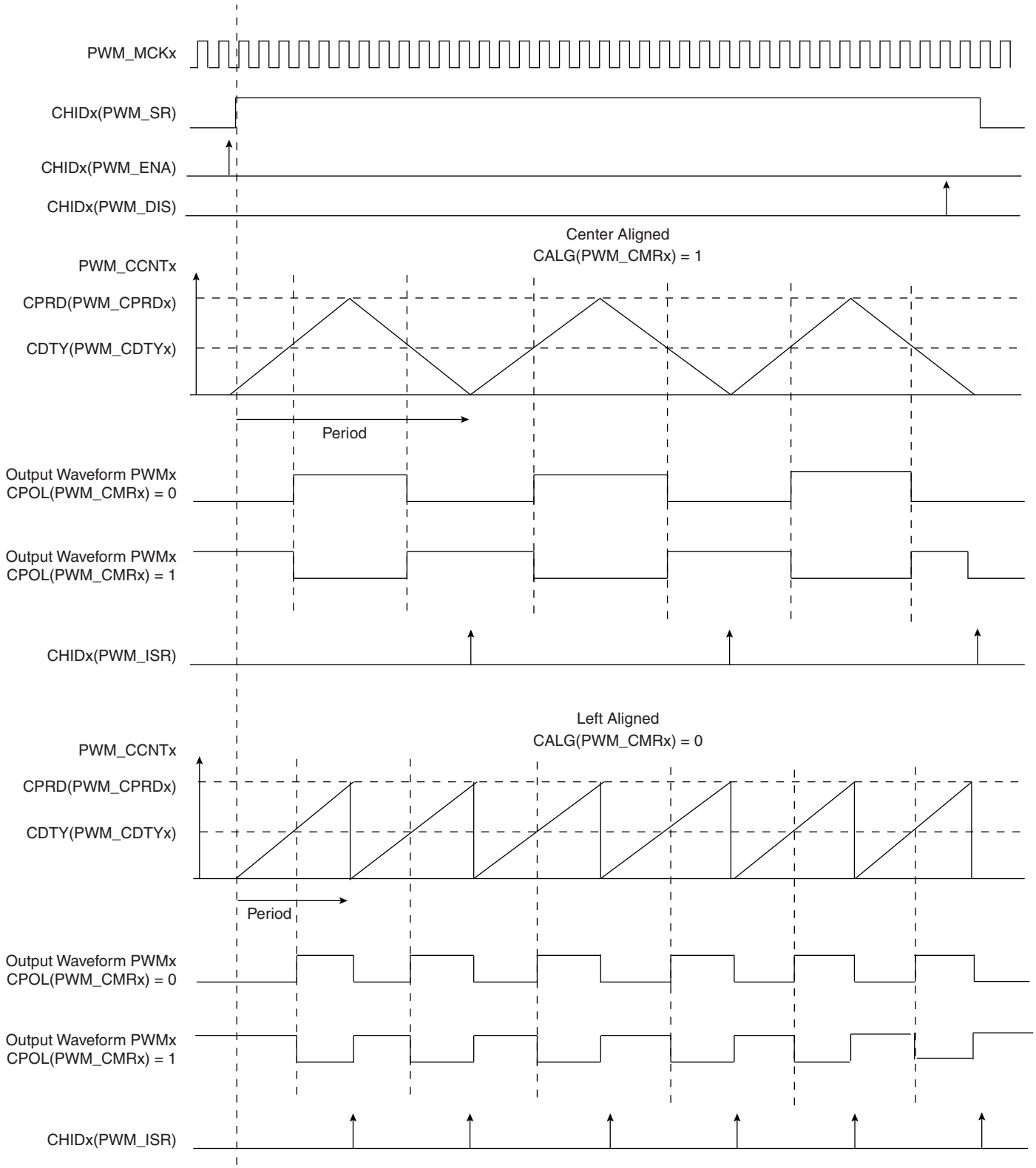
- CDTY = CPRD and CPOL = 0
- CDTY = 0 and CPOL = 1

Waveforms are fixed at 1 (once the channel is enabled) when:

- CDTY = 0 and CPOL = 0
- CDTY = CPRD and CPOL = 1

The waveform polarity must be set before enabling the channel. This immediately affects the channel output level. Changes on channel polarity are not taken into account while the channel is enabled.

Figure 38-5. Waveform Properties



38.6.3 PWM Controller Operations

38.6.3.1 Initialization

Before enabling the output channel, this channel must have been configured by the software application:

- Configuration of the clock generator if DIVA and DIVB are 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).
- Configuration of the period for each channel (CPRD in PWM_CPRDx). Writing in PWM_CPRDx is possible while the channel is disabled. After validation of the channel, the user must use the PWM Channel Update Register (PWM_CUPDx) to update PWM_CPRDx as explained below.
- Configuration of the duty cycle for each channel (CDTY in PWM_CDTYx). Writing in PWM_CDTYx is possible while the channel is disabled. After validation of the channel, the user must use PWM_CUPDx to update PWM_CDTYx as explained below.
- Configuration of the output waveform polarity for each channel (CPOL in PWM_CMRx).
- Enable Interrupts (set CHIDx in the PWM Interrupt Enable Register (PWM_IER)).
- Enable the PWM channel (set CHIDx in the PWM Enable Register (PWM_ENA)).

It is possible to synchronize different channels by enabling them at the same time by means of simultaneously setting several CHIDx bits in PWM_ENA.

- In such a situation, all channels may have the same clock selector configuration and the same period specified.

38.6.3.2 Source Clock Selection Criteria

The large number of source clocks can make selection difficult. The relationship between the value in the Channel Period Register (PWM_CPRDx) and the Channel Duty Cycle Register (PWM_CDTYx) can help the user in choosing. The event number written in PWM_CPRDx gives the PWM accuracy. The Duty Cycle quantum cannot be lower than *1/PWM_CPRDx* value. The higher the value of PWM_CPRDx, the greater the PWM accuracy.

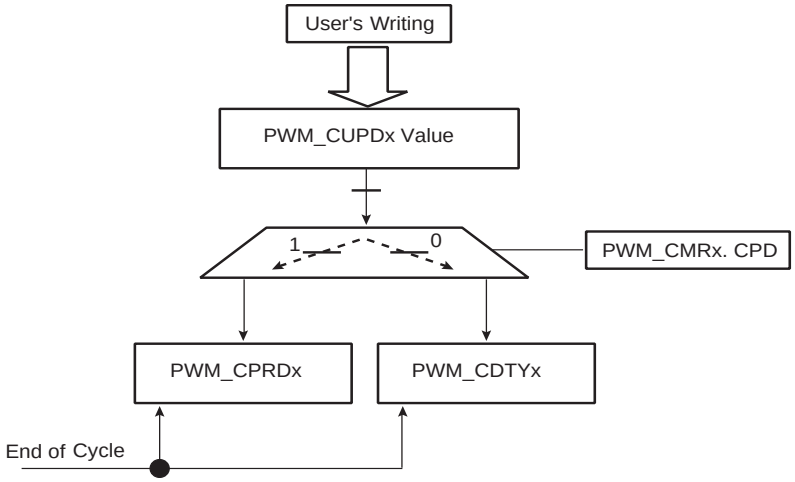
For example, if the user sets 15 (in decimal) in PWM_CPRDx, the user is able to set a value between 1 up to 14 in PWM_CDTYx. The resulting duty cycle quantum cannot be lower than 1/15 of the PWM period.

38.6.3.3 Changing the Duty Cycle or the Period

It is possible to modulate the output waveform duty cycle or period.

To prevent unexpected output waveform, the user must use PWM_CUPDx to change waveform parameters while the channel is still enabled. The user can write a new period value or duty cycle value in PWM_CUPDx. This register holds the new value until the end of the current cycle and updates the value for the next cycle. Depending on the CPD field in PWM_CMRx, PWM_CUPDx either updates PWM_CPRDx or PWM_CDTYx. Note that even if the update register is used, the period must not be smaller than the duty cycle.

Figure 38-6. Synchronized Period or Duty Cycle Update



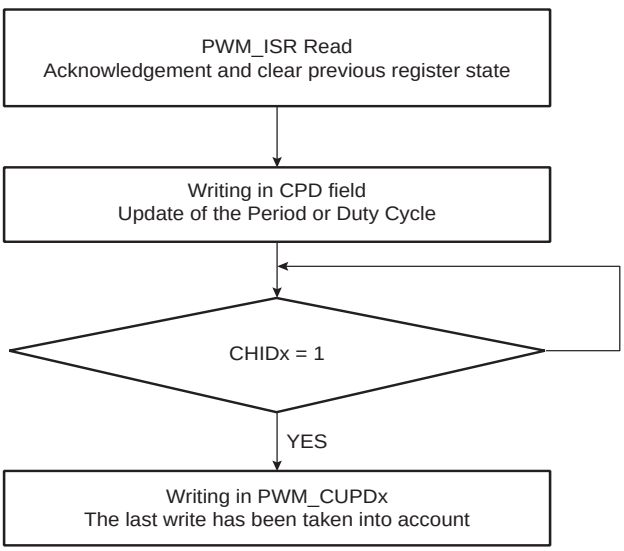
To prevent overwriting the PWM_CUPDx by software, the user can use status events in order to synchronize his software. Two methods are possible. In both, the user must enable the dedicated interrupt in PWM_IER at PWM Controller level.

The first method (polling method) consists of reading the relevant status bit in the PWM Interrupt Status Register (PWM_ISR) according to the enabled channel(s). See Figure 38-7.

The second method uses an Interrupt Service Routine associated with the PWM channel.

Note: Reading the PWM_ISR automatically clears CHIDx flags.

Figure 38-7. Polling Method



Note: Polarity and alignment can be modified only when the channel is disabled.

38.7 Pulse Width Modulation Controller (PWM) User Interface

Table 38-4. Register Mapping[1]

Offset	Register	Name	Access	Reset
0x00	PWM Mode Register	PWM_MR	Read/Write	0
0x04	PWM Enable Register	PWM_ENA	Write-only	-
0x08	PWM Disable Register	PWM_DIS	Write-only	-
0x0C	PWM Status Register	PWM_SR	Read-only	0
0x10	PWM Interrupt Enable Register	PWM_IER	Write-only	-
0x14	PWM Interrupt Disable Register	PWM_IDR	Write-only	-
0x18	PWM Interrupt Mask Register	PWM_IMR	Read-only	0
0x1C	PWM Interrupt Status Register	PWM_ISR	Read-only	0
0x20 - 0xFC	Reserved	–	–	–
0x100 - 0x1FC	Reserved	–	–	–
0x200 + ch_num * 0x20 + 0x00	PWM Channel Mode Register	PWM_CMR	Read/Write	0x0
0x200 + ch_num * 0x20 + 0x04	PWM Channel Duty Cycle Register	PWM_CDTY	Read/Write	0x0
0x200 + ch_num * 0x20 + 0x08	PWM Channel Period Register	PWM_CPRD	Read/Write	0x0
0x200 + ch_num * 0x20 + 0x0C	PWM Channel Counter Register	PWM_CCNT	Read-only	0x0
0x200 + ch_num * 0x20 + 0x10	PWM Channel Update Register	PWM_CUPD	Write-only	-

Notes: 1. Some registers are indexed with “ch_num” index ranging from 0 to 3.

38.7.1 PWM Mode Register

Name: PWM_MR

Address: 0x48008000

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							

- **DIVA: CLKA Divide Factor**

Value	Name	Description
0	CLK_OFF	CLKA clock is turned off
1	CLK_DIV1	CLKA clock is clock selected by PREA
2 - 255	-	CLKA clock is clock selected by PREA divided by DIVA factor

- **DIVB: CLKB Divide Factor**

Value	Name	Description
0	CLK_OFF	CLKB clock is turned off
1	CLK_DIV1	CLKB clock is clock selected by PREB
2 - 255	-	CLKB clock is clock selected by PREB divided by DIVB factor

- **PREA: CLKA Source Clock Selection**

Value	Name	Description
0000	MCK	Master Clock
0001	MCKDIV2	Master Clock divided by 2
0010	MCKDIV4	Master Clock divided by 4
0011	MCKDIV8	Master Clock divided by 8
0100	MCKDIV16	Master Clock divided by 16
0101	MCKDIV32	Master Clock divided by 32
0110	MCKDIV64	Master Clock divided by 64
0111	MCKDIV128	Master Clock divided by 128
1000	MCKDIV256	Master Clock divided by 256
1001	MCKDIV512	Master Clock divided by 512
1010	MCKDIV1024	Master Clock divided by 1024
Other	-	Reserved

- **PREB: CLKB Source Clock Selection**

Value	Name	Description
0000	MCK	Master Clock
0001	MCKDIV2	Master Clock divided by 2
0010	MCKDIV4	Master Clock divided by 4
0011	MCKDIV8	Master Clock divided by 8
0100	MCKDIV16	Master Clock divided by 16
0101	MCKDIV32	Master Clock divided by 32
0110	MCKDIV64	Master Clock divided by 64
0111	MCKDIV128	Master Clock divided by 128
1000	MCKDIV256	Master Clock divided by 256
1001	MCKDIV512	Master Clock divided by 512
1010	MCKDIV1024	Master Clock divided by 1024
Other	-	Reserved

38.7.2 PWM Enable Register

Name: PWM_ENA

Address: 0x48008004

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.

38.7.3 PWM Disable Register

Name: PWM_DIS

Address: 0x48008008

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 = Disable PWM output for channel x.

38.7.4 PWM Status Register

Name: PWM_SR

Address: 0x4800800C

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.

38.7.5 PWM Interrupt Enable Register

Name: PWM_IER

Address: 0x48008010

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 interrupt for PWM channel x.

38.7.6 PWM Interrupt Disable Register

Name: PWM_IDR

Address: 0x48008014

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 = Disable interrupt for PWM channel x.

38.7.7 PWM Interrupt Mask Register

Name: PWM_IMR

Address: 0x48008018

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 = Interrupt for PWM channel x is disabled.

1 = Interrupt for PWM channel x is enabled.

38.7.8 PWM Interrupt Status Register

Name: PWM_ISR

Address: 0x4800801C

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 = No new channel period has been achieved since the last read of the PWM_ISR.

1 = At least one new channel period has been achieved since the last read of the PWM_ISR.

Note: Reading PWM_ISR automatically clears CHIDx flags.

38.7.9 PWM Channel Mode Register

Name: PWM_CMR[0..3]

Address: 0x48008200 [0], 0x48008220 [1], 0x48008240 [2], 0x48008260 [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
–	–	–	–	–	CPD	CPOL	CALG
7	6	5	4	3	2	1	0
–	–	–	–	CPRE			

- **CPRE: Channel Prescaler**

Value	Name	Description
0000	MCK	Master Clock
0001	MCKDIV2	Master Clock divided by 2
0010	MCKDIV4	Master Clock divided by 4
0011	MCKDIV8	Master Clock divided by 8
0100	MCKDIV16	Master Clock divided by 16
0101	MCKDIV32	Master Clock divided by 32
0110	MCKDIV64	Master Clock divided by 64
0111	MCKDIV128	Master Clock divided by 128
1000	MCKDIV256	Master Clock divided by 256
1001	MCKDIV512	Master Clock divided by 512
1010	MCKDIV1024	Master Clock divided by 1024
1011	CLKA	Clock A
1100	CLKB	Clock B
Other	-	Reserved

- **CALG: Channel Alignment**

0 = The period is left aligned.

1 = The period is center aligned.

- **CPOL: Channel Polarity**

0 = The output waveform starts at a low level.

1 = The output waveform starts at a high level.

- **CPD: Channel Update Period**

0 = Writing to the PWM_CUPDx will modify the duty cycle at the next period start event.

1 = Writing to the PWM_CUPDx will modify the period at the next period start event.

38.7.10 PWM Channel Duty Cycle Register

Name: PWM_CDTY[0..3]

Address: 0x48008204 [0], 0x48008224 [1], 0x48008244 [2], 0x48008264 [3]

Access: Read/Write

31	30	29	28	27	26	25	24
CDTY							

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 (internal channel counter size) are significant.

- **CDTY: Channel Duty Cycle**

Defines the waveform duty cycle. This value must be defined between 0 and CPRD (PWM_CPRx).

38.7.11 PWM Channel Period Register

Name: PWM_CPRD[0..3]

Address: 0x48008208 [0], 0x48008228 [1], 0x48008248 [2], 0x48008268 [3]

Access: Read/Write

31	30	29	28	27	26	25	24
CPRD							
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							

Only the first 16 bits (internal channel counter size) are significant.

- **CPRD: Channel Period**

If the waveform is left-aligned, then the output waveform period depends on the counter source clock and can be calculated:

– By using the Master Clock (MCK) divided by an X given prescaler value (with X being 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024). The resulting period formula will be:

$$\frac{(X \times CPRD)}{MCK}$$

– By using a Master Clock divided by one of both DIVA or DIVB divider, the formula becomes, respectively:

$$\frac{(CPRD \times DIVA)}{MCK} \text{ or } \frac{(CPRD \times DIVB)}{MCK}$$

If the waveform is center-aligned, then the output waveform period depends on the counter source clock and can be calculated:

– By using the Master Clock (MCK) divided by an X given prescaler value (with X being 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024). The resulting period formula will be:

$$\frac{(2 \times X \times CPRD)}{MCK}$$

– By using a Master Clock divided by one of both DIVA or DIVB divider, the formula becomes, respectively:

$$\frac{(2 \times CPRD \times DIVA)}{MCK} \text{ or } \frac{(2 \times CPRD \times DIVB)}{MCK}$$

38.7.12 PWM Channel Counter Register

Name: PWM_CCNT[0..3]

Address: 0x4800820C [0], 0x4800822C [1], 0x4800824C [2], 0x4800826C [3]

Access: Read-only

31	30	29	28	27	26	25	24
CNT							

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							

- **CNT: Channel Counter Register**

Internal counter value. This register is reset when:

- The channel is enabled (PWM_ENA.CHIDx = 1).
- The counter reaches CPRD value defined in PWM_CPRDx if the waveform is left aligned.

38.7.13 PWM Channel Update Register

Name: PWM_CUPD[0..3]

Address: 0x48008210 [0], 0x48008230 [1], 0x48008250 [2], 0x48008270 [3]

Access: Write-only

31	30	29	28	27	26	25	24
CUPD							

23	22	21	20	19	18	17	16
CUPD							

15	14	13	12	11	10	9	8
CUPD							

7	6	5	4	3	2	1	0
CUPD							

- **CUPD: Channel Update Register**

This register acts as a double buffer for the period or the duty cycle. This prevents an unexpected waveform when modifying the waveform period or duty-cycle.

Only the first 16 bits (internal channel counter size) are significant.

When PWM_CMRx.CPD = 0, the duty-cycle (CDTY of PWM_CDTYx) is updated with the CUPD value at the beginning of the next period.

When PWM_CMRx.CPD = 1, the period (CPRD of PWM_CPRDx) is updated with the CUPD value at the beginning of the next period.

39. Segment Liquid Crystal Display Controller (SLCDC)

39.1 Description

The Segment Liquid Crystal Display Controller (SLCDC) drives a monochrome passive liquid crystal display (LCD) with up to 5 common terminals and up to 46 segment terminals.

An LCD consists of several segments (pixels or complete symbols) which can be visible or invisible. A segment has two electrodes with liquid crystal between them. When a voltage above a threshold voltage is applied across the liquid crystal, the segment becomes visible.

The voltage must alternate to avoid an electrophoresis effect in the liquid crystal, which degrades the display. Hence the waveform across a segment must not have a DC component.

The SLCDC is programmable to support many different requirements such as:

- Adjusting the driving time of the LCD pads in order to save power and increase the controllability of the DC offset.
- Driving smaller LCD (down to 1 common by 1 segment).
- Adjusting the SLCDC frequency in order to obtain the best compromise between frequency and consumption and adapt it to the LCD driver.
- Assigning the segments in a user defined pattern to simplify the use of the digital functions multiplexed on these pins.

Note: Please check SLCDC available signals in the main Block Diagram Figure 2-1.

Table 39-1. List of Terms

Term	Description
LCD	A passive display panel with terminals leading directly to a segment.
Segment	The least viewing element (pixel) which can be on or off.
Common(s)	Denotes how many segments are connected to a segment terminal.
Duty	1/(Number of common terminals on a current LCD display).
Bias	1/(Number of voltage levels used driving an LCD display -1).
Frame Rate	Number of times the LCD segments are energized per second.

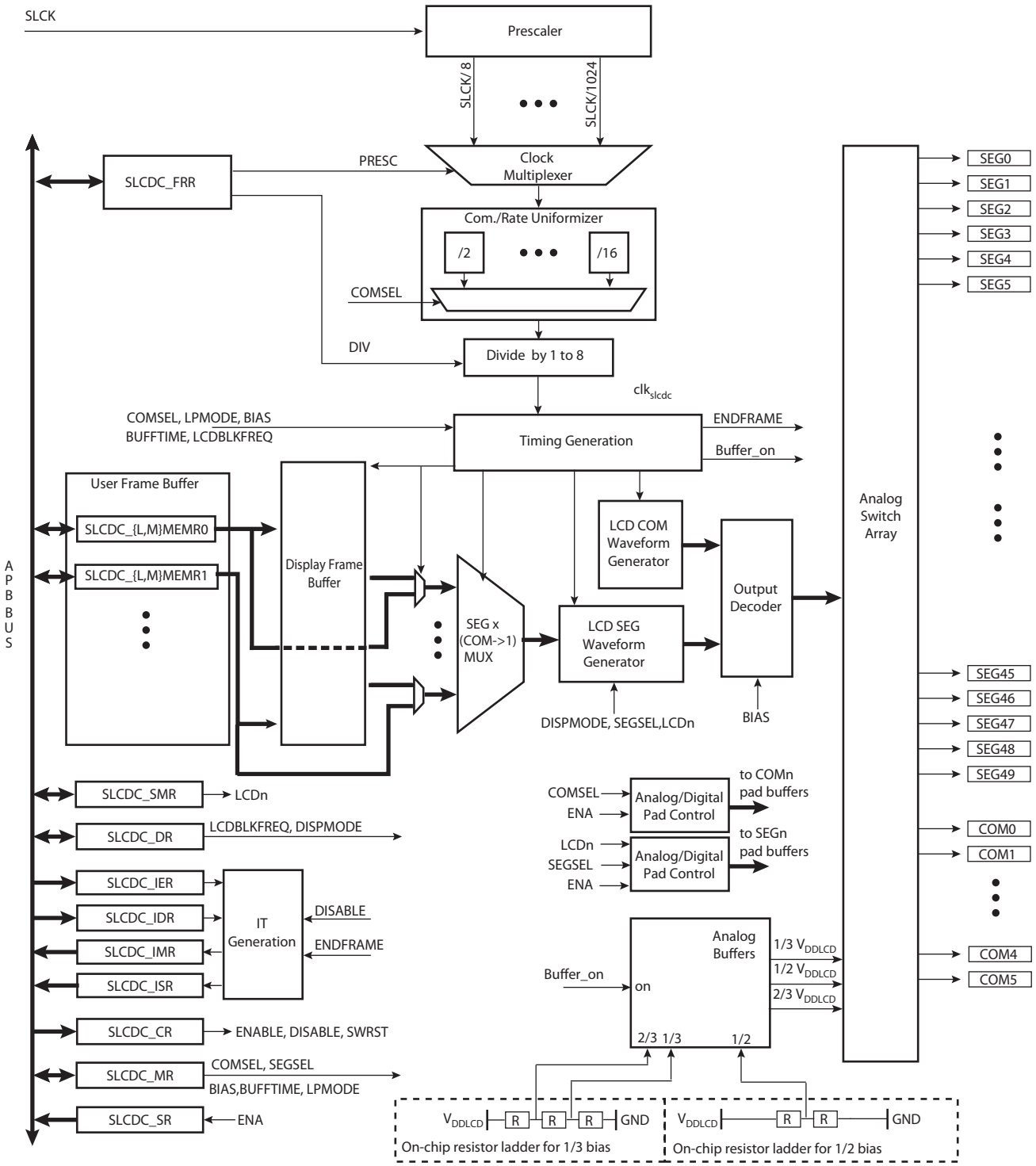
39.2 Embedded Characteristics

The SLCDC provides the following capabilities:

- Display Capacity: Up to 46 Segments and 5 Common Terminals
- Support from Static to 1/6 Duty
- Supports: Static and 1/2 and 1/3 Bias.
- Two LCD Supply Sources:
  - Internal (On-chip LCD Power Supply)
  - External
- LCD Output Voltage Software Selectable from 2.4V to VDDIN in 16 Steps (Control Embedded in the Supply Controller)
- Flexible Selection of Frame Frequency
- Two Interrupt Sources: End Of Frame and Disable
- Versatile Display Modes
- Equal Source and Sink Capability to Maximize LCD Lifetime
- Segment and Common Pins Not Needed to Drive the Display Can be Used as Ordinary I/O Pins
- Segments Layout Can be Fully Defined by User to Optimize Usage of Multiplexed Digital Functions
- Latching of Display Data Gives Full Freedom in Register Updates
- Power-Saving Modes for Extremely Low Power Consumption

39.3 Block Diagram

Figure 39-1. SLCDC Block Diagram



39.4 I/O Lines Description

Table 39-2. I/O Lines Description

Name	Description	Type
SEG [3:47], SEG49	Segments control signals	Output
COM [0:4]	Commons control signals	Output

39.5 Product Dependencies

39.5.1 I/O Lines

The pins used for interfacing the SLCD Controller may be multiplexed with PIO lines. Refer to the product block diagram.

If I/O lines of the SLCD Controller are not used by the application, they can be used for other purposes by the PIO Controller.

By default (SLCDC_SMR0/1 registers cleared), the assignment of the segment controls and commons are automatically done depending on COMSEL and SEGSEL in SLCDC_MR. For example, if 10 segments are programmed in the SEGSEL field, they are automatically assigned to SEG[9:0] whereas the remaining SEG pins are automatically selected to be driven by the multiplexed digital functions.

In any case, the user can define a new layout pattern for the segment assignment by programming the SLCDC_SMR0/1 registers in order to optimize the usage of multiplexed digital function. If at least 1 bit is set in SLCDC_SMR0/1 registers, the corresponding I/O line is driven by an LCD segment, whereas any cleared bit of this register selects the corresponding multiplexed digital function.

Table 39-3. I/O Lines

Instance	Signal	I/O Line	Peripheral
SLCDC	COM0	PA0	X1
SLCDC	COM1	PA1	X1
SLCDC	COM2	PA2	X1
SLCDC	COM3	PA3	X1
SLCDC	COM4/AD1	PA4	X1
SLCDC	SEG3	PA9	X1
SLCDC	SEG4	PA10	X1
SLCDC	SEG5	PA11	X1
SLCDC	SEG6/AD0	PA12	X1
SLCDC	SEG7	PA13	X1
SLCDC	SEG8	PA14	X1
SLCDC	SEG9	PA15	X1
SLCDC	SEG10	PA16	X1
SLCDC	SEG11	PA17	X1
SLCDC	SEG12	PA18	X1
SLCDC	SEG13	PA19	X1
SLCDC	SEG14	PA20	X1
SLCDC	SEG15	PA21	X1

Table 39-3. I/O Lines (Continued)

SLCDC	SEG16	PA22	X1
SLCDC	SEG17	PA23	X1
SLCDC	SEG18	PA24	X1
SLCDC	SEG19	PA25	X1
SLCDC	SEG20	PA26	X1
SLCDC	SEG21	PA27	X1
SLCDC	SEG22	PA28	X1
SLCDC	SEG23	PA29	X1
SLCDC	SEG24	PB6	X1
SLCDC	SEG25	PB7	X1
SLCDC	SEG26	PB8	X1
SLCDC	SEG27	PB9	X1
SLCDC	SEG28	PB10	X1
SLCDC	SEG29	PB11	X1
SLCDC	SEG30	PB12	X1
SLCDC	SEG31/AD3	PB13	X1
SLCDC	SEG32	PB14	X1
SLCDC	SEG33	PB15	X1
SLCDC	SEG34	PB16	X1
SLCDC	SEG35	PB17	X1
SLCDC	SEG36	PB18	X1
SLCDC	SEG37	PB19	X1
SLCDC	SEG38	PB20	X1
SLCDC	SEG39	PB21	X1
SLCDC	SEG40	PB22	X1
SLCDC	SEG41/AD4	PB23	X1
SLCDC	SEG42	PB24	X1
SLCDC	SEG43	PB25	X1
SLCDC	SEG44	PB26	X1
SLCDC	SEG45	PB27	X1
SLCDC	SEG46	PB28	X1
SLCDC	SEG47	PB29	X1
SLCDC	SEG49/AD5	PB31	X1

39.5.2 Power Management

The SLCD Controller is clocked by the slow clock (SLCK). All the timings are based upon a typical value of 32 kHz for SLCK.

The LCD segment/common pad buffers are supplied by the VDDLCD domain.

39.5.3 Interrupt Sources

The SLCD Controller interrupt line is connected to one of the internal sources of the Interrupt Controller. Using the SLCD Controller interrupt requires prior programming of the Interrupt Controller.

Table 39-4. Peripheral IDs

Instance	ID
SLCDC	32

39.5.4 Number of Segments and Commons

The product, embeds 46 segments and 5 Commons.

39.6 Functional Description

The use of the SLCDC comprises three phases of functionality: initialization sequence, display phase and disable sequence.

- Initialization Sequence:

1. Select the LCD supply source in the Shutdown Controller.
   - Internal: the On-chip LCD Power Supply is selected.
   - External: the external supply source must be between 2.5 to 3.6V.
2. Select the clock division (SLCDC_FRR) to use a proper frame rate.
3. Enter the number of common and segments terminals (SLCDC_MR).
4. Select the bias in compliance with the LCD manufacturer datasheet (SLCDC_MR).
5. Enter buffer driving time (SLCDC_MR).
6. Define the segments remapping pattern if required (SLCDC_SMR0/1).

- During the Display Phase:

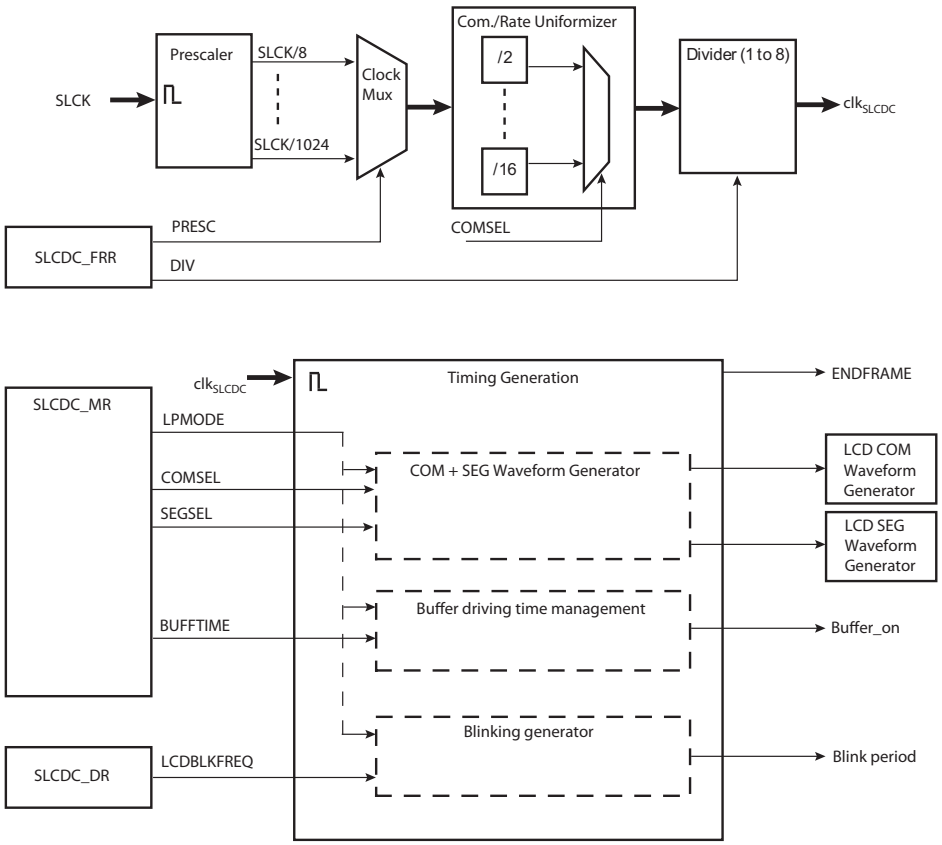
1. Data may be written at any time in the SLCDC memory. The data is automatically latched and displayed at the next LCD frame.
2. It is possible to:
   - Adjust contrast.
   - Adjust the frame frequency.
   - Adjust buffer driving time.
   - Reduce the SLCDC consumption by entering in low-power waveform at any time.
   - Use the large set of display features such as blinking, inverted blink, etc.

- Disable Sequence: See Section 39.6.7 ”Disabling the SLCDC”.

39.6.1 Clock Generation

39.6.1.1 Block Diagram

Figure 39-2. Clock Generation Block Diagram

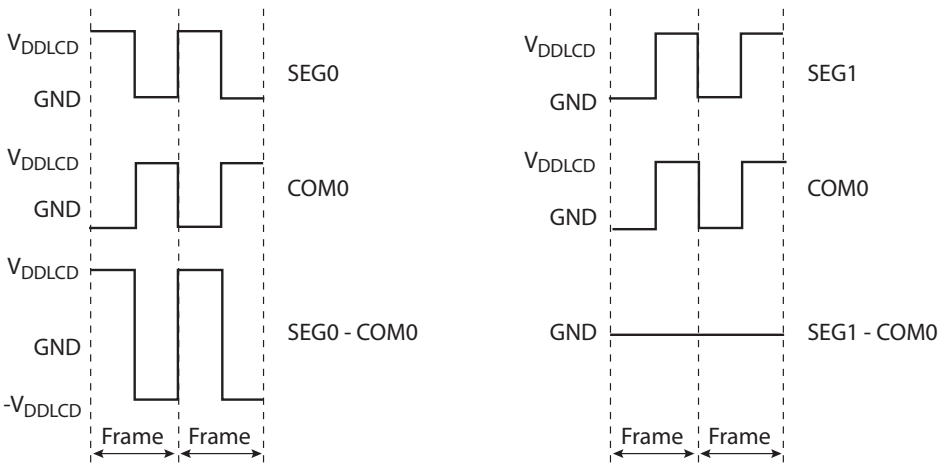


39.6.2 Waveform Generation

39.6.2.1 Static Duty and Bias

This kind of display is driven with the waveform shown in Figure 39-3. SEG0 - COM0 is the voltage across a segment that is on, and SEG1 - COM0 is the voltage across a segment that is off.

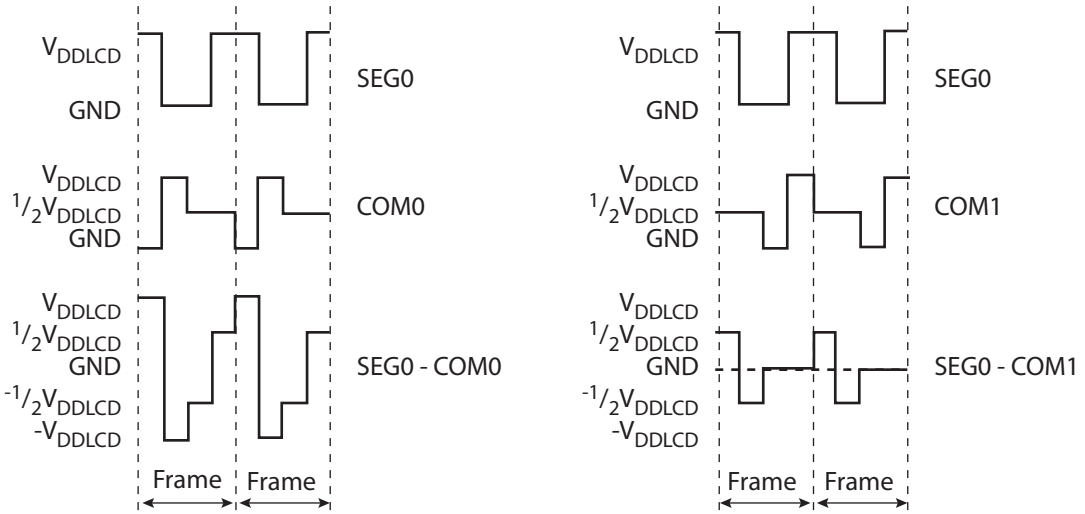
Figure 39-3. Driving an LCD with One Common Terminal



39.6.2.2 1/2 Duty and 1/2 Bias

For an LCD with two common terminals (1/2 duty), a more complex waveform must be used to control segments individually. Although 1/3 bias can be selected, 1/2 bias is most common for these displays. In the waveform shown in Figure 39-4, SEG0 - COM0 is the voltage across a segment that is on, and SEG0 - COM1 is the voltage across a segment that is off.

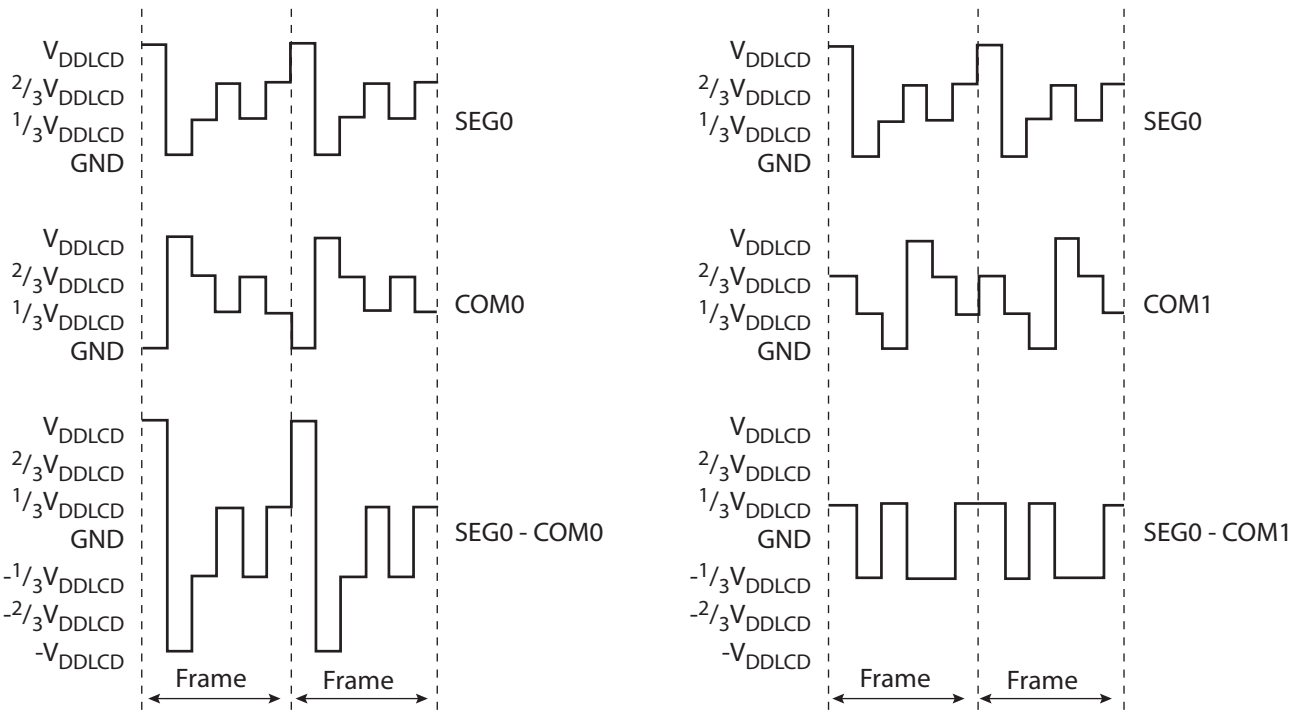
Figure 39-4. Driving an LCD with Two Common Terminals



39.6.2.3 1/3 Duty and 1/3 Bias

1/3 bias is recommended for an LCD with three common terminals (1/3 duty). In the waveform shown in Figure 39-5, SEG0 - COM0 is the voltage across a segment that is on and SEG0 - COM1 is the voltage across a segment that is off.

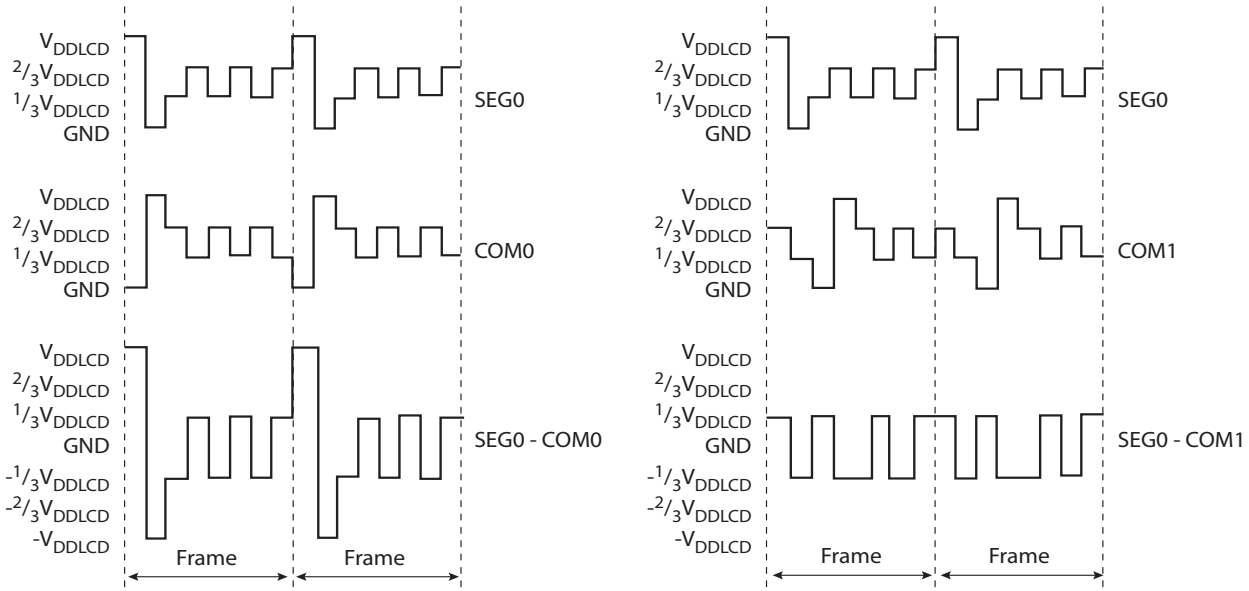
Figure 39-5. Driving an LCD with Three Common Terminals



39.6.2.4 1/4 Duty and 1/3 Bias

1/3 bias is optimal for LCD displays with four common terminals (1/4 duty). In the waveform shown in Figure 39-6, SEG0 - COM0 is the voltage across a segment that is on and SEG0 - COM1 is the voltage across a segment that is off.

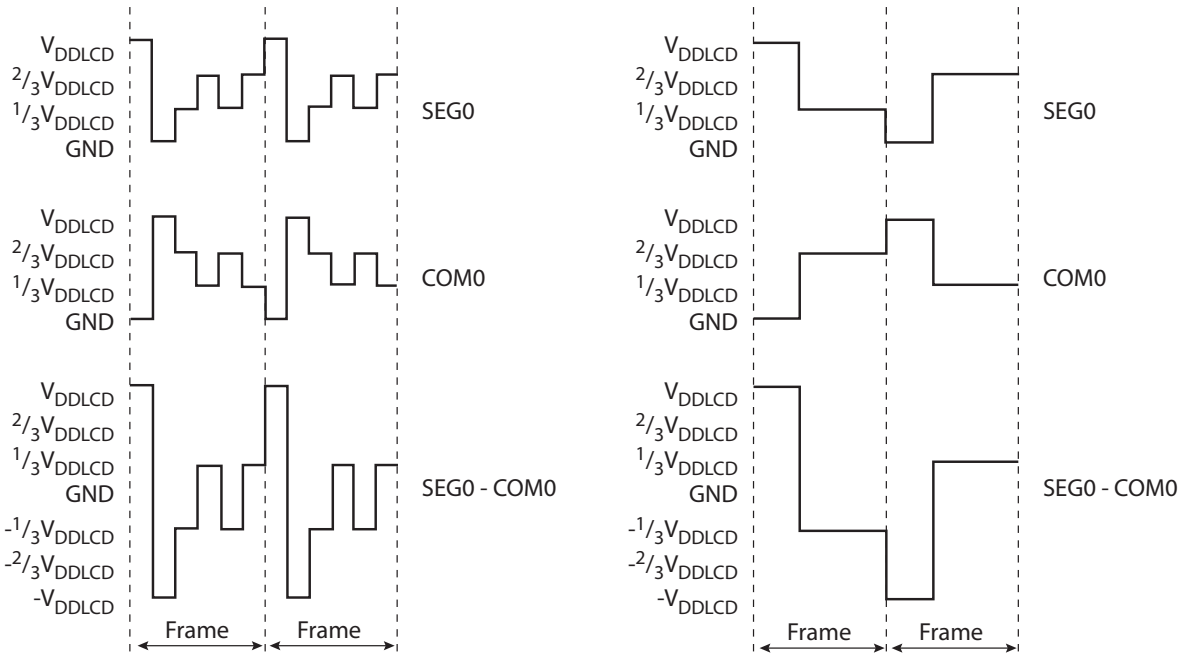
Figure 39-6. Driving an LCD with Four Common Terminals



39.6.2.5 Low-Power Waveform

To reduce toggle activity and hence power consumption, a low-power waveform can be selected by writing LPMODE to '1'. The default and low-power waveform is shown in Figure 39-7 for 1/3 duty and 1/3 bias. For other selections of duty and bias, the effect is similar.

Figure 39-7. Default and Low-Power Waveform



Note: Refer to the LCD specification to verify that low-power waveforms are supported.

39.6.2.6 Frame Rate

The Frame Rate register (SLCDC_FRR) enables the generation of the frequency used by the SLCDC. It is done by a prescaler (division by 8, 16, 32, 64, 128, 256, 512 and 1024) followed by a finer divider (division by 1, 2, 3, 4, 5, 6, 7 or 8).

To calculate the needed frame frequency, the equation below must be used:

$$f_{frame} = \frac{f_{SLCK}}{(PRESC \cdot DIV \cdot NCOM)}$$

where:

f_{SLCK} = slow clock frequency

f_{frame} = frame frequency

PRESC = prescaler value (8, 16, 32, 64, 128, 256, 512 or 1024)

DIV = divider value (1, 2, 3, 4, 5, 6, 7, or 8)

NCOM = depends of number of commons and is defined in Table 39-5

NCOM is automatically provided by the SLCDC

For example, if SLCDC_MR.COMSEL is written to ‘0’ (1 common terminal on display device), the SLCDC introduces a divider by 16 so that NCOM = 16. If COMSEL is written to ‘3’ (3 common terminals on display device), the SLCDC introduces a divider by 5 so that the NCOM remains close to 16 (the frame rate is standardized regardless of the number of driven commons).

Table 39-5. NCOM

Number of Commons	NCOM	Uniformizer Divider
1	16	16
2	16	8
3	15	5
4	16	4
5	15	3
6	18	3

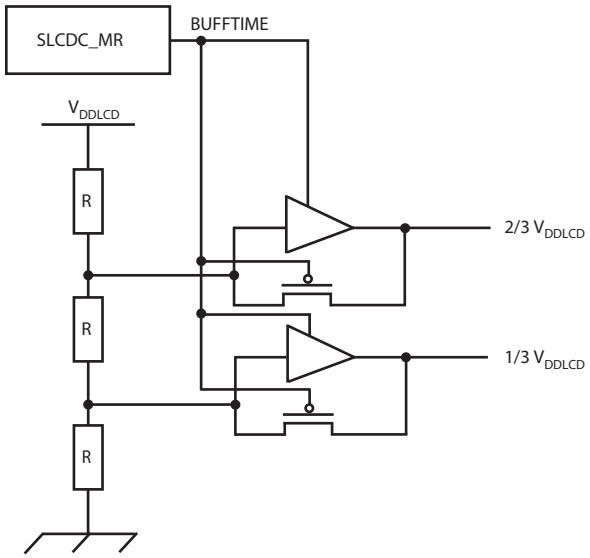
39.6.2.7 Buffer Driving Time

Intermediate voltage levels are generated from buffer drivers. The buffers are active for the amount of time specified by SLCDC_MR.BUFTIME, then they are bypassed.

Shortening the drive time reduces power consumption, but displays with high internal resistance or capacitance may need a longer drive time to achieve sufficient contrast.

Example for bias = 1/3.

Figure 39-8. Buffer Driving

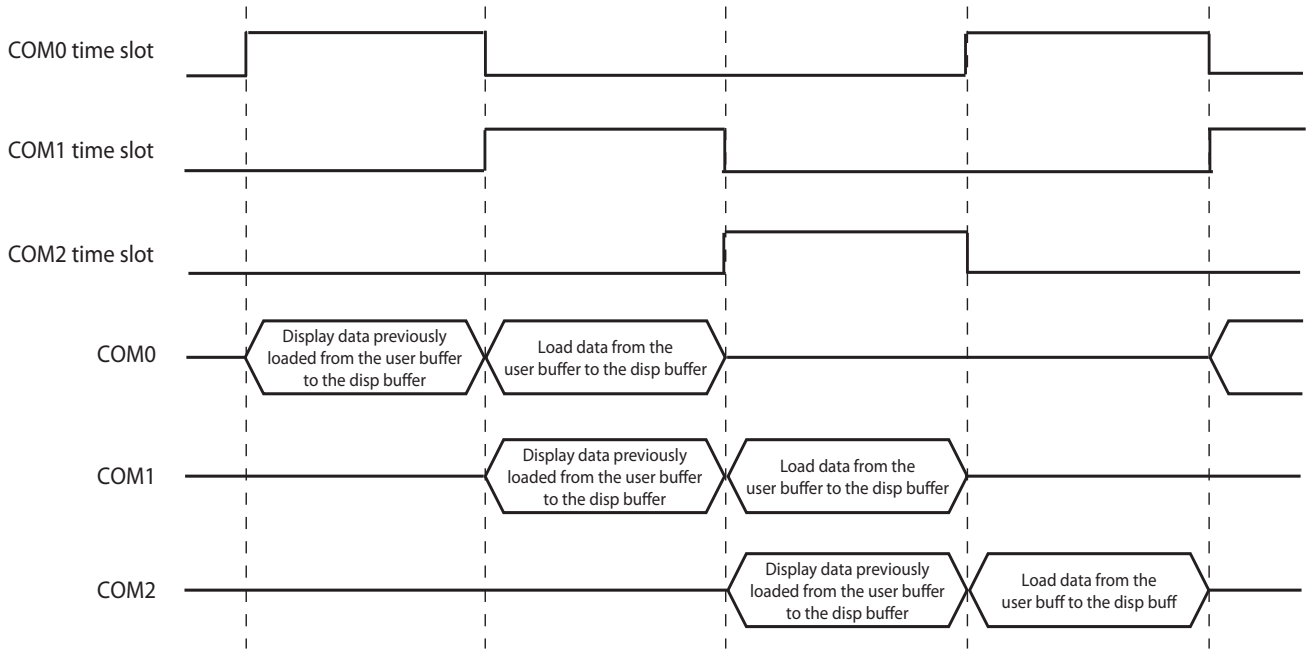


39.6.3 Number of Commons, Segments and Bias

It is important to note that the selection of the number of commons, segments and the bias can be programmed when the SLCDC is disabled.

39.6.4 SLCDC Memory

Figure 39-9. Memory Management



When a bit in the display memory (SLCDC_LMEMRx and SLCDC_MMEMRx registers) is written to ‘1’, the corresponding segment is energized (on), and non-energized when a bit in the display memory is written to ‘0’.

At the beginning of each common, the display buffer is updated. The value of the previous common is latched in the display memory (its value is transferred from the user buffer to the frame buffer).

The advantages of this solution are:

- Ability to access the user buffer at any time in the frame, in any display mode and even in low-power waveform.
- Ability to change only one pixel without reloading the picture.

39.6.5 Display Features

In order to improve the flexibility of SLCDC the following set of display modes are embedded:

1. Force mode Off: All pixels are turned off and the memory content is kept.
2. Force mode On: All pixels are turned on and the memory content is kept.
3. Inverted mode: All pixels are set in the inverted state as defined in SLCDC memory and the memory content is kept.
4. Two Blinking Modes:
   - Standard Blinking mode: All pixels are alternately turned off to the predefined state in SLCDC memory at LCDBLKFREQ frequency.
   - Inverted Blinking mode: All pixels are alternately turned off to the predefined opposite state in SLCDC memory at LCDBLKFREQ frequency.
5. Buffer Swap Mode: All pixels are alternatively assigned to the state defined in the user buffer then to the state defined in the display buffer.

39.6.6 Buffer Swap Mode

This mode is used to assign all pixels to two states alternatively without reloading the user buffer at each change.

The means to alternatively display two states is as follows:

1. Initially, the SLCDC must be in normal mode or in standard blinking mode.
2. Data corresponding to the first pixel state is written in the user buffer (through the SLCDC_MEM registers).
3. Wait two ENDFRAME events (to be sure that the user buffer is entirely transferred in the display buffer).
4. SLCDC_DR must be programmed with DISPMODE = 6 (User Buffer Only Load Mode). This mode blocks the automatic transfer from the user buffer to the display buffer.
5. Wait ENDFRAME event. (The display mode is internally updated at the beginning of each frame).
6. Data corresponding to the second pixel state is written in the user buffer (through the SLCDC_MEM registers). So, now the first pixel state is in the display buffer and the second pixel state is in the user buffer.
7. SLCDC_DR must be programmed with DISPMODE = 7 (buffer swap mode) and LCDBLKFREQ must be programmed with the required blinking frequency (if not previously done).

Now, each state is alternatively displayed at LCDBLKFREQ frequency.

Except for the phase dealing with the storage of the two display states, the management of the Buffer Swap Mode is the same as the standard blinking mode.

39.6.7 Disabling the SLCDC

There are two ways to disable the SLCDC:

1. By using SLCDC_CR.LCDDIS (recommended method). In this case, SLCDC configuration and memory content are maintained.
2. By using SLCDC_CR.SWRST that acts like a hardware reset for SLCDC only.

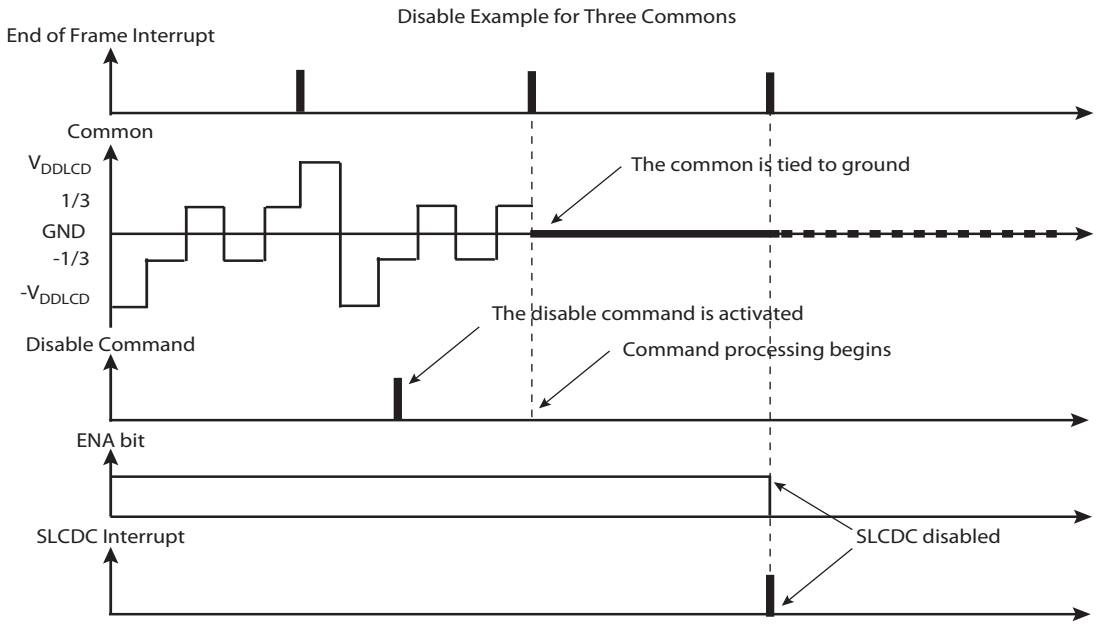
Both methods are described in the following sections.

39.6.7.1 Disable Bit

SLCDC_CR.LCDDIS can be set at any time. When the LCD Disable Command is activated during a frame, the SLCDC is not immediately stopped (see Figure 39-10).

The next frame is generated in “All Ground” mode (whereby all commons and segments are tied to ground). At the end of this “All Ground” frame, the disable interrupt is asserted if SLCDC_IMR.DIS is set. The SLCDC is then disabled.

Figure 39-10. Disabling Sequence

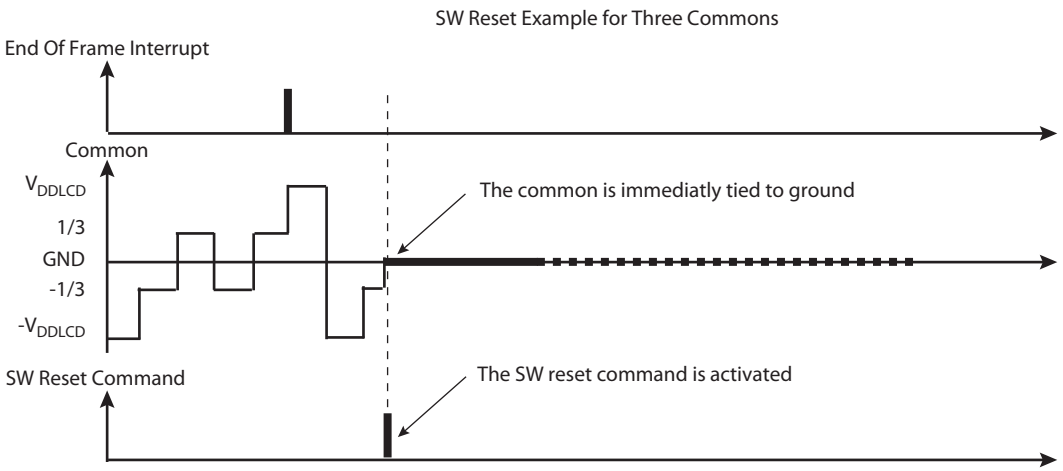


39.6.7.2 Software Reset

When the SLCDC software reset command is activated during a frame, it is immediately processed and all commons and segments are tied to ground.

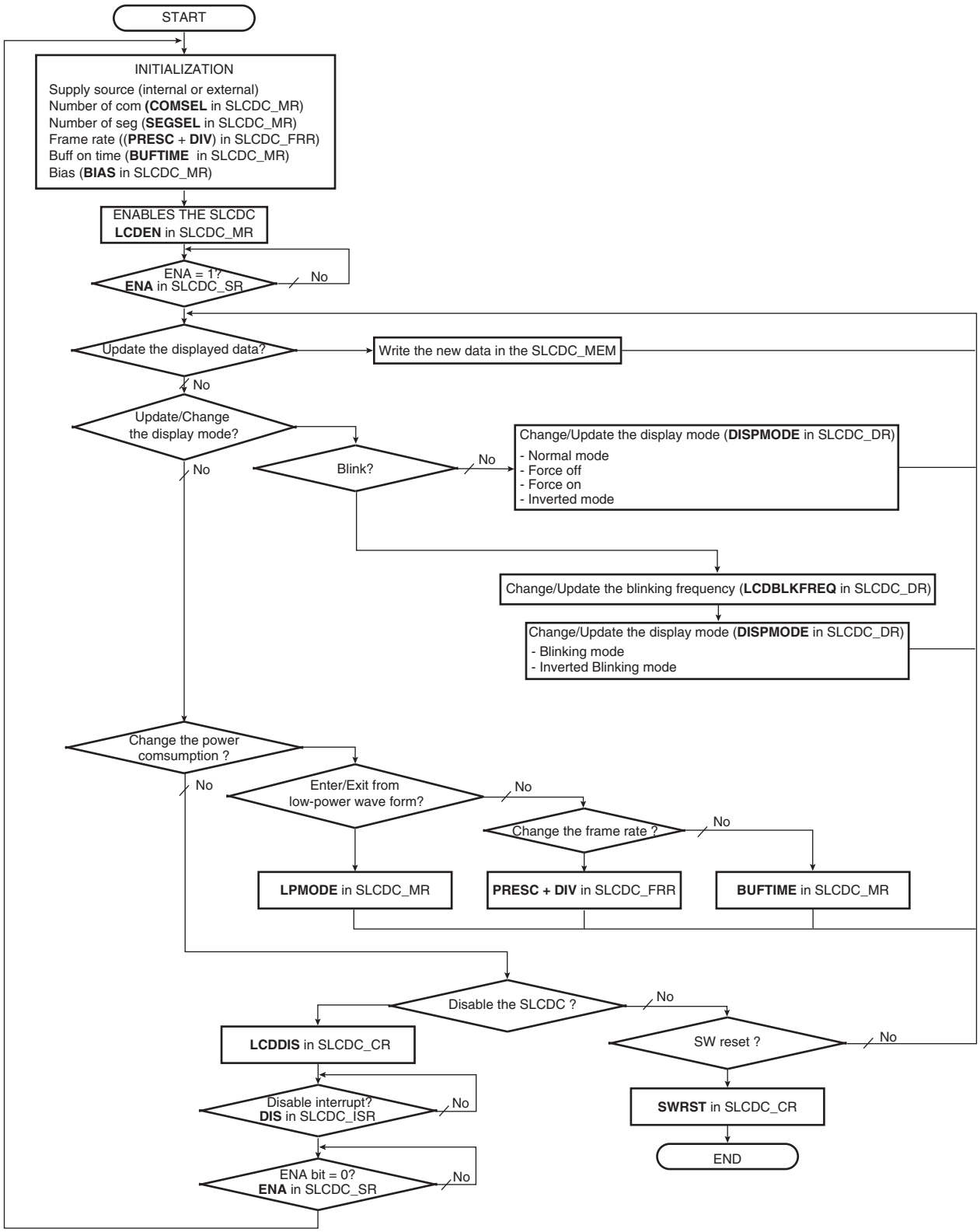
Note that in the case of a software reset, the disable interrupt is not asserted.

Figure 39-11. Software Reset



39.6.8 Flowchart

Figure 39-12. SLCDC Flow Chart



39.6.9 User Buffer Organization

The pixels to be displayed are written into SLCDC_LMEMRx and SLCDC_MMEMRx. There are up to two 32-bit registers for each common terminal. Table 39-6 provides the address mapping of all commons/segments to be displayed.

If the segment map registers (SLCDC_SMR0/1) are cleared and the number of segments to handle (SLCDC_MR.SEGSEL) is lower than or equal to 32, the registers SLCDC_MMEMRx are not required to be programmed and can be left cleared (default value).

In case segments are remapped, the SLCDC_MMEMRx registers are not required to be programmed if SLCDC_SMR1 register is cleared (i.e., no segment remapped on SEG32 to SEG49 I/O pins). In this case SLCDC_MMEMRx registers must be cleared.

In the same way if all segments are remapped on the upper part of the SEG terminals (SEG32 to SEG49) there is no need to program SLCDC_LMEMRx registers (they must be cleared).

When segment remap is used (SLCDC_SMR0/1 registers differ from 0), the unmapped segments must be kept cleared to limit internal signal switching.

Table 39-6. Commons/segments Address Mapping

Register	Common Terminal	SEG0	--	SEG31	SEG32	--	SEG49	Memory address
SLCDC_MMEMR5	COM5				X	--	X	0x22C
SLCDC_LMEMR5	COM5	X	--	X				0x228
SLCDC_MMEMR4	COM4				X	--	X	0x224
SLCDC_LMEMR4	COM4	X	--	X				0x220
SLCDC_MMEMR3	COM3				X	--	X	0x21C
SLCDC_LMEMR3	COM3	X	--	X				0x218
SLCDC_MMEMR2	COM2				X	--	X	0x214
SLCDC_LMEMR2	COM2	X	--	X				0x210
SLCDC_MMEMR1	COM1				X	--	X	0x20C
SLCDC_LMEMR1	COM1	X	--	X				0x208
SLCDC_MMEMR0	COM0				X	--	X	0x204
SLCDC_LMEMR0	COM0	X	--	X				0x200

39.6.10 Segments Mapping Function

By default the segments pins (SEG0:49) are automatically assigned according to the SEGSEL configuration in the SLCDC_MR. The unused SEG I/O pins are forced to be driven by a digital peripheral or can be used as I/O through the PIO controller.

The automatic assignment is performed if the segment mapping function is not used (SLCDC_SMR0/1 registers are cleared). The following table provides such assignments.

Table 39-7. Segment Pin Assignments

SEGSEL	I/O Port in Use as Segment Driver	I/O Port Pin if SLCDC_SMR0/1 = 0
0	SEG0	SEG1:49
1	SEG0:1	SEG2:49
...	...	...
48	SEG0:48	SEG49
49	SEG0:49	None

Programming is straightforward in this mode but it prevents flexibility of use of the digital peripheral multiplexed on SEG0:49 especially when the number of segments to drive is close to the maximum (50).

For example, if the SEGSEL is set to 48, only the digital peripheral associated to SEG49 can be used and none of the other digital peripherals multiplexed on SEG0:48 I/O can be used.

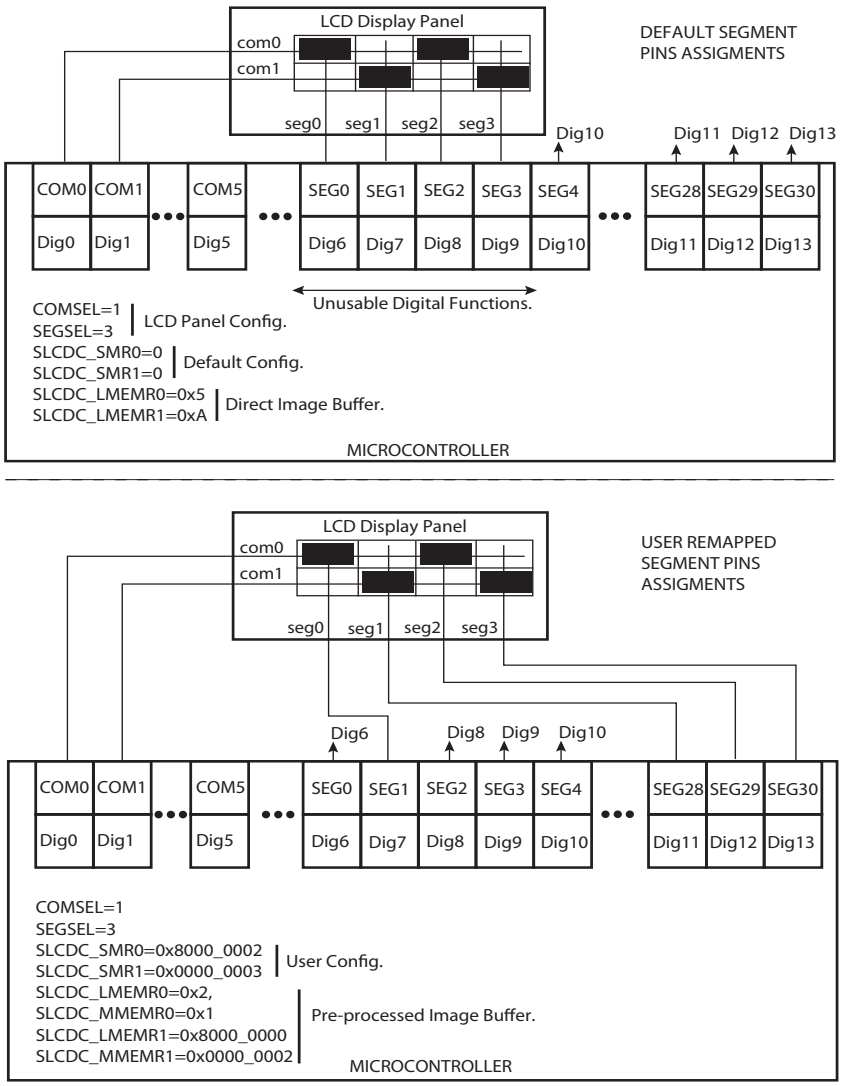
To offer a flexible selection of digital peripherals multiplexed on SEG0:49 the user can manually configure the SEG I/O pins to be driven by the SLCDC.

This is done by programming the SLCDC_SMR0/1 registers. As soon as their values differ from 0 the segment remapping mode is used.

When configuring a logic 1 at index n (n = 0..49) in SLCDC_SMR0 or SLCDC_SMR1, the SLCDC forces the SEGn I/O pin to be driven by a segment waveform. In this mode, the SEGSEL field configuration value in SLCDC_MR is ignored.

In remapping mode, the software dispatches the pixels into SLCDC_LMEMRx or SLCDC_MMEMRx according to what is programmed in SLCDC_SMR0 or SLCDC_SMR1.

Figure 39-13. Segments Remapping Example



39.7 Waveform Specifications

39.7.1 DC Characteristics

Refer to the section “DC Characteristics”.

39.7.2 LCD Contrast

The peak value (V_{DDLCD}) on the output waveform determines the LCD Contrast. V_{DDLCD} is controlled by software in 16 steps from 2.4V to VDDIN.

This is a function of the Supply Controller.

39.8 Segment LCD Controller (SLCDC) User Interface

Table 39-8. Register Mapping

Offset	Register	Name	Access	Reset
0x00	SLCDC Control Register	SLCDC_CR	Write-only	-
0x04	SLCDC Mode Register	SLCDC_MR	Read/Write	0x0
0x08	SLCDC Frame Rate Register	SLCDC_FRR	Read/Write	0x0
0x0C	SLCDC Display Register	SLCDC_DR	Read/Write	0x0
0x10	SLCDC Status Register	SLCDC_SR	Read-only	0x0
0x20	SLCDC Interrupt Enable Register	SLCDC_IER	Write-only	-
0x24	SLCDC Interrupt Disable Register	SLCDC_IDR	Write-only	-
0x28	SLCDC Interrupt Mask Register	SLCDC_IMR	Read-only	-
0x2C	SLCDC Interrupt Status Register	SLCDC_ISR	Read-only	0x0
0x30	SLCDC Segment Map Register 0	SLCDC_SMR0	Read/Write	0x0
0x34	SLCDC Segment Map Register 1	SLCDC_SMR1	Read/Write	0x0
0x38 - 0xE0	Reserved	-	-	-
0xE4 - 0xE8	Reserved	-	-	-
0xEC - 0xF8	Reserved	-	-	-
0xFC	Reserved	-	-	-
0x200 + com*0x8 + 0x0	SLCDC LSB Memory Register	SLCDC_LMEMR	Read/Write	0x0
0x200 + com*0x8 + 0x4	SLCDC MSB Memory Register	SLCDC_MMEMR	Read/Write	0x0

39.8.1 SLCDC Control Register

Name: SLCDC_CR

Address: 0x4003C000

Access: Write-only

Reset: 0x00000000

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

- **LCDEN: Enable the LCDC**

0 = No effect.

1 = The SLCDC is enabled.

- **LCDDIS: Disable LCDC**

0 = No effect.

1 = The SLCDC is disabled.

Note: LCDDIS is processed at the beginning of the next frame.

- **SWRST: Software Reset**

0 = No effect.

1 = Equivalent to a power-up reset. When this command is performed, the SLCDC immediately ties all segments end commons lines to values corresponding to a “ground voltage”.

39.8.2 SLCDC Mode Register

Name: SLCDC_MR

Address: 0x4003C004

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	LPMODE

23	22	21	20	19	18	17	16
–	–	BIAS		BUFTIME			

15	14	13	12	11	10	9	8
–	–	SEGSEL					

7	6	5	4	3	2	1	0
–	–	–	–	–	COMSEL		

- **COMSEL: Selection of the Number of Commons**

(For safety reasons, can be configured when SLCDC is disabled).

Value	Name	Description
0	COM_0	COM0 is driven by SLCDC, COM1:4 are driven by digital function
1	COM_0TO1	COM0:1 are driven by SLCDC, COM2:4 are driven by digital function
2	COM_0TO2	COM0:2 are driven by SLCDC, COM3:4 are driven by digital function
3	COM_0TO3	COM0:3 are driven by SLCDC, COM4 is driven by digital function
4	COM_0TO4	COM0:4 are driven by SLCDC, No COM pin driven by digital function

- **SEGSEL: Selection of the Number of Segments**

(For safety reasons, can be configured when SLCDC is disabled).

SEGSEL must be programmed with the number of segments of the display panel minus 1.

If segment remapping function is not used (i.e., SLCDC_SMRx equal 0) the SEGn [n = 0..49] I/O pins where n is greater than SEGSEL are forced to be driven by digital function. When segments remapping function is used, SEGn pins are driven by SLCDC only if corresponding PIXELn configuration bit is set in SLCDC_SMR1/0 registers.

- **BIAS: LCD Display Configuration**

(For safety reasons, can be configured when SLCDC is disabled).

Value	Name	Description
0	STATIC	Static
1	BIAS_1_2	Bias 1/2
2	BIAS_1_3	Bias 1/3

- **LPMODE: Low-Power Mode**

(Processed at beginning of next frame).

0 = Normal Mode.

1 = Low-Power Waveform is enabled.

- **BUFTIME: Buffer On-Time**

(Processed at beginning of next frame).

Value	Name	Description
0	OFF	Nominal drive time is 0% of SLCK period
1	X2_SLCK_PERIOD	Nominal drive time is 2 periods of SLCK clock
2	X4_SLCK_PERIOD	Nominal drive time is 4 periods of SLCK clock
3	X8_SLCK_PERIOD	Nominal drive time is 8 periods of SLCK clock
4	X16_SLCK_PERIOD	Nominal drive time is 16 periods of SLCK clock
5	X32_SLCK_PERIOD	Nominal drive time is 32 periods of SLCK clock
6	X64_SLCK_PERIOD	Nominal drive time is 64 periods of SLCK clock
7	X128_SLCK_PERIOD	Nominal drive time is 128 periods of SLCK clock
8	PERCENT_50	Nominal drive time is 50% of SLCK period
9	PERCENT_100	Nominal drive time is 100% of SLCK period

39.8.3 SLCDC Frame Rate Register

Name: SLCDC_FRR

Address: 0x4003C008

Access: Read/Write

Reset: 0x00000000

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

- **PRESC: Clock Prescaler**

(Processed at beginning of next frame).

Value	Name	Description
0	SLCK_DIV8	Slow clock is divided by 8
1	SLCK_DIV16	Slow clock is divided by 16
2	SLCK_DIV32	Slow clock is divided by 32
3	SLCK_DIV64	Slow clock is divided by 64
4	SLCK_DIV128	Slow clock is divided by 128
5	SLCK_DIV256	Slow clock is divided by 256
6	SLCK_DIV512	Slow clock is divided by 512
7	SLCK_DIV1024	Slow clock is divided by 1024

- **DIV: Clock Division**

(Processed at beginning of next frame).

Value	Name	Description
0	PRESC_CLK_DIV1	Clock output from prescaler is divided by 1
1	PRESC_CLK_DIV2	Clock output from prescaler is divided by 2
2	PRESC_CLK_DIV3	Clock output from prescaler is divided by 3
3	PRESC_CLK_DIV4	Clock output from prescaler is divided by 4
4	PRESC_CLK_DIV5	Clock output from prescaler is divided by 5
5	PRESC_CLK_DIV6	Clock output from prescaler is divided by 6
6	PRESC_CLK_DIV7	Clock output from prescaler is divided by 7
7	PRESC_CLK_DIV8	Clock output from prescaler is divided by 8

39.8.4 SLCDC Display Register

Name: SLCDC_DR

Address: 0x4003C00C

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
LCDBLKFREQ							

7	6	5	4	3	2	1	0
–	–	–	–	–	DISPMODE		

- **DISPMODE: Display Mode Register**

(Processed at beginning of next frame).

Value	Name	Description
0	NORMAL	Normal Mode: Latched data are displayed.
1	FORCE_OFF	Force Off Mode: All pixels are invisible. (The SLCDC memory is unchanged).
2	FORCE_ON	Force On Mode: All pixels are visible. (The SLCDC memory is unchanged).
3	BLINKING	Blinking Mode: All pixels are alternately turned off to the predefined state in SLCDC memory at LCDBLKFREQ frequency. (The SLCDC memory is unchanged).
4	INVERTED	Inverted Mode: All pixels are set in the inverted state as defined in SLCDC memory. (The SLCDC memory is unchanged).
5	INVERTED_BLINK	Inverted Blinking Mode: All pixels are alternately turned off to the predefined opposite state in SLCDC memory at LCDBLKFREQ frequency. (The SLCDC memory is unchanged).
6	USER_BUFFER_LOAD	User Buffer Only Load Mode: Blocks the automatic transfer from User Buffer to Display Buffer.
7	BUFFERS_SWAP	Buffer Swap Mode: All pixels are alternatively assigned to the state defined in the User Buffer, then to the state defined in the Display Buffer at LCDBLKFREQ frequency.

- **LCDBLKFREQ: LCD Blinking Frequency Selection**

(Processed at beginning of next frame).

Blinking frequency = Frame Frequency/LCDBLKFREQ[7:0].

Note: 0 written in LCDBLKFREQ stops blinking.

39.8.5 SLCDC Status Register

Name: SLCDC_SR

Address: 0x4003C010

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
–	–	–	–	–	–	–	ENA

- **ENA: Enable Status (Automatically Set/Reset)**

0 = The SLCDC is disabled.

1 = The SLCDC is enabled.

39.8.6 SLCDC Interrupt Enable Register

Name: SLCDC_IER

Address: 0x4003C020

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
–	–	–	–	–	DIS	–	ENDFRAME

- **ENDFRAME: End of Frame Interrupt Enable**

0 = No effect.

1 = Enables the corresponding interrupt.

- **DIS: SLCDC Disable Completion Interrupt Enable**

0 = No effect.

1 = Enables the corresponding interrupt.

39.8.7 SLCDC Interrupt Disable Register

Name: SLCDC_IDR

Address: 0x4003C024

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
–	–	–	–	–	DIS	–	ENDFRAME

- **ENDFRAME: End of Frame Interrupt Disable**

0 = No effect.

1 = Disables the corresponding interrupt.

- **DIS: SLCDC Disable Completion Interrupt Disable**

0 = No effect.

1 = Disables the corresponding interrupt.

39.8.8 SLCDC Interrupt Mask Register

Name: SLCDC_IMR

Address: 0x4003C028

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
–	–	–	–	–	DIS	–	ENDFRAME

- **ENDFRAME: End of Frame Interrupt Mask**

0 = The corresponding interrupt is not enabled.

1 = The corresponding interrupt is enabled.

- **DIS: SLCDC Disable Completion Interrupt Mask**

0 = The corresponding interrupt is not enabled.

1 = The corresponding interrupt is enabled.

39.8.9 SLCDC Interrupt Status Register

Name: SLCDC_ISR

Address: 0x4003C02C

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
–	–	–	–	–	DIS	–	ENDFRAME

- **ENDFRAME: End of Frame Interrupt Status**

0 = No End of Frame occurred since the last read.

1 = End of Frame occurred since the last read.

- **DIS: SLCDC Disable Completion Interrupt Status**

0 = The SLCDC is enabled.

1 = The SLCDC is disabled.

39.8.10 SLCDC Segment Map Register 0

Name: SLCDC_SMR0

Address: 0x4003C030

Access: Read/Write

31	30	29	28	27	26	25	24
LCD31	LCD30	LCD29	LCD28	LCD27	LCD26	LCD25	LCD24
23	22	21	20	19	18	17	16
LCD23	LCD22	LCD21	LCD20	LCD19	LCD18	LCD17	LCD16
15	14	13	12	11	10	9	8
LCD15	LCD14	LCD13	LCD12	LCD11	LCD10	LCD9	LCD8
7	6	5	4	3	2	1	0
LCD7	LCD6	LCD5	LCD4	LCD3	LCD2	LCD1	LCD0

- **LCDx: LCD Segment Mapped on SEGx I/O Pin**

(For safety reasons, can be configured when SLCDC is disabled).

0 = The corresponding I/O pin is driven either by SLCDC or digital function, depending on the SEGSEL field configuration in the SLCDC_MR.

1 = An LCD segment is driven on the corresponding I/O pin.

39.8.11 SLCDC Segment Map Register 1

Name: SLCDC_SMR1

Address: 0x4003C034

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	LCD49	LCD48

15	14	13	12	11	10	9	8
LCD47	LCD46	LCD45	LCD44	LCD43	LCD42	LCD41	LCD40

7	6	5	4	3	2	1	0
LCD39	LCD38	LCD37	LCD36	LCD35	LCD34	LCD33	LCD32

- **LCDx: LCD Segment Mapped on SEGx I/O Pin**

(For safety reasons, can be configured when SLCDC is disabled).

0 = The corresponding I/O pin is driven either by SLCDC or digital function, depending on the SEGSEL field configuration in the SLCDC_MR.

1 = An LCD segment is driven on the corresponding I/O pin.

39.8.12 SLCDC LSB Memory Register

Name: SLCDC_LMEMRx [x = 0..5]

Address: 0x4003C200 [0], 0x4003C208 [1], 0x4003C210 [2], 0x4003C218 [3], 0x4003C220 [4], 0x4003C228 [5]

Access: Read/Write

31	30	29	28	27	26	25	24
LPIXEL							

23	22	21	20	19	18	17	16
LPIXEL							

15	14	13	12	11	10	9	8
LPIXEL							

7	6	5	4	3	2	1	0
LPIXEL							

- **LPIXEL: LSB Pixels pattern associated to COMx terminal**

0 = The pixel associated to COMx terminal is not visible (if Non-inverted Display mode is used).

1 = The pixel associated to COMx terminal is visible (if Non-inverted Display mode is used).

Note: LPIXEL[n] (n = 0..31) drives SEGn terminal.

39.8.13 SLCDC MSB Memory Register

Name: SLCDC_MMEMRx [x = 0..5]

Address: 0x4003C204 [0], 0x4003C20C [1], 0x4003C214 [2], 0x4003C21C [3], 0x4003C224 [4], 0x4003C22C [5]

Access: Read/Write

31	30	29	28	27	26	25	24
MPIXEL							

23	22	21	20	19	18	17	16
MPIXEL							

15	14	13	12	11	10	9	8
MPIXEL							

7	6	5	4	3	2	1	0
MPIXEL							

- **MPIXEL: MSB Pixels pattern associated to COMx terminal**

0 = The pixel associated to COMx terminal is not visible (if Non-inverted Display mode is used).

1 = The pixel associated to COMx terminal is visible (if Non-inverted Display mode is used).

Note: MPIXEL[n] (n = 32..49) drives SEGn terminal.

40. Analog-to-Digital Converter (ADC)

40.1 Description

The ADC is based on a 10-bit Analog-to-Digital Converter (ADC) managed by an ADC Controller. It also integrates an 8-to-1 analog multiplexer, making possible the analog-to-digital conversions of 8 analog lines. The conversions extend from 0V to the voltage carried on pin ADVREF or the voltage provided by the internal reference voltage which can be programmed in the Analog Control register (ADC_ACR). Selection of the reference voltage source is defined by the ONREF and FORCEREF bits in the Analog Control Register (ADC_ACR).

The ADC supports the 8-bit or 10-bit Resolution mode. The 8-bit resolution mode prevents using the 16-bit peripheral DMA transfer into memory when only 8-bit resolution is required by the application. Note that using this low resolution mode does not increase the conversion rate.

Conversion results are reported in a common register for all channels, as well as in a channel-dedicated register.

The 11-bit and 12-bit resolution modes are obtained by averaging multiple samples to decrease quantization noise. For 11-bit mode, four samples are used, giving an effective sample rate of 1/4 of the actual sample frequency. For 12-bit mode, 16 samples are used, giving an effective sample rate of 1/16th of the actual sample frequency.This allows conversion speed to be traded for better accuracy.

The last channel is internally connected to a temperature sensor. The processing of this channel can be fully configured for efficient downstream processing due to the slow frequency variation of the value carried on such a sensor. The seventh channel is reserved for measurement of VDDBU voltage.

The software trigger, the external trigger on rising edge of the ADTRG pin or internal triggers from Timer Counter output(s) are configurable.

The main 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 ADC also integrates a Sleep mode and a conversion sequencer, and connects with a PDC channel. These features reduce both power consumption and processor intervention.

Finally, the user can configure ADC timings, such as startup time and tracking time.

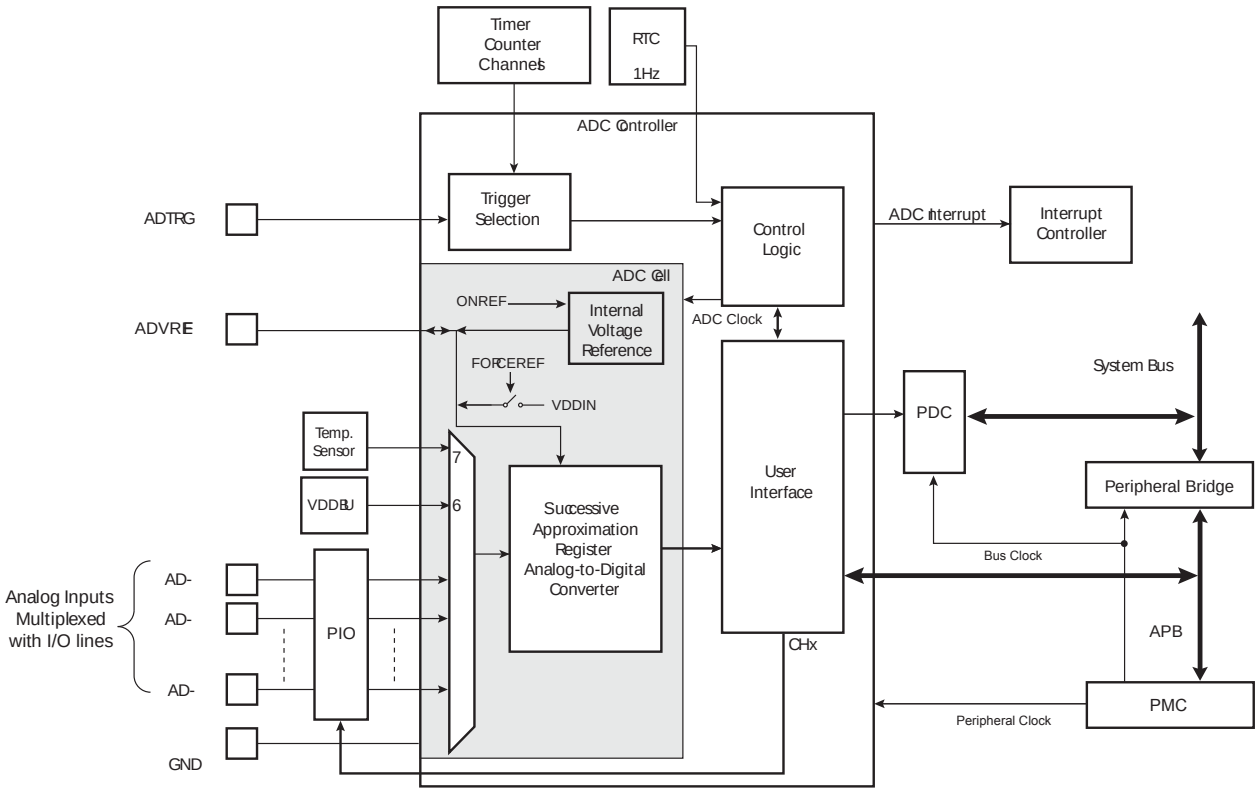
Note: Please check ADC available signals in the main Block Diagram Figure 2-1.

40.2 Embedded Characteristics

- 10-bit Resolution with Enhanced Mode up to 12 Bits
- 500 kHz Conversion Rate
- Digital Averaging Function Provides Enhanced Resolution Mode up to 12 Bits
- On-chip Temperature Sensor Management
- Wide Range of Power Supply Operation
- Selectable External Voltage Reference or Programmable Internal Reference
- Integrated Multiplexer Offering Up to 8 Independent Analog Inputs
- Individual Enable and Disable of Each Channel
- Hardware or Software Trigger
  - External Trigger Pin
  - Timer Counter Outputs (Corresponding TIOA Trigger)
- PDC Support
- Possibility of ADC Timings Configuration
- Two Sleep Modes and Conversion Sequencer
  - Automatic Wakeup on Trigger and Back to Sleep Mode after Conversions of all Enabled Channels
  - Possibility of Customized Channel Sequence
- Standby Mode for Fast Wakeup Time Response
  - Power Down Capability
- Automatic Window Comparison of Converted Values
- Register Write Protection

40.3 Block Diagram

Figure 40-1. Analog-to-Digital Converter Block Diagram



40.4 Signal Description

Table 40-1. ADC Pin Description

Pin Name	Description
ADVREF	External Reference voltage
AD0 - AD7[1][2][3]	Analog input channels
ADTRG	External trigger

Notes: 1. AD7 is not an actual pin; it is internally connected to a temperature sensor.
2. AD6 is not an actual pin; it is internally connected to VDDBU.
3. AD2 is not bounded to an external pin.

40.5 Product Dependencies

40.5.1 Power Management

The ADC Controller is not continuously clocked. The programmer must first enable the ADC Controller peripheral clock in the Power Management Controller (PMC) before using the ADC Controller. However, if the application does not require ADC operations, the ADC Controller clock can be stopped when not needed and restarted when necessary. Configuring the ADC Controller does not require the ADC Controller clock to be enabled.

40.5.2 Interrupt Sources

The ADC interrupt line is connected on one of the internal sources of the Interrupt Controller. Using the ADC interrupt requires the interrupt controller to be programmed first.

Table 40-2. Peripheral IDs

Instance	ID
ADC	29

40.5.3 Analog Inputs

The analog input pins can be multiplexed with PIO lines. In this case, the assignment of the ADC input is automatically done as soon as the corresponding channel is enabled by writing the Channel Enable register (ADC_CHER). By default, after reset, the PIO line is configured as a digital input with its pull-up enabled, and the ADC input is connected to the GND.

40.5.4 Temperature Sensor

The temperature sensor is internally connected to channel index 7 of the ADC.

The temperature sensor provides an output voltage V_T that is proportional to the absolute temperature (PTAT). To activate the temperature sensor, the TEMPON bit in the Temperature Sensor Mode register (ADC_TEMPMR) must be set. After setting the bit, the startup time of the temperature sensor must be achieved prior to initiating any measurement.

40.5.5 I/O Lines

The digital input ADTRG is multiplexed with digital functions on the I/O line and the selection of ADTRG is made using the PIO controller by configuring the I/O Input mode.

The analog inputs ADx are multiplexed with digital functions on the I/O lines. ADx inputs are selected as inputs of the ADCC when writing a one in the corresponding CHx bit of ADC_CHER and the digital functions are not selected.

Table 40-3. I/O Lines

Instance	Signal	I/O Line	Peripheral
ADC	ADTRG	PB23	A
ADC	COM4/AD1	PA4	X1
ADC	SEG6/AD0	PA12	X1
ADC	SEG31/AD3	PB13	X1
ADC	SEG41/AD4	PB23	X1
ADC	SEG49/AD5	PB31	X1

40.5.6 Timer Triggers

Timer Counters may or may not be used as hardware triggers depending on user requirements. Thus, some or all of the timer counters may be unconnected.

40.5.7 Conversion Performances

For performance and electrical characteristics of the ADC, see the section “Electrical Characteristics”.

40.6 Functional Description

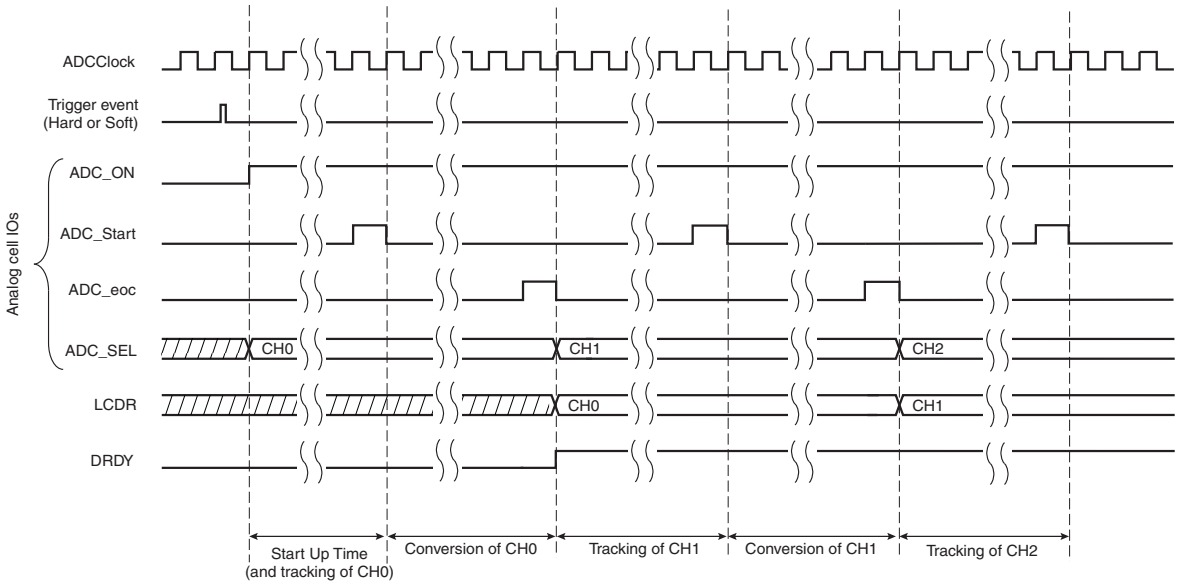
40.6.1 Analog-to-digital Conversion

The ADC uses the ADC Clock to perform conversions. Converting a single analog value to a 10-bit digital data requires tracking clock cycles as defined in the field TRACKTIM of the "ADC Mode Register" (ADC_MR). The ADC clock frequency is selected in the PRESCAL field of ADC_MR.

The ADC clock frequency is between $f_{peripheral\ clock}/2$ if PRESCAL is 0, and $f_{peripheral\ clock}/512$ if PRESCAL is set to 255 (0xFF).

PRESCAL must be programmed in order to provide an ADC clock frequency according to the parameters given in the section "Electrical Characteristics".

Figure 40-2. Sequence of ADC Conversions



40.6.2 Conversion Reference

The conversion is performed on a full range between 0V and the reference voltage. The reference voltage is defined by the external pin ADVREF, or programmed using the internal reference voltage configured in ADC_ACR. Analog inputs between these voltages convert to values based on a linear conversion.

40.6.3 Conversion Resolution

The ADC supports 8-bit or 10-bit resolutions. The 8-bit selection is performed by setting the LOWRES bit in ADC_MR. By default, after a reset, the resolution is the highest and the DATA field in the data registers is fully used. By setting the LOWRES bit, the ADC switches to the lowest resolution and the conversion results can be read in the lowest significant bits of the data registers. The two highest bits of the DATA field in the corresponding Channel Data register (ADC_CDR) and of the LDATA field in the Last Converted Data register (ADC_LCDR) read 0.

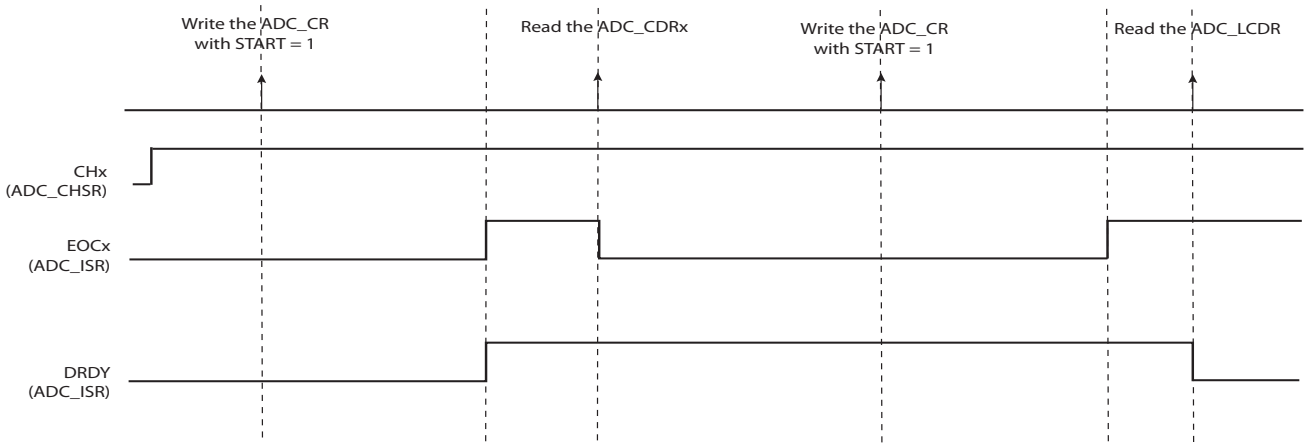
40.6.4 Conversion Results

When a conversion is completed, the resulting 10-bit digital value is stored in ADC_CDRx of the current channel and in ADC_LCDR. By setting the TAG bit in the Extended Mode register (ADC_EMR), ADC_LCDR presents the channel number associated with the last converted data in the CHNB field.

The EOCx and DRDY bits in the Interrupt Status register (ADC_ISR) are set. In the case of a connected PDC channel, DRDY rising triggers a data request. In any case, both EOC and DRDY can trigger an interrupt.

Reading one ADC_CDRx clears the corresponding EOCx bit. Reading ADC_LCDR clears the DRDY bit.

Figure 40-3. EOCx and DRDY Flag Behavior

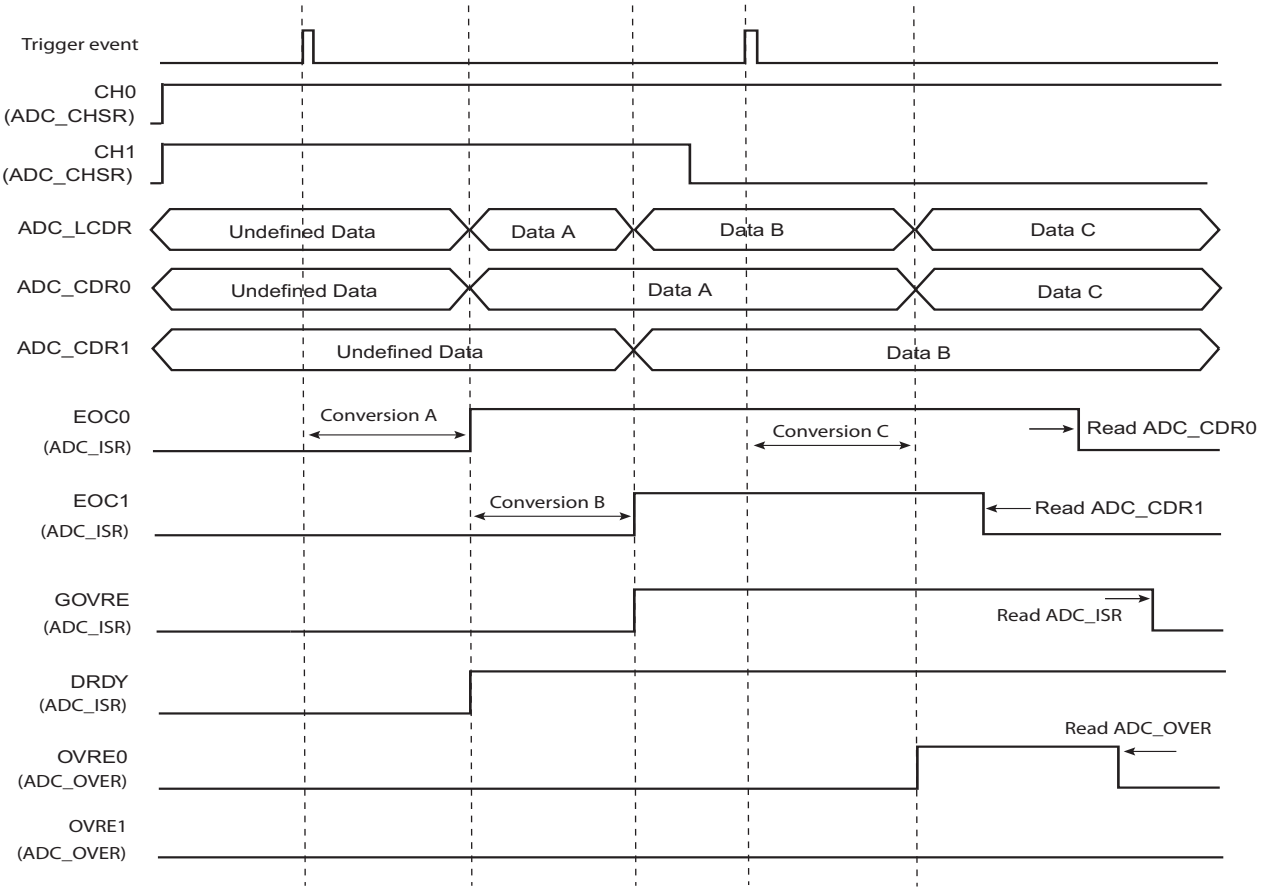


If ADC_CDR is not read before further incoming data is converted, the corresponding Overrun Error (OVREx) flag is set in the Overrun Status register (ADC_OVER).

New data converted when DRDY is high sets the GOVRE bit in ADC_ISR.

The OVREx flag is automatically cleared when ADC_OVER is read, and GOVRE flag is automatically cleared when ADC_ISR is read.

Figure 40-4. EOCx, GOVRE and OVREx Flag Behavior



Warning: If the corresponding channel is disabled during a conversion or if it is disabled and then re-enabled during a conversion, the associated data and the corresponding EOCx and GOVRE flags in ADC_ISR and OVREx flags in ADC_OVER are unpredictable.

40.6.5 Conversion Triggers

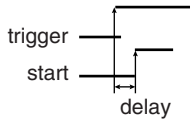
Conversions of the active analog channels are started with a software or hardware trigger. The software trigger is provided by writing the Control register (ADC_CR) with the START bit at 1.

The hardware trigger can be one of the TIOA outputs of the Timer Counter channels or the external trigger input of the ADC (ADTRG). The hardware trigger is selected with the TRGSEL field in ADC_MR. The selected hardware trigger is enabled with the TRGEN bit in ADC_MR.

The minimum time between two consecutive trigger events must be strictly greater than the duration time of the longest conversion sequence as configured in ADC_MR, ADC_CHSR and ADC_SEQR1.

If a hardware trigger is selected, the start of a conversion is triggered after a delay which starts 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 ADC clock period.

Figure 40-5. Hardware Trigger Delay



If one of the TIOA outputs is selected, the corresponding Timer Counter channel must be programmed in Waveform mode.

Only one start command is necessary to initiate a conversion sequence on all the channels. The ADC hardware logic automatically performs the conversions on the active channels, then waits for a new request. The Channel Enable (ADC_CHER) and Channel Disable (ADC_CHDR) registers permit the analog channels to be enabled or disabled independently.

If the ADC is used with a PDC, only the transfers of converted data from enabled channels are performed and the resulting data buffers should be interpreted accordingly.

40.6.6 Sleep Mode and Conversion Sequencer

The ADC Sleep mode maximizes power saving by automatically deactivating the ADC when it is not being used for conversions. Sleep mode is selected by setting the SLEEP bit in ADC_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 ADC. See the section “ADC Characteristics” in the “Electrical Characteristics”.

When a start conversion request occurs, the ADC is automatically activated. As the analog cell requires a start-up time, the logic waits during this time and starts the conversion on the enabled channels. When all conversions are complete, the ADC is deactivated until the next trigger. Triggers occurring during the sequence are ignored.

The conversion sequencer allows automatic processing with minimum processor intervention and optimized power consumption. Conversion sequences can be performed periodically using a Timer/Counter output. By using the PDC, the periodic acquisition of several samples can be processed automatically without processor intervention.

The sequence can be customized by programming ADC_SEQR1 and setting the USEQ bit of ADC_MR. The user can choose a specific order of channels and can program up to 8 conversions by sequence. The user is free to create a personal sequence by writing channel numbers in ADC_SEQR1. Not only can channel numbers be written in any sequence, channel numbers can be repeated several times. When the bit USEQ in ADC_MR is set, the fields USCHx in ADC_SEQR1 are used to define the sequence. Only enabled USCHx fields are part of the sequence. Each USCHx

field has a corresponding enable, CHx, in ADC_CHER (USCHx field with the lowest x index is associated with bit CHx of the lowest index).

40.6.7 Comparison Window

The ADC Controller features automatic comparison functions. It compares converted values to a low threshold, a high threshold or both, depending on the value of the CMPMODE field in the Extended Mode register (ADC_EMR). The comparison can be done on all channels or only on the channel specified in the CMPSEL field of ADC_EMR. To compare all channels, the CMPALL bit of ADC_EMR should be set.

Moreover, a filtering option can be set by writing the number of consecutive comparison events needed to raise the flag. This number can be written and read in the CMPFILTER field of ADC_EMR.

The flag can be read on the COMPE bit of ADC_ISR and can trigger an interrupt.

The high threshold and the low threshold can be read/write in the Compare Window register (ADC_CWR).

If the comparison window is to be used with the LOWRES bit set in ADC_MR, the thresholds do not need to be adjusted, as the adjustment is done internally. Whether or not the LOWRES bit is set, thresholds must always be configured in accordance with the maximum ADC resolution.

40.6.8 ADC Timings

Each ADC has its own minimal startup time that is programmed through the field STARTUP in ADC_MR.

A minimal tracking time is necessary for the ADC to guarantee the best converted final value between two channel selections. This time must be programmed in the TRACKTIM field in ADC_MR.

Warning: No input buffer amplifier to isolate the source is included in the ADC. This must be taken into consideration to program a precise value in the TRACKTIM field. See the section “ADC Characteristics” in the “Electrical Characteristics”.

40.6.9 Temperature Sensor

The temperature sensor is internally connected to channel index 7. To enable temperature measurement, the TEMPON bit must be set in ADC_TEMPMR.

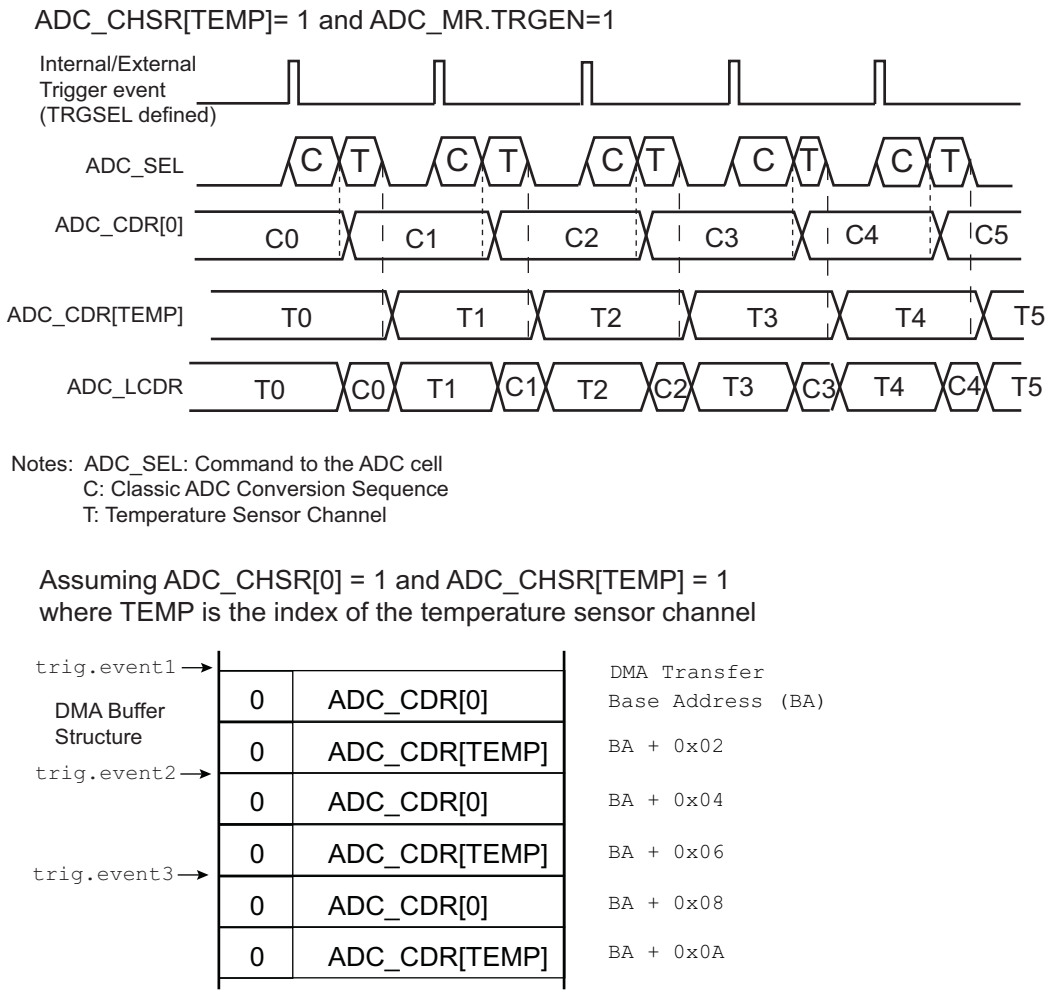
The ADC Controller manages temperature measurement in several ways. The different methods of measurement depend on the configuration bits TRGEN in ADC_MR and CH7 in ADC_CHSR.

Temperature measurement can be triggered like the other channels by enabling its associated conversion channel index 7, writing 1 in CH7 of ADC_CHER.

A manual start can only be performed if the TRGEN bit in ADC_MR is cleared. When the START bit in ADC_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 the ADC_CDR7 register and the associated flag EOC7 is set in ADC_ISR.

If the TRGEN bit is set in ADC_MR, the channel of the temperature sensor is periodically converted together with other enabled channels. The result is placed in the registers ADC_LCDR and ADC_CDR7. Thus the temperature conversion result is part of the Peripheral DMA Controller buffer. The temperature channel can be enabled/disabled at anytime, however this may not be optimal for downstream processing.

Figure 40-6. Non-optimized Temperature Conversion



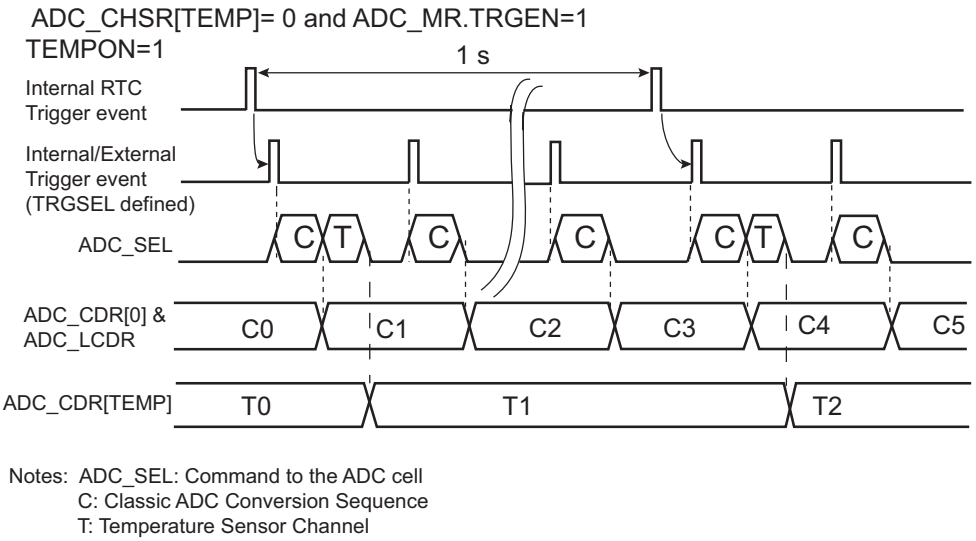
The temperature factor has a slow variation rate and is potentially different from other conversion channels. As a result, the ADC Controller triggers the measurement differently when TEMPON is set in ADC_TEMPMR but CH7 is not set in the ADC_CHSR.

Under these conditions, the measurement is triggered every second by means of an internal trigger generated by the RTC, always enabled and totally independent of the internal/external triggers. The RTC event will be processed on the next internal/external trigger event as described in Figure 40-7, "Optimized Temperature Conversion Combined With Classical Conversions". The internal/external trigger is selected through the TRGSEL field of ADC_MR.

In this mode of operation, the temperature sensor is only powered for a period of time covering the startup time and conversion time (refer to Figure 40-8, "Temperature Conversion Only").

Every second, a conversion is scheduled for channel 7 but the result of the conversion is only uploaded in ADC_CDR7 and not in ADC_LCDR. Therefore there is no change in the structure of the Peripheral DMA Controller buffer due to the conversion of the temperature channel; only the enabled channels are kept in the buffer. The end of conversion of the temperature channel is reported by means of EOC7 flag in ADC_ISR.

Figure 40-7. Optimized Temperature Conversion Combined With Classical Conversions

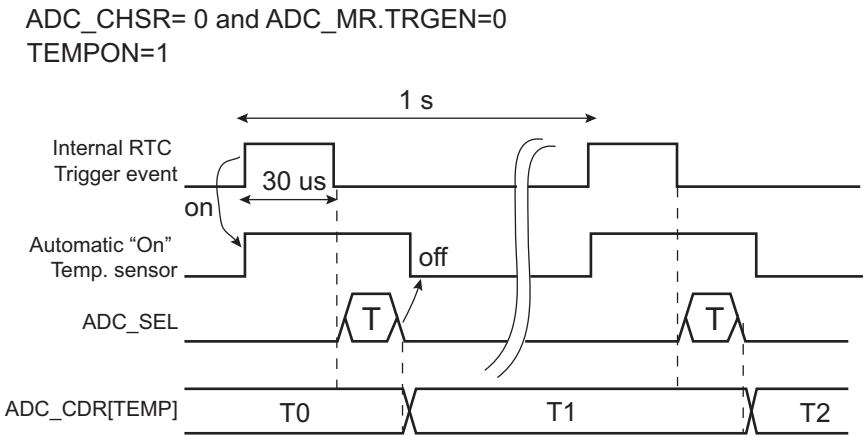


Notes: ADC_SEL: Command to the ADC cell
C: Classic ADC Conversion Sequence
T: Temperature Sensor Channel

If TEMPON = 1, TRGEN is disabled and none of the channels are enabled in ADC_CHSR (ADC_CHSR = 0), then only channel 7 is converted at a rate of one conversion per second (see Figure 40-8, "Temperature Conversion Only").

This mode of operation, when combined with the Sleep mode operation of the ADC Controller, provides a low-power mode for temperature measurement. This assumes there is no other ADC conversion to schedule at a high sampling rate, or no other channel to convert.

Figure 40-8. Temperature Conversion Only



Notes: ADC_SEL: Command to the ADC cell
C: Classic ADC Conversion Sequence
T: Temperature Sensor Channel

It is possible to raise a flag only if there is a predefined change in the temperature. The user can define a range of temperature or a threshold in the Temperature Compare Window register (ADC_TEMPCWR), and the mode of comparison that can be programmed into the TEMPCMPMOD field into ADC_TEMPMR. These values define how the TEMPCHG flag is raised in ADC_ISR.

The TEMPCHG flag can be used to generate a temperature-dependent interrupt instead of the end-of-conversion interrupt. More specifically, the interrupt is generated only if the temperature sensor as measured by the ADC reports a temperature value below, above, inside or outside programmable thresholds (see ADC_TEMPMR).

In any case, if TEMPON is set, the temperature can be read at anytime in ADC_CDR7 without any specific software intervention.

40.6.10 VDDBU Measurement

The seventh ADC channel (CH6) of the ADC Controller is reserved for measurement of the VDDBU power supply pin. For this channel, setting up, starting conversion, and other tasks must be performed the same way as for all other channels. VDDBU is measured without any attenuation. This means that for VDDBU greater than the voltage reference applied to the ADC, the digital output clamps to the maximum value.

40.6.11 Enhanced Resolution Mode and Digital Averaging Function

The Enhanced Resolution mode is enabled if LOWRES is cleared in ADC_MR, and the OSR field is set to 1 or 2 in ADC_EMR. The enhancement is based on a digital averaging function.

FREERUN in ADC_MR must be cleared when digital averaging is used (OSR not equal to 0 in ADC_EMR).

There is no averaging on the last index channel if the measure is triggered by an RTC event (see Section 40.6.9 ”Temperature Sensor”).

In Enhanced Resolution mode, the ADC Controller trades conversion speed for quantization noise by averaging multiple samples, thus providing a digital low-pass filter function.

If 1-bit enhancement resolution is selected (OSR = 1 in ADC_EMR), the ADC real sample rate is the maximum ADC sample rate divided by 4. Thus, the oversampling ratio is 4.

When the 2-bit enhancement resolution is selected (OSR = 2 in ADC_EMR), the ADC real sample rate is the maximum ADC sample rate divided by 16 (oversampling ratio is 16).

The selected oversampling ratio applies to all enabled channels except for the temperature sensor channel when triggered by an RTC event.

The average result is valid into the ADC_CDRx register (x corresponding to the index of the channel) only if EOCn flag is set in ADC_ISR and OVREn flag is cleared in ADC_OVER. The average result for all channels is valid in ADC_LCDR only if DRDY is set and GOVRE is cleared in ADC_ISR.

Note that registers ADC_CDRx are not buffered. Therefore, when an averaging sequence is ongoing, the value in these registers changes after each averaging sample. However, overrun flags in ADC_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.

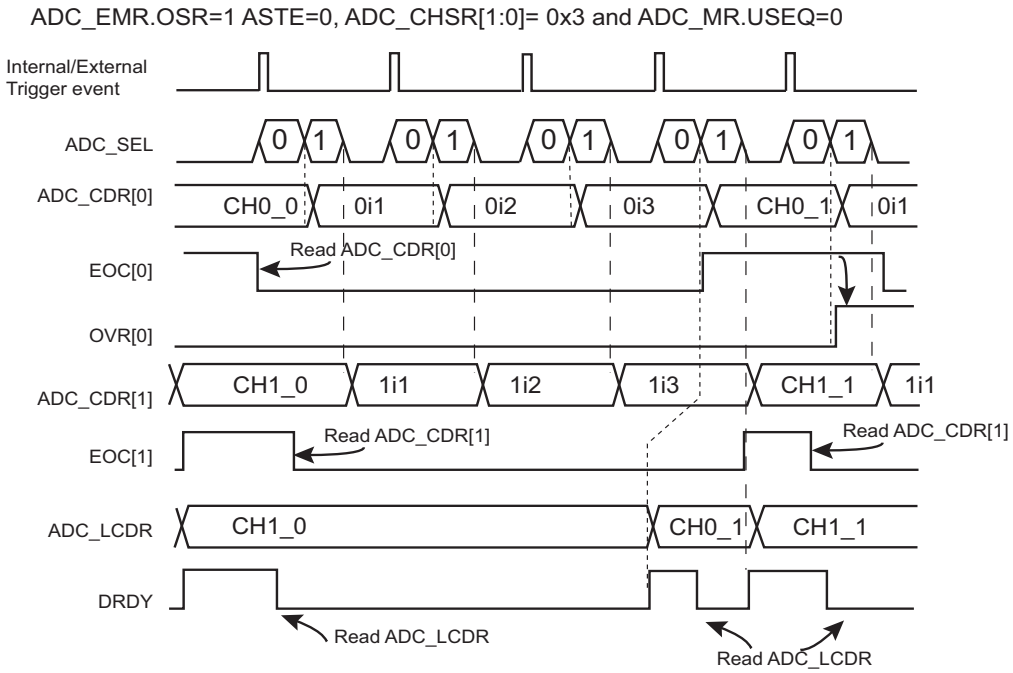
As a result, when an overrun flag rises in ADC_OVER, the previous unread data is lost. However, the data has not been overwritten by the new averaged value, as the averaging sequence for this channel may still be on-going.

40.6.11.1 Averaging Function versus Trigger Events

The samples can be defined in different ways for the averaging function depending on the configuration of the ASTE bit in ADC_EMR and the USEQ bit in ADC_MR.

When USEQ is cleared, there are two ways to generate the averaging through the trigger event. If ASTE is cleared in ADC_EMR, every trigger event generates one sample for each enabled channel as described in Figure 40-9, "Digital Averaging Function Waveforms over Multiple Trigger Events". Therefore, four trigger events are requested to get the result of averaging if OSR = 1.

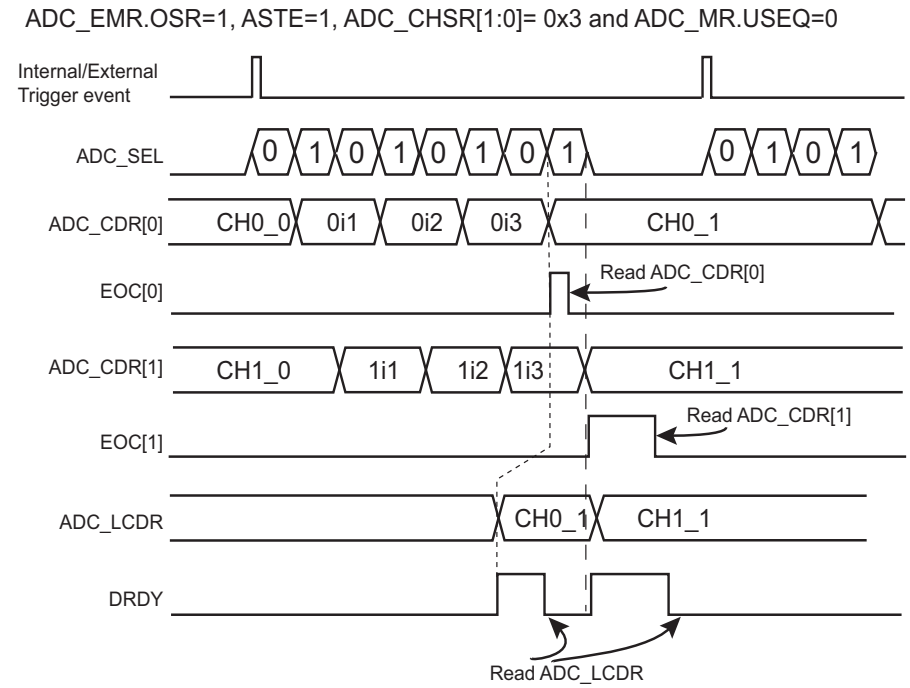
Figure 40-9. Digital Averaging Function Waveforms over Multiple Trigger Events



Notes: ADC_SEL: Command to the ADC cell
0i1,0i2,0i3, 1i1, 1i2, 1i3 are intermediate results and CH0/1_0/1 are final results of average function.

If ASTE = 1 in ADC_EMR and USEQ = 0 in ADC_MR, then the sequence to be converted, defined in ADC_CHSR, is automatically repeated n times, where n corresponds to the oversampling ratio defined in the OSR field in ADC_EMR. As a result, only one trigger is required to obtain the result of the averaging function as described in Figure 40-10, "Digital Averaging Function Waveforms on a Single Trigger Event".

Figure 40-10. Digital Averaging Function Waveforms on a Single Trigger Event



Notes: ADC_SEL: Command to the ADC cell
0i1,0i2,0i3, 1i1, 1i2, 1i3 are intermediate results and CH0/1_0/1 are final results of average function.

When USEQ is set, the user can define the channel sequence to be converted by configuring ADC_SEQRx and ADC_CHER so that channels are not interleaved during the averaging period. Under these conditions, a sample is defined for each end of conversion as shown in Figure 40-11, "Digital Averaging Function Waveforms on Single Trigger Event, Non-interleaved".

Therefore, if the same channel is configured to be converted four times consecutively, and OSR = 1 in ADC_EMR, the averaging result is placed in the corresponding Channel Data Register (ADC_CDRx) and Last Converted Data Register (ADC_LCDR) for each trigger event.

In this case, the ADC real sample rate remains the maximum ADC sample rate divided by 4 or 16, depending on OSR.

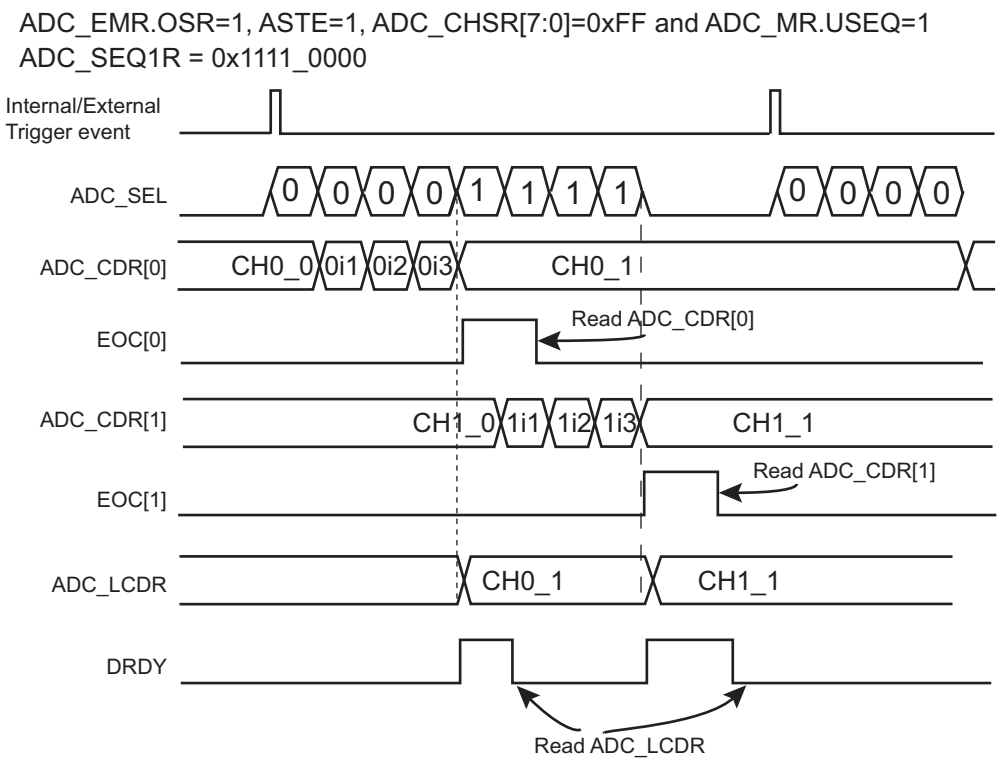
When USEQ = 1, ASTE = 1 and OSR is different from 0, it is important to note that the user sequence must follow a specific pattern. The user sequence must be programmed so that it generates a stream of conversion, where a given channel is successively converted with an integer multiple depending on the value of OSR. Up to four channels can be converted in this specific mode.

When OSR = 1, each channel to convert must be repeated four times consecutively in the sequence, so the first four single bits enabled in ADC_CHSR must have the associated channel index programmed to the same value in ADC_SEQ1/2. Therefore, for OSR = 1, a maximum of four channels can be converted. The user sequence allows a maximum of 16 conversions for each trigger event.

When OSR = 2, a channel to convert must be repeated 16 times consecutively in the sequence, so all fields must be enabled in the ADC_CHSR register, and their associated channel index programmed to the same value in ADC_SEQ1/2. Therefore, for OSR = 2, only one channel can be converted. The user sequence allows a maximum of 16 conversions for each trigger event.

OSR = 3 and OSR = 4 are prohibited when USEQ = 1 and ASTE = 1.

Figure 40-11. Digital Averaging Function Waveforms on Single Trigger Event, Non-interleaved



Notes: ADC_SEL: Command to the ADC cell
0i1,0i2,0i3, 1i1, 1i2, 1i3 are intermediate results and CH0/1_0/1 are final results of average function.

40.6.11.2 Oversampling Digital Output Range

When an oversampling is performed, the maximum value that can be read on ADC_CDRx or ADC_LCDR is not the full scale value, even if the maximum voltage is supplied on the analog input. This is due to the digital averaging algorithm. For example, when OSR = 1, four samples are accumulated and the result is then right-shifted by 1 (divided by 2).

The maximum output value carried on ADC_CDRx or ADC_LCDR depends on the configuration of the field OSR in ADC_EMR.

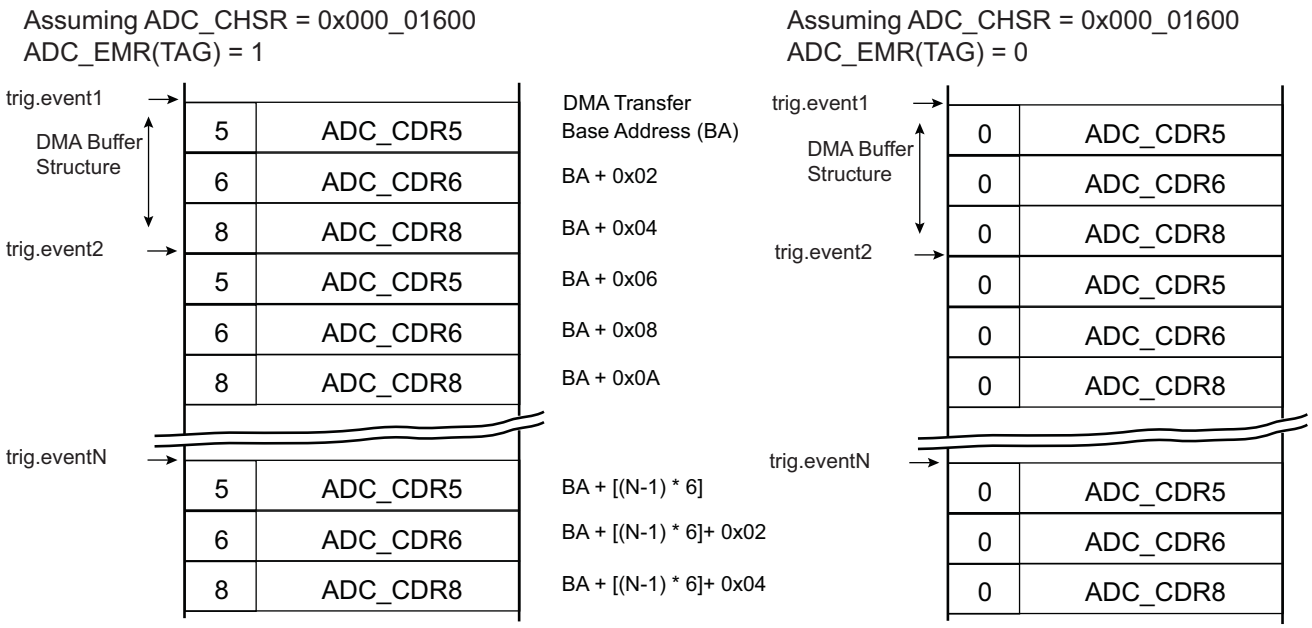
Table 40-4. Oversampling Digital Output Range Values

Resolution	Samples	Shift	Full Scale Value	Maximum Value
8-bit	1	0	255	255
10-bit	1	0	1023	1023
11-bit	4	1	2047	2046
12-bit	16	2	4095	4092

40.6.12 Buffer Structure

The PDC read channel is triggered each time a new data is stored in ADC_LCDR. The same data structure is repeatedly stored in ADC_LCDR each time a trigger event occurs. Depending on the user mode of operation (ADC_MR, ADC_CHSR, ADC_SEQR1), the structure differs. Each data read to the PDC buffer, carried on a half-word (16-bit), consists of the last converted data right-aligned. When TAG is set in ADC_EMR, the four most significant bits carry the channel number, thus simplifying post-processing in the PDC buffer or improved checking of the PDC buffer integrity.

Figure 40-12. Buffer Structure



40.6.13 Register Write Protection

To prevent any single software error from corrupting ADC behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the "ADC Write Protection Mode Register" (ADC_WPMR).

If a write access to a write-protected register is detected, the WPVS flag in the "ADC Write Protection Status Register" (ADC_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 ADC_WPSR.

The following registers can be write-protected:

- "ADC Mode Register"
- "ADC Channel Sequence 1 Register"
- "ADC Channel Enable Register"
- "ADC Channel Disable Register"
- "ADC Temperature Sensor Mode Register"
- "ADC Temperature Compare Window Register"
- "ADC Extended Mode Register"
- "ADC Compare Window Register"
- "ADC Analog Control Register"

40.7 Analog-to-Digital Converter (ADC) User Interface

Table 40-5. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	ADC_CR	Write-only	–
0x04	Mode Register	ADC_MR	Read/Write	0x00000000
0x08	Channel Sequence 1 Register	ADC_SEQR1	Read/Write	0x00000000
0x0C	Reserved	–	–	–
0x10	Channel Enable Register	ADC_CHER	Write-only	–
0x14	Channel Disable Register	ADC_CHDR	Write-only	–
0x18	Channel Status Register	ADC_CHSR	Read-only	0x00000000
0x1C	Reserved	–	–	–
0x20	Last Converted Data Register	ADC_LCDR	Read-only	0x00000000
0x24	Interrupt Enable Register	ADC_IER	Write-only	–
0x28	Interrupt Disable Register	ADC_IDR	Write-only	–
0x2C	Interrupt Mask Register	ADC_IMR	Read-only	0x00000000
0x30	Interrupt Status Register	ADC_ISR	Read-only	0x00000000
0x34	Temperature Sensor Mode Register	ADC_TEMPMR	Read/Write	0x00000000
0x38	Temperature Compare Window Register	ADC_TEMPCWR	Read/Write	0x00000000
0x3C	Overrun Status Register	ADC_OVER	Read-only	0x00000000
0x40	Extended Mode Register	ADC_EMR	Read/Write	0x00000000
0x44	Compare Window Register	ADC_CWR	Read/Write	0x00000000
0x50	Channel Data Register 0	ADC_CDR0	Read-only	0x00000000
0x54	Channel Data Register 1	ADC_CDR1	Read-only	0x00000000
...	...	...	...	...
0x6C	Channel Data Register 7	ADC_CDR7	Read-only	0x00000000
0x70 - 0x90	Reserved	–	–	–
0x94	Analog Control Register	ADC_ACR	Read/Write	0x00000000
0x98 - 0xE0	Reserved	–	–	–
0xE4	Write Protection Mode Register	ADC_WPMR	Read/Write	0x00000000
0xE8	Write Protection Status Register	ADC_WPSR	Read-only	0x00000000
0xEC - 0xF8	Reserved	–	–	–
0xFC	Reserved	–	–	–
0x100 - 0x124	Reserved for PDC registers	–	–	–

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

40.7.1 ADC Control Register

Name: ADC_CR

Address: 0x40038000

Access: Write-only

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

- **SWRST: Software Reset**

0 = No effect.

1 = Resets the ADC simulating a hardware reset.

- **START: Start Conversion**

0 = No effect.

1 = Begins analog-to-digital conversion.

40.7.2 ADC Mode Register

Name: ADC_MR

Address: 0x40038004

Access: Read/Write

31	30	29	28	27	26	25	24
USEQ	–	–	–	TRACKTIM			
23	22	21	20	19	18	17	16
–	–	–	–	STARTUP			
15	14	13	12	11	10	9	8
PRESCAL							
7	6	5	4	3	2	1	0
FREERUN	–	SLEEP	LOWRES	TRGSEL			TRGEN

This register can only be written if the WPEN bit is cleared in “ADC 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	ADC_TRIG0	External trigger ADTRG
1	ADC_TRIG1	Timer Counter Channel 0 Output
2	ADC_TRIG2	Timer Counter Channel 1 Output
3	ADC_TRIG3	Timer Counter Channel 2 Output
4	ADC_TRIG4	Timer Counter Channel 3 Output
5	ADC_TRIG5	Timer Counter Channel 4 Output
6	ADC_TRIG6	Timer Counter Channel 5 Output
7	–	Reserved

- **LOWRES: Resolution**

Value	Name	Description
0	BITS_10	10-bit resolution. For higher resolution by averaging, refer to Section 40.7.15 ”ADC Extended Mode Register”
1	BITS_8	8-bit resolution

- **SLEEP: Sleep Mode**

Value	Name	Description
0	NORMAL	Normal Mode: The ADC core and reference voltage circuitry are kept ON between conversions
1	SLEEP	Sleep Mode: The ADC core and reference voltage circuitry are OFF between conversions

- **FREERUN: Free Run Mode**

Value	Name	Description
0	OFF	Normal Mode
1	ON	Free Run Mode: Never wait for any trigger

Note: FREERUN must be set to 0 when digital averaging is used (OSR differs from 0 in ADC_EMR register).

- **PRESCAL: Prescaler Rate Selection**

$f_{ADC\ Clock} = f_{peripheral\ clock} / ((PRESCAL + 1) \times 2)$

- **STARTUP: Start Up Time**

Value	Name	Description
0	SUT0	0 periods of ADC Clock
1	SUT8	8 periods of ADC Clock
2	SUT16	16 periods of ADC Clock
3	SUT24	24 periods of ADC Clock
4	SUT64	64 periods of ADC Clock
5	SUT80	80 periods of ADC Clock
6	SUT96	96 periods of ADC Clock
7	SUT112	112 periods of ADC Clock
8	SUT512	512 periods of ADC Clock
9	SUT576	576 periods of ADC Clock
10	SUT640	640 periods of ADC Clock
11	SUT704	704 periods of ADC Clock
12	SUT768	768 periods of ADC Clock
13	SUT832	832 periods of ADC Clock
14	SUT896	896 periods of ADC Clock
15	SUT960	960 periods of ADC Clock

- **TRACKTIM: Tracking Time**

Tracking Time = (TRACKTIM + 1) × ADC Clock periods

- **USEQ: User Sequence Enable**

Value	Name	Description
0	NUM_ORDER	Normal Mode: The controller converts channels in a simple numeric order depending only on the channel index
1	REG_ORDER	User Sequence Mode: The sequence respects what is defined in ADC_SEQR1 and can be used to convert the same channel several times

40.7.3 ADC Channel Sequence 1 Register

Name: ADC_SEQR1

Address: 0x40038008

Access: Read/Write

31	30	29	28	27	26	25	24
USCH8				USCH7			
23	22	21	20	19	18	17	16
USCH6				USCH5			
15	14	13	12	11	10	9	8
USCH4				USCH3			
7	6	5	4	3	2	1	0
USCH2				USCH1			

This register can only be written if the WPEN bit is cleared in “ADC Write 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 7. So it is only possible to use the sequencer from CH0 to CH7.

This register activates only if ADC_MR(USEQ) field is set to 1.

Any USCHx field is taken into account only if ADC_CHSR(CHx) register field reads logical 1; else any value written in USCHx does not add the corresponding channel in the conversion sequence.

Configuring the same value in different fields leads to multiple samples of the same channel during the conversion sequence. This can be done consecutively, or not, depending on user needs.

40.7.4 ADC Channel Enable Register

Name: ADC_CHER

Address: 0x40038010

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
CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0

This register can only be written if the WPEN bit is cleared in “ADC Write Protection Mode Register” .

- **CHx: Channel x Enable**

0 = No effect.

1 = Enables the corresponding channel.

Note: If USEQ = 1 in ADC_MR, CHx corresponds to the xth channel of the sequence described in ADC_SEQR1.

40.7.5 ADC Channel Disable Register

Name: ADC_CHDR

Address: 0x40038014

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
CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0

This register can only be written if the WPEN bit is cleared in “ADC 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 re-enabled during a conversion, the associated data and corresponding EOCx and GOVRE flags in ADC_ISR and OVREx flags in ADC_OVER are unpredictable.

40.7.6 ADC Channel Status Register

Name: ADC_CHSR

Address: 0x40038018

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
CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0

- **CHx: Channel x Status**

0 = The corresponding channel is disabled.

1 = The corresponding channel is enabled.

40.7.7 ADC Last Converted Data Register

Name: ADC_LCDR

Address: 0x40038020

Access: Read-only

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

- **LDATA: Last Data Converted**

The analog-to-digital conversion data is placed into this register at the end of a conversion and remains until a new conversion is completed.

- **CHNB: Channel Number**

Indicates the last converted channel when the TAG option is set to 1 in ADC_EMR. If the TAG option is not set, CHNB = 0.

40.7.8 ADC Interrupt Enable Register

Name: ADC_IER

Address: 0x40038024

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	RXBUFF	ENDRX	COMPE	GOVRE	DRDY
23	22	21	20	19	18	17	16
–	–	–	–	TEMPCHG	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
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**
- **TEMPCHG: Temperature Change Interrupt Enable**
- **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**

40.7.9 ADC Interrupt Disable Register

Name: ADC_IDR

Address: 0x40038028

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	RXBUFF	ENDRX	COMPE	GOVRE	DRDY
23	22	21	20	19	18	17	16
–	–	–	–	TEMPCHG	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
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**
- **TEMPCHG: Temperature Change Interrupt Disable**
- **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**

40.7.10 ADC Interrupt Mask Register

Name: ADC_IMR

Address: 0x4003802C

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	RXBUFF	ENDRX	COMPE	GOVRE	DRDY
23	22	21	20	19	18	17	16
–	–	–	–	TEMPCHG	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
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**
- **TEMPCHG: Temperature Change Interrupt Mask**
- **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**

40.7.11 ADC Interrupt Status Register

Name: ADC_ISR

Address: 0x40038030

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	RXBUFF	ENDRX	COMPE	GOVRE	DRDY
23	22	21	20	19	18	17	16
–	–	–	–	TEMPCHG	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
EOC7	EOC6	EOC5	EOC4	EOC3	EOC2	EOC1	EOC0

- **EOCx: End of Conversion x (automatically set / cleared)**

0 = The corresponding analog channel is disabled, or the conversion is not finished. This flag is cleared when reading the corresponding ADC_CDRx registers.

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 ADC_TEMPCWR) since the last read of ADC_ISR.

1 = The temperature value reported on ADC_CDR7 has changed since the last read of ADC_ISR, according to what is defined in ADC_TEMPMR and ADC_TEMPCWR.

- **DRDY: Data Ready (automatically set / cleared)**

0 = No data has been converted since the last read of ADC_LCDR.

1 = At least one data has been converted and is available in ADC_LCDR.

- **GOVRE: General Overrun Error (cleared on read)**

0 = No general overrun error occurred since the last read of ADC_ISR.

1 = At least one general overrun error has occurred since the last read of ADC_ISR.

- **COMPE: Comparison Event (cleared on read)**

0 = No comparison event since the last read of ADC_ISR.

1 = At least one comparison event (defined in ADC_EMR and ADC_CWR) has occurred since the last read of ADC_ISR.

- **ENDRX: End of Receive Transfer (automatically set / cleared)**

0 = The Receive Counter Register has not reached 0 since the last write in ADC_RCR(1) or ADC_RNCR(1).

1 = The Receive Counter Register has reached 0 since the last write in ADC_RCR(1) or ADC_RNCR(1).

- **RXBUFF: Receive Buffer Full (automatically set / cleared)**

0 = ADC_RCR(1) or ADC_RNCR(1) has a value other than 0.

1 = Both ADC_RCR(1) and ADC_RNCR(1) have a value of 0.

Notes: 1. ADC_RCR and ADC_RNCR are PDC registers.

40.7.12 ADC Temperature Sensor Mode Register

Name: ADC_TEMPMR

Address: 0x40038034

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		–	–	–	TEMPON

This register can only be written if the WPEN bit is cleared in “ADC Write Protection Mode Register” .

- **TEMPON: Temperature Sensor ON**

0 = The temperature sensor is not enabled.

1 = The temperature sensor is enabled and the measurements are triggered.

- **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

40.7.13 ADC Temperature Compare Window Register

Name: ADC_TEMPCWR

Address: 0x40038038

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 “ADC Write Protection Mode Register” .

- **TLOWTHRES: Temperature Low Threshold**

Low threshold associated to compare settings of ADC_TEMPMR.

- **THIGHTHRES: Temperature High Threshold**

High threshold associated to compare settings of ADC_TEMPMR.

40.7.14 ADC Overrun Status Register

Name: ADC_OVER

Address: 0x4003803C

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
OVRE7	OVRE6	OVRE5	OVRE4	OVRE3	OVRE2	OVRE1	OVRE0

- **OVREx: Overrun Error x**

0 = No overrun error on the corresponding channel since the last read of ADC_OVER.

1 = There has been an overrun error on the corresponding channel since the last read of ADC_OVER.

Note: An overrun error does not always mean that the unread data has been replaced by a new valid data. Refer to Section 40.6.11 ”Enhanced Resolution Mode and Digital Averaging Function” for details.

40.7.15 ADC Extended Mode Register

Name: ADC_EMR

Address: 0x40038040

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	TAG

23	22	21	20	19	18	17	16
–	–	–	ASTE	–	–	OSR	

15	14	13	12	11	10	9	8
–	–	CMPFILTER		–	–	CMPALL	–

7	6	5	4	3	2	1	0
CMPSEL				–	–	CMPMODE	

This register can only be written if the WPEN bit is cleared in "ADC Write 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.

- **OSR: Over Sampling Rate**

Value	Name	Description
0	NO_AVERAGE	No averaging. ADC sample rate is maximum
1	OSR4	1-bit enhanced resolution by averaging. ADC sample rate divided by 4
2	OSR16	2-bit enhanced resolution by averaging. ADC sample rate divided by 16

This field is active if LOWRES is cleared in ADC_MR.

Note: FREERUN (see ADC_MR) must be set to 0 when digital averaging is used.

- **ASTE: Averaging on Single Trigger Event**

Value	Name	Description
0	MULTI_TRIG_AVERAGE	The average requests several trigger events
1	SINGLE_TRIG_AVERAGE	The average requests only one trigger event

- **TAG: TAG of the ADC_LDCR register**

0 = Sets CHNB to zero in ADC_LDCR.

1 = Appends the channel number to the conversion result in ADC_LDCR.

40.7.16 ADC Compare Window Register

Name: ADC_CWR

Address: 0x40038044

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in “ADC Write Protection Mode Register” .

- **LOWTHRES: Low Threshold**

Low threshold associated to compare settings of ADC_EMR.

If LOWRES is set in ADC_MR, only the 10 LSB of LOWTHRES must be programmed. The 2 LSBs are automatically discarded to match the value carried on ADC_CDR (8-bit).

- **HIGHTHRES: High Threshold**

High threshold associated to compare settings of ADC_EMR.

If LOWRES is set in ADC_MR, only the 10 LSB of HIGHTHRES must be programmed. The 2 LSBs are automatically discarded to match the value carried on ADC_CDR (8-bit).

40.7.17 ADC Channel Data Register

Name: ADC_CDRx [x=0..7]

Address: 0x40038050, 0x40038054, 0x40038058, 0x4003805C, 0x40038060, 0x40038064, 0x40038068, 0x4003806C

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	DATA			

7	6	5	4	3	2	1	0
DATA							

- **DATA: Converted Data**

The analog-to-digital conversion data is placed into this register at the end of a conversion and remains until a new conversion is completed. ADC_CDRx is only loaded if the corresponding analog channel is enabled.

40.7.18 ADC Analog Control Register

Name: ADC_ACR

Address: 0x40038094

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in "ADC Write Protection Mode Register" .

- **IRVCE: Internal Reference Voltage Change Enable**

0 (STUCK_AT_DEFAULT) = The internal reference voltage is stuck at the default value (see the Electrical Characteristics for further details).

1 (SELECTION) = The internal reference voltage is defined by field IRVS.

- **IRVS: Internal Reference Voltage Selection**

See the "Programmable Voltage Reference Selection Values" table in the Electrical Characteristics for further details.

- **FORCEREF: Force Internal Reference Voltage**

0 = The internal ADC voltage reference input is connected to the ADVREF line.

1 = The internal ADC voltage reference input is forced to VDDIO (ONREF must be cleared).

- **ONREF: Internal Voltage Reference ON**

0 = The programmable voltage reference is OFF. The user can either force the internal ADC voltage reference input on the ADVREF pin or set the FORCEREF bit to connect VDDIO to the internal ADC voltage reference input.

1 = The programmable voltage reference is ON and its output is connected both to the ADC voltage reference input and to the external ADVREF pin for decoupling (FORCEREF must be cleared).

40.7.19 ADC Write Protection Mode Register

Name: ADC_WPMR

Address: 0x400380E4

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 0x414443 (“ADC” in ASCII).

1 = Enables the write protection if WPKEY corresponds to 0x414443 (“ADC” in ASCII).

See Section 40.6.13 ”Register Write Protection” for the list of registers that 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.

40.7.20 ADC Write Protection Status Register

Name: ADC_WPSR

Address: 0x400380E8

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 ADC_WPSR register.

1 = A write protection violation has occurred since the last read of the ADC_WPSR register. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protection Violation Source**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

41. Advanced Encryption Standard (AES)

41.1 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,* as well as Galois/Counter Mode (GCM) as specified in the *NIST Special Publication 800-38D Recommendation*. It is compatible with all these modes via Peripheral DMA Controller channels, minimizing processor intervention for large buffer transfers.

The 128-bit/192-bit/256-bit key is stored in four/six/eight 32-bit write-only AES Key Word Registers (AES_KEYWR0-7).

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 PDC channels.

41.2 Embedded Characteristics

- Compliant with *FIPS Publication 197, Advanced Encryption Standard (AES).*
- 128-bit/192-bit/256-bit Cryptographic Key.
- 10/12/14 Clock Cycles Encryption/Decryption Inherent 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 and NIST Special Publication 800-38D:*
  - Electronic Codebook (ECB).
  - Cipher Block Chaining (CBC) including CBC-MAC.
  - Cipher Feedback (CFB).
  - Output Feedback (OFB).
  - Counter (CTR).
  - Galois/Counter Mode (GCM).
- 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 PDC Channel Capabilities Optimizes Data Transfers for all Operating Modes.
  - One Channel for the Receiver, One Channel for the Transmitter.
  - Next Buffer Support.

41.3 Product Dependencies

41.3.1 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.

41.3.2 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 41-1. Peripheral IDs

Instance	ID
AES	36

41.4 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 Key Registers (AES_KEYWRx).

The input to the encryption processes of the CBC, CFB, and OFB modes includes, in addition to the plaintext, a 128-bit data block called the initialization vector (IV), which must be set in 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. AES_IVRx are also used by the CTR mode to set the counter value.

41.4.1 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 AES_IDATAR0 and AES_IDATAR1 registers must be written with the following pattern:

- AES_IDATAR0 = 0xcadefeca
- AES_IDATAR1 = 0x67452301

41.4.2 Operating Modes

The AES supports the following modes of operation:

- ECB: Electronic Codebook.
- 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.
- GCM: Galois/Counter Mode.

The data preprocessing, data postprocessing and data chaining for the concerned modes are performed automatically. Refer to the *NIST Special Publication 800-38A and NIST Special Publication 800-38D* for more complete information.

Mode selection is done by configuring the OPMOD field in 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 AES_MR (Section 41.5.2 ”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 Mbyte of data. If the file to be processed is greater than 1 Mbyte, this file must be split into fragments of 1 Mbyte 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 x (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.

41.4.3 Double Input Buffer

AES_IDATARx can be double-buffered to reduce the runtime of large files.

This mode allows a new message block when the previous message block is being processed. This is only possible when DMA accesses are performed (SMOD = 2).

The DUALBUFF bit in AES_MR must be set to ‘1’ to access the double buffer.

41.4.4 Start Modes

The SMOD field in AES_MR allows selection of the encryption (or decryption) Start mode.

41.4.4.1 Manual Mode

The sequence of actions is as follows:

1. Write 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 AES_KEYWRx.
3. Write the initialization vector (or counter) in AES_IVRx.

Note: AES_IVRx concerns 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 (refer to Table 41-2).
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 AES_IER, the interrupt line of the AES is activated.
8. When software reads one of AES_ODATARx, the DATRDY bit is automatically cleared.

Table 41-2. Authorized Input Data Registers

Operating Mode	Input Data Registers to Write
ECB	All
CBC	All
OFB	All
128-bit CFB	All
64-bit CFB(1)	AES_IDATAR0 and AES_IDATAR1
32-bit CFB(2)	AES_IDATAR0
16-bit CFB(2)	AES_IDATAR0
8-bit CFB(2)	AES_IDATAR0
CTR	All
GCM	All

Notes: 1. In 64-bit CFB mode, writing to AES_IDATAR2 and AES_IDATAR3 is not allowed and may lead to errors in processing.

2. 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.

41.4.4.2 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 AES_CR.

41.4.4.3 PDC Mode

The Peripheral DMA Controller (PDC) can be used in association with the AES to perform an encryption/decryption of a buffer without any action by software during processing.

The field SMOD in AES_MR must be configured to 2.

The sequence order is as follows:

1. Write AES_MR with all required fields, including but not limited to SMOD and OPMOD.
2. Write the key in AES_KEYWRx.
3. Write the initialization vector (or counter) in AES_IVRx.

Note: AES_IVRx concerns all modes except ECB.

4. Set the Transmit Pointer Register (AES_TPR) to the address where the data buffer to encrypt/decrypt is stored and the Receive Pointer Register (AES_RPR) where it must be encrypted/decrypted.

Note: Transmit and receive buffers can be identical.

5. Set the Transmit and the Receive Counter Registers (AES_TCR and AES_RCR) to the same value. This value must be a multiple of the data transfer type size (refer to Table 41-3 "Data Transfer Type for the Different Operating Modes").

Note: The same requirements are necessary for the Next Pointer(s) and Counter(s) of the PDC (AES_TNPR, AES_RNPR, AES_TNCR, AES_RNCR).

6. If not already done, set the bit ENDRX (or RXBUFF if the next pointers and counters are used) in AES_IER, depending on whether an interrupt is required or not at the end of processing.
7. Enable the PDC in transmission and reception to start the processing (AES_PTCR).

When the processing completes, the ENDRX (or RXBUFF) flag in AES_ISR is raised. If an interrupt has been enabled by setting the corresponding bit in AES_IER, the interrupt line of the AES is activated.

When PDC is used, the data size to transfer (byte, half-word or word) depends on the AES mode of operations. This size is automatically configured by the AES.

Table 41-3. Data Transfer Type for the Different Operating Modes

Operating Mode	Data Transfer Type
ECB	Word
CBC	Word
OFB	Word
CFB 128-bit	Word
CFB 64-bit	Word
CFB 32-bit	Word
CFB 16-bit	Half-word
CFB 8-bit	Byte
CTR	Word
GCM	Word

41.4.5 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 PDC mode (refer to Table 41-4 "Last Output Data Mode Behavior versus Start Modes").

The Last Output Data (LOD) bit in 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 PDC mode.

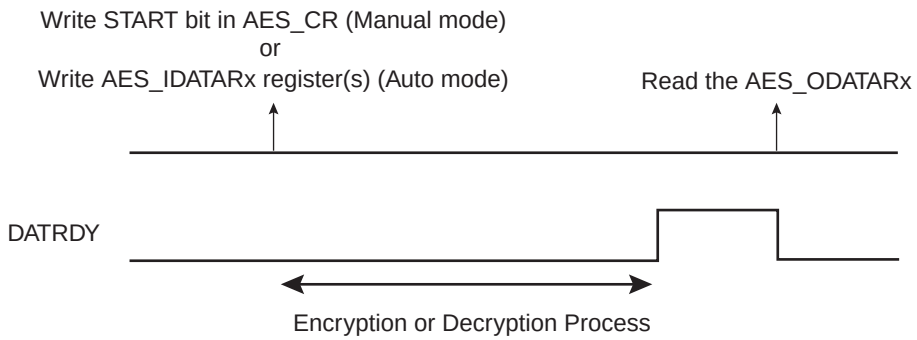
This data are only available in AES_ODATARx.

41.4.5.1 Manual and Auto Modes

If AES_MR.LOD = 0

The DATRDY flag is cleared when at least one AES_ODATARx is read (refer to Figure 41-1).

Figure 41-1. Manual and Auto Modes with AES_MR.LOD = 0



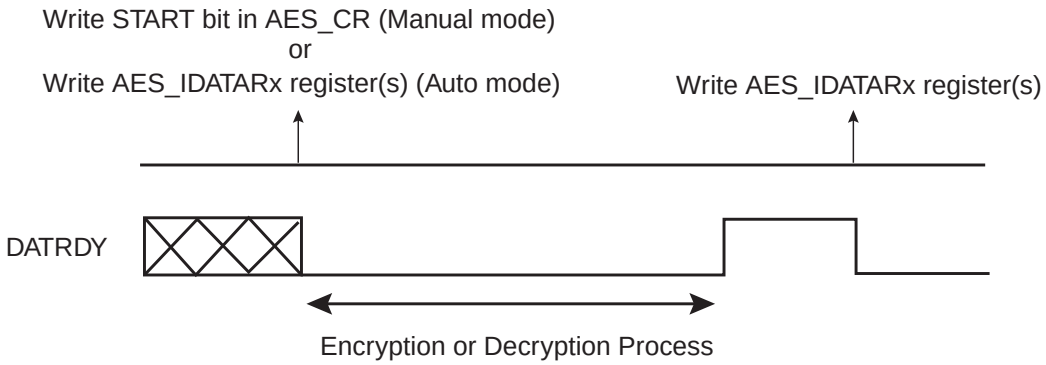
If the user does not want to read AES_ODATARx between each encryption/decryption, the DATRDY flag will not be cleared. If the DATRDY flag is not cleared, the user cannot know the end of the following encryptions/decryptions.

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 (refer to Figure 41-2). No additional AES_ODATAR reads are necessary between consecutive encryptions/decryptions.

Figure 41-2. Manual and Auto Modes with AES_MR.LOD = 1



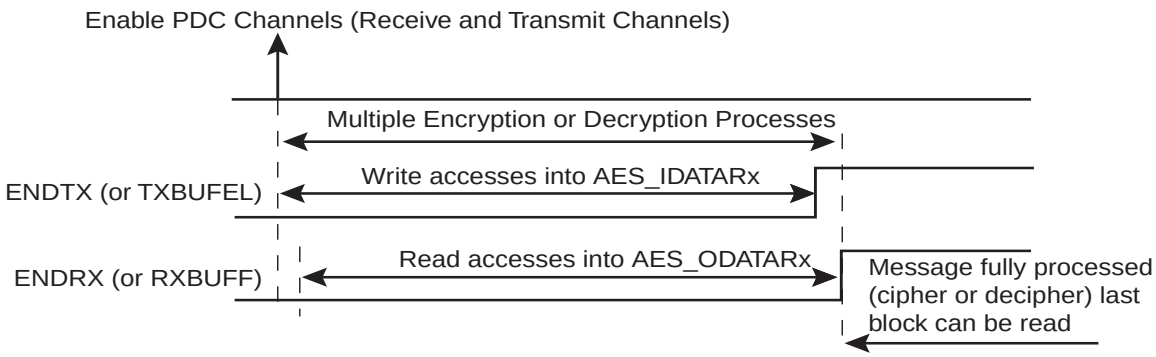
41.4.5.2 PDC Mode

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 when the ENDRX (or RXBUFF) flag is raised (refer to Figure 41-3).

Figure 41-3. PDC 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 ENDTX (or TXBUFE) flag to be raised, then for DATRDY to ensure that the encryption/decryption is completed (refer to Figure 41-4).

In this case, no receive buffers are required.

The output data are only available in AES_ODATARx.

Figure 41-4. PDC Transfer with AES_MR.LOD = 1

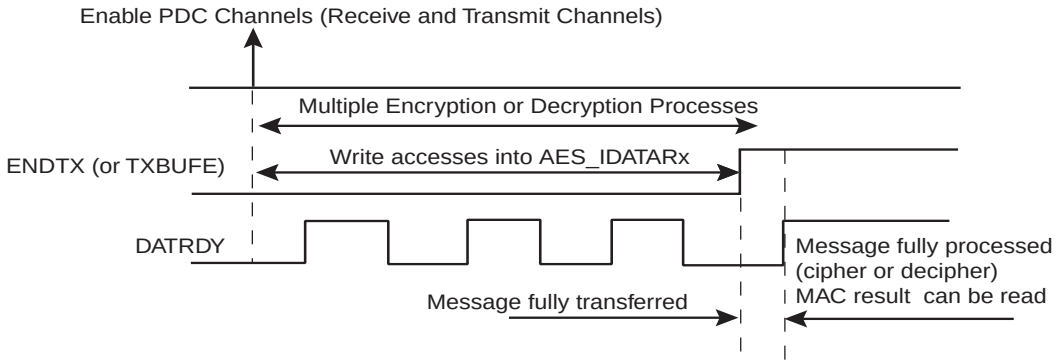


Table 41-4 summarizes the different cases.

Table 41-4. Last Output Data Mode Behavior versus Start Modes

Sequence	Manual and Auto Modes		PDC Mode	
	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 PDC
End of Encryption/Decryption Notification	DATRDY	DATRDY	ENDRX (or RXBUFF)	ENDTX (or TXBUFE) then DATRDY
Encrypted/Decrypted Data Result Location	In AES_ODATARx	In AES_ODATARx	At the address specified in AES_RPR	In AES_ODATARx

Notes: 1. Depending on the mode, there are other ways of clearing the DATRDY flag. Refer to Section 41.5.6 "AES Interrupt Status Register".

Warning: In PDC mode, reading the AES_ODATARx before the last data transfer may lead to unpredictable results.

41.4.6 Galois/Counter Mode (GCM)

41.4.6.1 Description

GCM comprises the AES engine in CTR mode along with a universal hash function (GHASH engine) that is defined over a binary Galois field to produce a message authentication tag (the AES CTR engine and the GHASH engine are depicted in Figure 41-5).

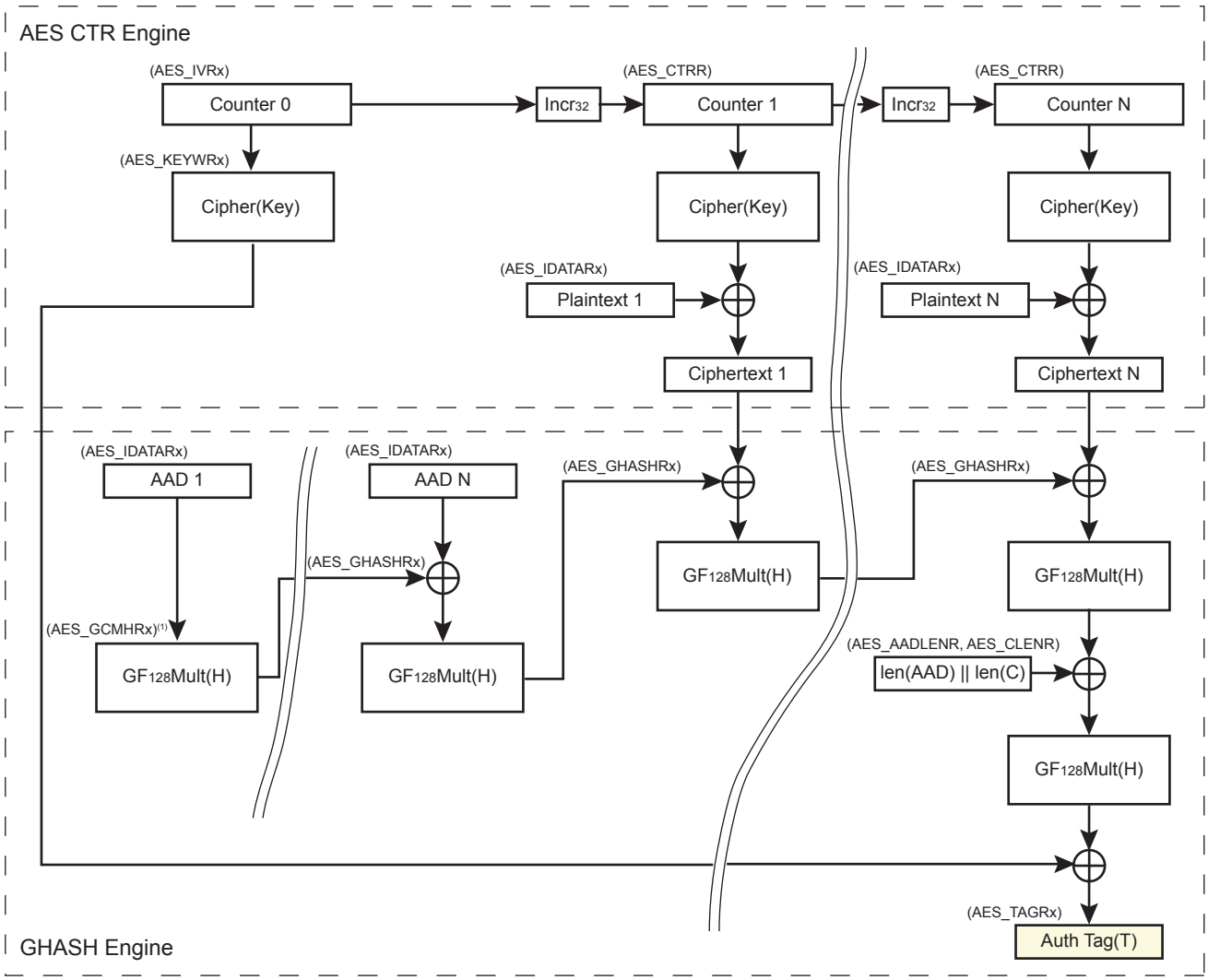
The GHASH engine processes data packets after the AES operation. GCM assures of the confidentiality of data through the AES Counter mode of operation for encryption. Authenticity of the confidential data is assured through the GHASH engine. GCM can also provide assurance of data that is not encrypted. Refer to the *NIST Special Publication 800-38D* for more complete information.

GCM can be used with or without the PDC master. Messages may be processed as a single complete packet of data or they may be broken into multiple packets of data over time.

GCM processing is computed on 128-bit input data fields. There is no support for unaligned data. The AES key length can be whatever length is supported by the AES module.

The recommended programming procedure when using PDC is described in Section 41.4.6.3.

Figure 41-5. GCM Block Diagram



Notes: 1. Optional

41.4.6.2 Key Writing and Automatic Hash Subkey Calculation

Whenever a new key (AES_KEYWRx) is written to the hardware, two automatic actions are processed:

- GCM Hash Subkey *H* generation—The GCM hash subkey (*H*) is automatically generated. The GCM hash subkey generation must be complete before doing any other action. The DATRDY bit of AES_ISR indicates when the subkey generation is complete (with interrupt if configured). The GCM hash subkey calculation is processed with the formula *H* = CIPHER (Key, <128 bits to zero>). The generated GCM *H* value is then available in AES_GCMHRx. If the application software requires a specific hash subkey, the automatically generated H value can be overwritten in AES_GCMHRx.
  AES_GCMHRx can be written after the end of the hash subkey generation (refer to AES_ISR.DATRDY) and prior to starting the input data feed.
- AES_GHASHRx Clear—AES_GHASHRx are automatically cleared. If a hash initial value is needed for the GHASH, it must be written to AES_GHASHRx:
  - After a write to AES_KEYWRx, if any
  - Before starting the input data feed

41.4.6.3 GCM Processing

GCM processing is made up of three phases:

1. Processing the Additional Authenticated Data (AAD), hash computation only.
2. Processing the Ciphertext (C), hash computation + ciphering/deciphering.
3. Generating the Tag using length of AAD, length of C and J_0 (refer to NIST documentation for details).

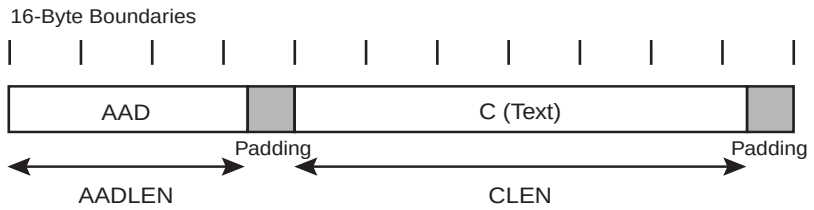
The Tag generation can be done either automatically, after the end of AAD/C processing if GTAGEN bit is set in AES_MR or done manually using the GHASH field in AES_GHASHRx (Refer to Section 41.4.6.3.1 and Section 41.4.6.3.4 for details).

41.4.6.3.1 Processing a Complete Message with Tag Generation

Use this procedure only if J_0 four LSB bytes ≠ 0xFFFFFFFF.

Note: If J_0 four LSB bytes = 0xFFFFFFFF or if the value is unknown, use the procedure described in "Processing a Complete Message without Tag Generation" followed by the procedure in "Manual GCM Tag Generation" .

Figure 41-6. Full Message Alignment



To process a complete message with Tag generation, the sequence is as follows:

1. In AES_MR, set OPMOD to GCM and GTAGEN to '1'.
2. Set KEYW in AES_KEYWRx and wait until AES_ISR.DATRDY is set (GCM hash subkey generation complete); use interrupt if needed. Refer to Section 41.4.6.2 "Key Writing and Automatic Hash Subkey Calculation".
3. Calculate the J_0 value as described in NIST documentation $J_0 = IV \,||\, 0^{31} \,||\, 1$ when len(IV)=96 and J_0=GHASH$_H$($IV \,||\, 0^{s+64} \,||\, [\text{len}(IV)]_{64}$) if len($IV$) ≠ 96. Refer to Section 41.4.6.3.5 "Processing a Message with only AAD (GHASHH)" for J_0 generation.
4. Set IV in AES_IVRx with inc32 (J_0) (J_0 + 1 on 32 bits).
5. Set AADLEN field in AES_AADLENR and CLEN field in AES_CLENR.
6. Fill the IDATA field of AES_IDATARx with the message to process according to the SMOD configuration used. If Manual Mode or Auto Mode is used, the DATRDY bit indicates when the data have been processed (however, no output data are generated when processing AAD).
7. Wait for TAGRDY to be set (use interrupt if needed), then read the TAG field of AES_TAGRx to obtain the authentication tag of the message.

41.4.6.3.2 Processing a Complete Message without Tag Generation

Processing a message without generating the Tag can be used to customize the Tag generation, or to process a fragmented message. To manually generate the GCM Tag, refer to Section 41.4.6.3.4.

To process a complete message without Tag generation, the sequence is as follows:

1. In AES_MR, set OPMOD to GCM and GTAGEN to '0'.
2. Set KEYW in AES_KEYWRx and wait until DATRDY bit of AES_ISR is set (GCM hash subkey generation complete); use interrupt if needed. After the GCM hash subkey generation is complete the GCM hash subkey can be read or overwritten with specific value in AES_GCMHRx. Refer to Section 41.4.6.2 "Key Writing and Automatic Hash Subkey Calculation".

3. Calculate the J_0 value as described in NIST documentation $J_0 = IV \| 0^{31} \| 1$ when len(IV)=96 and J_0=GHASH$_H$($IV \| 0^{s+64} \| [\text{len}(IV)]_{64}$) if len($IV$) ≠ 96. Refer to Section 41.4.6.3.5 "Processing a Message with only AAD (GHASHH)" for J_0 generation example when len(IV) ≠ 96.
4. Set IV in AES_IVRx with inc32 (J_0) (J_0 + 1 on 32 bits).
5. Set AADLEN field in AES_AADLENR and CLEN field in AES_CLENR.
6. Fill the IDATA field of AES_IDATARx with the message to process according to the SMOD configuration used. If Manual Mode or Auto Mode is used, the DATRDY bit indicates when the data have been processed (however, no output data are generated when processing AAD).
7. Make sure the last output data have been read if CLEN ≠ 0 (or wait for DATRDY), then read the GHASH field of AES_GHASHRx to obtain the hash value after the last processed data.

41.4.6.3.3 Processing a Fragmented Message without Tag Generation

If needed, a message can be processed by fragments, in such case automatic GCM Tag generation is not supported.

To process a message by fragments, the sequence is as follows:

- First fragment:

1. In AES_MR, set OPMOD to GCM and GTAGEN to '0'.
2. Set KEYW in AES_KEYWRx and wait for DATRDY bit of AES_ISR to be set (GCM hash subkey generation complete); use interrupt if needed. After the GCM hash subkey generation is complete the GCM hash subkey can be read or overwritten with specific value in AES_GCMHRx. Refer to Section 41.4.6.2 "Key Writing and Automatic Hash Subkey Calculation".
3. Calculate the J_0 value as described in NIST documentation $J_0 = IV \| 0^{31} \| 1$ when len(IV)=96 and J_0=GHASH$_H$($IV \| 0^{s+64} \| [\text{len}(IV)]_{64}$) if len($IV$) ≠ 96. Refer to Section 41.4.6.3.5 "Processing a Message with only AAD (GHASHH)" for J_0 generation example when len(IV) ≠ 96.
4. Set IV in AES_IVRx with inc32 (J_0) (J_0 + 1 on 32 bits).
5. Set AADLEN field in AES_AADLENR and CLEN field in AES_CLENR according to the length of the first fragment, or set the fields with the full message length, both configurations work.
6. Fill the IDATA field of AES_IDATARx with the first fragment of the message to process (aligned on 16-byte boundary) according to the SMOD configuration used. If Manual Mode or Auto Mode is used the DATRDY bit indicates when the data have been processed (however, no output data are generated when processing AAD).
7. Make sure the last output data have been read if the fragment ends in C phase (or wait for DATRDY if the fragment ends in AAD phase), then read the GHASH field of AES_GHASHRx to obtain the value of the hash after the last processed data and finally read the CTR field of AES_CTR to obtain the value of the CTR encryption counter (not needed when the fragment ends in AAD phase).

- Next fragment (or last fragment):

1. In AES_MR, set OPMOD to GCM and GTAGEN to '0'.
2. Set KEYW in AES_KEYWRx and wait until DATRDY bit of AES_ISR is set (GCM hash subkey generation complete); use interrupt if needed. After the GCM hash subkey generation is complete the GCM hash subkey can be read or overwritten with specific value in AES_GCMHRx. Refer to Section 41.4.6.2 "Key Writing and Automatic Hash Subkey Calculation".
3. Set IV in AES_IVRx with:
   - If the first block of the fragment is a block of Additional Authenticated data, set IV in AES_IVRx with the J_0 initial value.
   - If the first block of the fragment is a block of Plaintext data, set IV in AES_IVRx with a value constructed as follows: '$\text{LSB}_{96}(J_0) \| \text{CTR}$' value, (96 bit LSB of J_0 concatenated with saved CTR value from previous fragment).
4. Set AADLEN field in AES_AADLENR and CLEN field in AES_CLENR according to the length of the current fragment, or set the fields with the remaining message length, both configurations work.
5. Fill the GHASH field of AES_GHASHRx with the value stored after the previous fragment.

6. Fill the IDATA field of AES_IDATARx with the current fragment of the message to process (aligned on 16 byte boundary) according to the SMOD configuration used. If Manual Mode or Auto Mode is used, the DATRDY bit indicates when the data have been processed (however, no output data are generated when processing AAD).
7. Make sure the last output data have been read if the fragment ends in C phase (or wait for DATRDY if the fragment ends in AAD phase), then read the GHASH field of AES_GHASHRx to obtain the value of the hash after the last processed data and finally read the CTR field of AES_CTR to obtain the value of the CTR encryption counter (not needed when the fragment ends in AAD phase).

Note: Step 1 and 2 are required only if the value of the concerned registers has been modified.

Once the last fragment has been processed, the GHASH value will allow manual generation of the GCM tag. Refer to Section 41.4.6.3.4.

41.4.6.3.4 Manual GCM Tag Generation

This section describes the last steps of the GCM Tag generation.

The Manual GCM Tag Generation is used to complete the GCM Tag Generation when the message has been processed without Tag Generation.

Note: The Message Processing without Tag Generation must be finished before processing the Manual GCM Tag Generation.

To generate a GCM Tag manually, the sequence is as follows:

Processing $S = GHASH_H (AAD \,||\, 0^v \,||\, C \,||\, 0^u \,||\, [\text{len}(AAD)]_{64} \,||\, [\text{len}(C)]_{64})$:

1. In AES_MR, set OPMOD to GCM and GTAGEN to ‘0’.
2. Set KEYW in AES_KEYWRx and wait for DATRDY bit of AES_ISR to be set (GCM hash subkey generation complete); use interrupt if needed. After the GCM hash subkey generation is complete the GCM hash subkey can be read or overwritten with specific value in AES_GCMHRx. Refer to Section 41.4.6.2 ”Key Writing and Automatic Hash Subkey Calculation”.
3. Set AADLEN field to 0x10 (16 bytes) in AES_AADLENR and CLEN field to ‘0’ in AES_CLENR. This will allow running a single $GHASH_H$ on a 16-byte input data (refer to Figure 41-7).
4. Fill the GHASH field of AES_GHASHRx with the state of the GHASH field stored at the end of the message processing.
5. Fill the IDATA field of AES_IDATARx according to the SMOD configuration used with ‘$\text{len}(AAD)_{64} \,||\, \text{len}(C)_{64}$’ value as described in the NIST documentation and wait for DATRDY to be set; use interrupt if needed.
6. Read the GHASH field of AES_GHASHRx to obtain the current value of the hash.

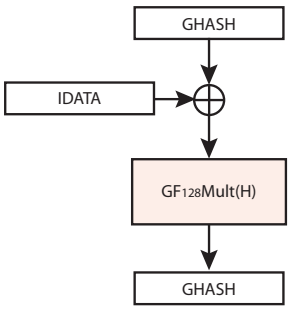
Processing $T = GCTR_K(J_0, S)$:

7. In AES_MR, set OPMOD to CTR.
8. Set the IV field in AES_IVRx with ‘J_0’ value.
9. Fill the IDATA field of AES_IDATARx with the GHASH value read at step 6 and wait for DATRDY to be set (use interrupt if needed).
10. Read the ODATA field of AES_ODATARx to obtain the GCM Tag value.

Note: Step 4 is optional if the GHASH field is to be filled with value ‘0’ (0 length packet for instance).

41.4.6.3.5 Processing a Message with only AAD (GHASHH)

Figure 41-7. Single GHASHH Block Diagram (AADLEN ≤ 0x10 and CLEN = 0)



It is possible to process a message with only AAD setting the CLEN field to '0' in AES_CLENR, this can be used for J_0 generation when len(IV) ≠ 96 for instance.

Example: Processing J_0 when len(IV) ≠ 96.

To process $J_0 = GHASH_H(IV \,||\, 0^{s+64} \,||\, [len(IV)]_{64})$, the sequence is as follows:

1. In AES_MR, set OPMOD to GCM and GTAGEN to '0'.
2. Set KEYW in AES_KEYWRx and wait until DATRDY bit of AES_ISR is set (GCM hash subkey generation complete); use interrupt if needed. After the GCM hash subkey generation is complete the GCM hash subkey can be read or overwritten with specific value in AES_GCMHRx. Refer to Section 41.4.6.2 "Key Writing and Automatic Hash Subkey Calculation".
3. Set AADLEN field with '$len(IV \,||\, 0^{s+64} \,||\, [len(IV)]_{64})$' in AES_AADLENR and CLEN field to '0' in AES_CLENR. This will allow running a $GHASH_H$ only.
4. Fill the IDATA field of AES_IDATARx with the message to process ($IV \,||\, 0^{s+64} \,||\, [len(IV)]_{64}$) according to the SMOD configuration used. If Manual Mode or Auto Mode is used, the DATRDY bit indicates when a $GHASH_H$ step is over (use interrupt if needed).
5. Read the GHASH field of AES_GHASHRx to obtain the J_0 value.

Note: The GHASH value can be overwritten at any time by writing the GHASH field value of AES_GHASHRx, used to perform a $GHASH_H$ with an initial value for GHASH (write GHASH field between step 3 and step 4 in this case).

41.4.6.3.6 Processing a Single GF128 Multiplication

The AES can also be used to process a single multiplication in the Galois field on 128 bits (GF_{128}) using a single $GHASH_H$ with custom H value (refer to Figure 41-7).

To run a GF_{128} multiplication (A x B), the sequence is as follows:

1. In AES_MR, set OPMOD to GCM and GTAGEN to '0'.
2. Set AADLEN field with 0x10 (16 bytes) in AES_AADLENR and CLEN field to '0' in AES_CLENR. This will allow running a single $GHASH_H$.
3. Fill the H field of AES_GCMHRx with B value.
4. Fill the IDATA field of AES_IDATARx with the A value according to the SMOD configuration used. If Manual Mode or Auto Mode is used, the DATRDY bit indicates when a $GHASH_H$ computation is over (use interrupt if needed).
5. Read the GHASH field of AES_GHASHRx to obtain the result.

Note: The GHASH field of AES_GHASHRx can be initialized with a value C between step 3 and step 4 to run a ((A XOR C) x B) GF_{128} multiplication.

41.4.7 Security Features

41.4.7.1 Unspecified Register Access Detection

When an unspecified register access occurs, the URAD flag in 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 AES_CR.

41.5 Advanced Encryption Standard (AES) User Interface

Table 41-5. 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	0x0000001E
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	–

Table 41-5. Register Mapping (Continued)

Offset	Register	Name	Access	Reset
0x68	Initialization Vector Register 2	AES_IVR2	Write-only	–
0x6C	Initialization Vector Register 3	AES_IVR3	Write-only	–
0x70	Additional Authenticated Data Length Register	AES_AADLENR	Read/Write	–
0x74	Plaintext/Ciphertext Length Register	AES_CLENR	Read/Write	–
0x78	GCM Intermediate Hash Word Register 0	AES_GHASHR0	Read/Write	–
0x7C	GCM Intermediate Hash Word Register 1	AES_GHASHR1	Read/Write	–
0x80	GCM Intermediate Hash Word Register 2	AES_GHASHR2	Read/Write	–
0x84	GCM Intermediate Hash Word Register 3	AES_GHASHR3	Read/Write	–
0x88	GCM Authentication Tag Word Register 0	AES_TAGR0	Read-only	–
0x8C	GCM Authentication Tag Word Register 1	AES_TAGR1	Read-only	–
0x90	GCM Authentication Tag Word Register 2	AES_TAGR2	Read-only	–
0x94	GCM Authentication Tag Word Register 3	AES_TAGR3	Read-only	–
0x98	GCM Encryption Counter Value Register	AES_CTRR	Read-only	–
0x9C	GCM H Word Register 0	AES_GCMHR0	Read/Write	–
0xA0	GCM H Word Register 1	AES_GCMHR1	Read/Write	–
0xA4	GCM H Word Register 2	AES_GCMHR2	Read/Write	–
0xA8	GCM H Word Register 3	AES_GCMHR3	Read/Write	–
0xAC	Reserved	–	–	–
0xB0 - 0xFC	Reserved	–	–	–
0x100 - 0x124	Reserved for the PDC	–	–	–

41.5.1 AES Control Register

Name: AES_CR

Address: 0x40000000

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	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.

41.5.2 AES Mode Register

Name: AES_MR

Address: 0x40000004

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	–	GTAGEN	CIPHER

- **CIPHER: Processing Mode**

0: Decrypts data.

1: Encrypts data.

- **GTAGEN: GCM Automatic Tag Generation Enable**

0: Automatic GCM Tag generation disabled.

1: Automatic GCM Tag generation enabled.

- **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**

$$\text{Processing Time} = N \times (PROCDLY + 1)$$

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 (PDC)

If a PDC transfer is used, configure SMOD to 2. Refer to Section 41.4.4.3 "PDC 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

- **OPMOD: Operating Mode**

Value	Name	Description
0	ECB	ECB: Electronic Codebook 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)
5	GCM	GCM: Galois/Counter mode

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 Receive Pointer Register (AES_RPR) for PDC 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 (refer to Section 41.4.5 ”Last Output Data Mode”).

Warning: In PDC 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

- **CKEY: Key**

Value	Name	Description
0xE	PASSWD	This field must be written with 0xE the first time AES_MR is programmed. For subsequent programming of AES_MR, any value can be written, including that of 0xE. Always reads as 0.

41.5.3 AES Interrupt Enable Register

Name: AES_IER

Address: 0x40000010

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	TAGRDY

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	URAD

7	6	5	4	3	2	1	0
–	–	–	TXBUFE	RXBUFF	ENDTX	ENDRX	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**
- **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**
- **URAD: Unspecified Register Access Detection Interrupt Enable**
- **TAGRDY: GCM Tag Ready Interrupt Enable**

41.5.4 AES Interrupt Disable Register

Name: AES_IDR

Address: 0x40000014

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	TAGRDY

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	URAD

7	6	5	4	3	2	1	0
–	–	–	TXBUFE	RXBUFF	ENDTX	ENDRX	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**
- **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**
- **URAD: Unspecified Register Access Detection Interrupt Disable**
- **TAGRDY: GCM Tag Ready Interrupt Disable**

41.5.5 AES Interrupt Mask Register

Name: AES_IMR

Address: 0x40000018

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	TAGRDY

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	URAD

7	6	5	4	3	2	1	0
–	–	–	TXBUFE	RXBUFF	ENDTX	ENDRX	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**
- **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**
- **URAD: Unspecified Register Access Detection Interrupt Mask**
- **TAGRDY: GCM Tag Ready Interrupt Mask**

41.5.6 AES Interrupt Status Register

Name: AES_ISR

Address: 0x4000001C

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	TAGRDY
15	14	13	12	11	10	9	8
URAT				–	–	–	URAD
7	6	5	4	3	2	1	0
–	–	–	TXBUFE	RXBUFF	ENDTX	ENDRX	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.

- **ENDRX: End of RX Buffer (cleared by writing AES_RCR or AES_RNCR)**

0: The Receive Counter Register has not reached 0 since the last write in AES_RCR or AES_RNCR.

1: The Receive Counter Register has reached 0 since the last write in AES_RCR or AES_RNCR.

Note: This flag must be used only in PDC mode with AES_MR.LOD bit cleared.

- **ENDTX: End of TX Buffer (cleared by writing AES_TCR or AES_TNCR)**

0: The Transmit Counter Register has not reached 0 since the last write in AES_TCR or AES_TNCR.

1: The Transmit Counter Register has reached 0 since the last write in AES_TCR or AES_TNCR.

Note: This flag must be used only in PDC mode with AES_MR.LOD bit set.

- **RXBUFF: RX Buffer Full (cleared by writing AES_RCR or AES_RNCR)**

0: AES_RCR or AES_RNCR has a value other than 0.

1: Both AES_RCR and AES_RNCR have a value of 0.

Note: This flag must be used only in PDC mode with AES_MR.LOD bit cleared.

- **TXBUFE: TX Buffer Empty (cleared by writing AES_TCR or AES_TNCR)**

0: AES_TCR or AES_TNCR has a value other than 0.

1: Both AES_TCR and AES_TNCR have a value of 0.

Note: This flag must be used only in PDC mode with AES_MR.LOD bit set.

- **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 = 2 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.

- **TAGRDY: GCM Tag Ready**

0: GCM Tag is not valid.

1: GCM Tag generation is complete (cleared by reading GCM Tag, starting another processing or when writing a new key).

41.5.7 AES Key Word Register x

Name: AES_KEYWRx [x=0..7]

Address: 0x40000020

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.

Whenever a new key (AES_KEYWRx) is written to the hardware, two automatic actions are processed:

- GCM hash subkey generation
- AES_GHASHRx Clear

Refer to Section 41.4.6.2 ”Key Writing and Automatic Hash Subkey Calculation” for details.

These registers are write-only to prevent the key from being read by another application.

41.5.8 AES Input Data Register x

Name: AES_IDATARx [x=0..3]

Address: 0x40000040

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.

41.5.9 AES Output Data Register x

Name: AES_ODATARx [x=0..3]

Address: 0x40000050

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.

41.5.10 AES Initialization Vector Register x

Name: AES_IVRx [x=0..3]

Address: 0x40000060

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.

41.5.11 AES Additional Authenticated Data Length Register

Name: AES_AADLENR

Address: 0x40000070

Access: Read/Write

31	30	29	28	27	26	25	24
AADLEN							

23	22	21	20	19	18	17	16
AADLEN							

15	14	13	12	11	10	9	8
AADLEN							

7	6	5	4	3	2	1	0
AADLEN							

- **AADLEN: Additional Authenticated Data Length**

Length in bytes of the Additional Authenticated Data (AAD) that is to be processed.

Note: The maximum byte length of the AAD portion of a message is limited to the 32-bit counter length.

41.5.12 AES Plaintext/Ciphertext Length Register

Name: AES_CLENR

Address: 0x40000074

Access: Read/Write

31	30	29	28	27	26	25	24
CLEN							

23	22	21	20	19	18	17	16
CLEN							

15	14	13	12	11	10	9	8
CLEN							

7	6	5	4	3	2	1	0
CLEN							

- **CLEN: Plaintext/Ciphertext Length**

Length in bytes of the plaintext/ciphertext (C) data that is to be processed.

Note: The maximum byte length of the C portion of a message is limited to the 32-bit counter length.

41.5.13 AES GCM Intermediate Hash Word Register x

Name: AES_GHASHRx [x=0..3]

Address: 0x40000078

Access: Read/Write

31	30	29	28	27	26	25	24
GHASH							

23	22	21	20	19	18	17	16
GHASH							

15	14	13	12	11	10	9	8
GHASH							

7	6	5	4	3	2	1	0
GHASH							

- **GHASH: Intermediate GCM Hash Word x**

The four 32-bit Intermediate Hash Word registers expose the intermediate GHASH value. May be read to save the current GHASH value so processing can later be resumed, presumably on a later message fragment. Whenever a new key (AES_KEYWRx) is written to the hardware two automatic actions are processed:

- GCM hash subkey generation
- AES_GHASHRx Clear

Refer to Section 41.4.6.2 ”Key Writing and Automatic Hash Subkey Calculation” for details.

If an application software-specific hash initial value is needed for the GHASH, it must be written to AES_GHASHRx:

- After a write to AES_KEYWRx, if any
- before starting the input data feed

41.5.14 AES GCM Authentication Tag Word Register x

Name: AES_TAGRx [x=0..3]

Address: 0x40000088

Access: Read-only

31	30	29	28	27	26	25	24
TAG							

23	22	21	20	19	18	17	16
TAG							

15	14	13	12	11	10	9	8
TAG							

7	6	5	4	3	2	1	0
TAG							

- **TAG: GCM Authentication Tag x**

The four 32-bit Tag registers contain the final 128-bit GCM Authentication tag (T) when GCM processing is complete. TAG0 corresponds to the first word, TAG3 to the last word.

41.5.15 AES GCM Encryption Counter Value Register

Name: AES_CTRR

Address: 0x40000098

Access: Read-only

31	30	29	28	27	26	25	24
CTR							

23	22	21	20	19	18	17	16
CTR							

15	14	13	12	11	10	9	8
CTR							

7	6	5	4	3	2	1	0
CTR							

- **CTR: GCM Encryption Counter**

Reports the current value of the 32-bit GCM counter.

41.5.16 AES GCM H Word Register x

Name: AES_GCMHRx [x=0..3]

Address: 0x4000009C

Access: Read/Write

31	30	29	28	27	26	25	24
H							

23	22	21	20	19	18	17	16
H							

15	14	13	12	11	10	9	8
H							

7	6	5	4	3	2	1	0
H							

- **H: GCM H Word x**

The four 32-bit H Word registers contain the 128-bit GCM hash subkey H value.

Whenever a new key (AES_KEYWRx) is written to the hardware, two automatic actions are processed:

- GCM hash subkey *H* generation
- AES_GHASHRx Clear

If the application software requires a specific hash subkey, the automatically-generated H value can be overwritten in the AES_GCMHRx. Refer to Section 41.4.6.2 ”Key Writing and Automatic Hash Subkey Calculation” for details.

Generating a GCM hash subkey *H* by a write in AES_GCMHRx enables to:

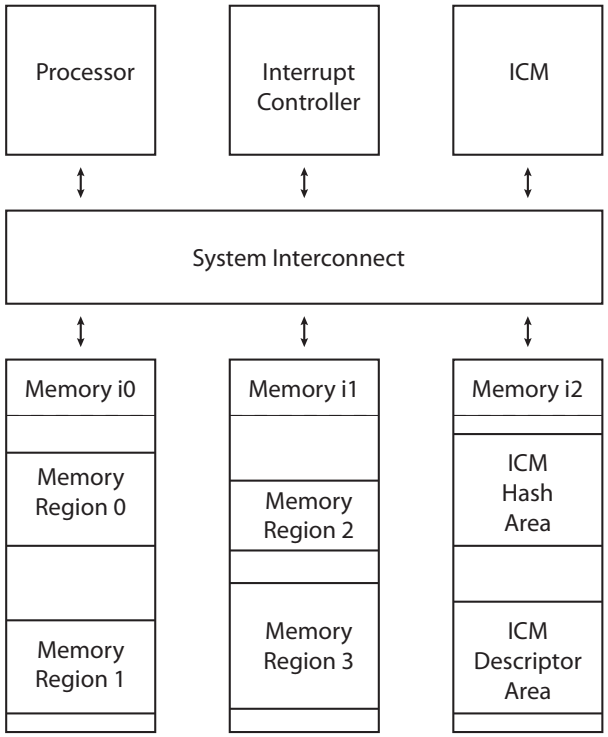
- select the GCM hash subkey *H* for GHASH operations.
- select one operand to process a single GF128 multiply.

42. Integrity Check Monitor (ICM)

42.1 Description

The Integrity Check Monitor (ICM) is a DMA controller that performs hash calculation over multiple memory regions through the use of transfer descriptors located in memory (ICM Descriptor Area). The Hash function is based on the Secure Hash Algorithm (SHA). The ICM controller integrates two modes of operation. The first one is used to hash a list of memory regions and save the digests to memory (ICM Hash Area). The second mode is an active monitoring of the memory. In that mode, the hash function is evaluated and compared to the digest located at a predefined memory address (ICM Hash Area). If a mismatch occurs, an interrupt is raised. See Figure 42-1 for an example of four-region monitoring. Hash and Descriptor areas are located in Memory instance i2, and the four regions are split in memory instances i0 and i1.

Figure 42-1. Four-region Monitoring Example



The ICM SHA engine is compliant with the American *FIPS (Federal Information Processing Standard) Publication 180-2* specification.

The following terms are concise definitions of the ICM concepts used throughout this document:

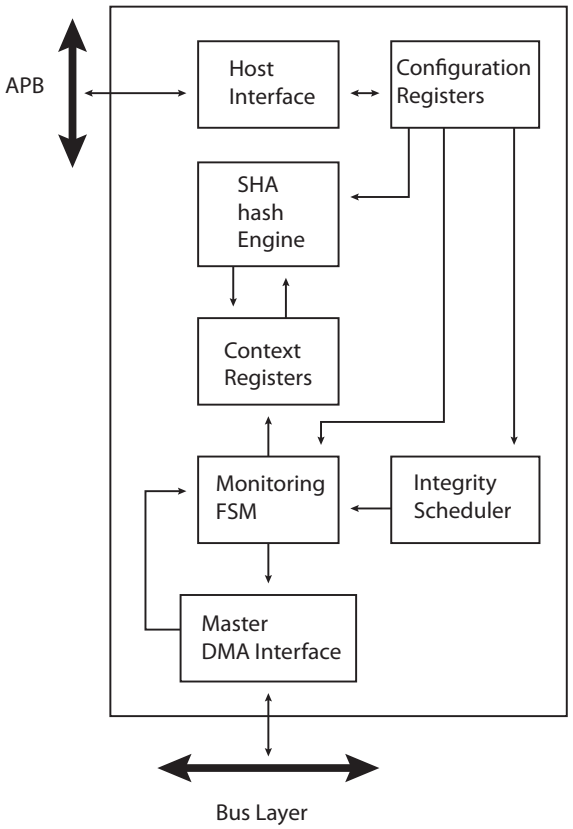
- Region: A partition of instruction or data memory space.
- Region Descriptor: A data structure stored in memory, defining region attributes.
- Region Attributes: Region start address, region size, region SHA engine processing mode, Write Back or Compare function mode.
- Context Registers: A set of ICM non-memory-mapped, internal registers which are automatically loaded, containing the attributes of the region being processed.
- Main List: A list of region descriptors. Each element associates the start address of a region with a set of attributes.
- Secondary List: A linked list defined on a per region basis that describes the memory layout of the region (when the region is non-contiguous).
- Hash Area: Predefined memory space where the region hash results (digest) are stored.

42.2 Embedded Characteristics

- DMA AHB master interface.
- Supports monitoring of up to 4 Non-Contiguous Memory Regions.
- Supports block gathering through the use of linked list.
- Supports Secure Hash Algorithm (SHA1, SHA224, SHA256).
- Compliant with *FIPS Publication 180-2.*
- Configurable Processing Period:
  - When SHA1 algorithm is processed, the runtime period is either 85 or 209 clock cycles.
  - When SHA256 or SHA224 algorithm is processed, the runtime period is either 72 or 194 clock cycles.
- Programmable Bus burden.

42.3 Block Diagram

Figure 42-2. Integrity Check Monitor Block Diagram



42.4 Product Dependencies

42.4.1 Power Management

The peripheral clock is not continuously provided to the ICM. The programmer must first enable the ICM clock in the Power Management Controller (PMC) before using the ICM.

42.4.2 Interrupt Sources

The ICM interface has an interrupt line connected to the Interrupt Controller. Handling the ICM interrupt requires programming the interrupt controller before configuring the ICM.

Table 42-1. Peripherals IDs

Instance	ID
ICM	34

42.5 Functional Description

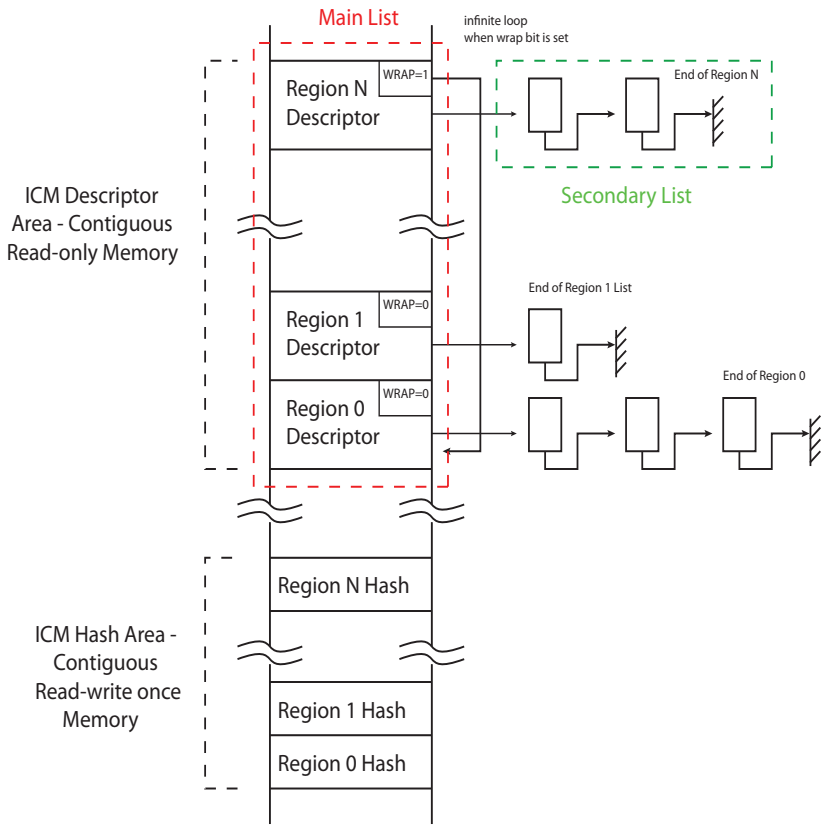
42.5.1 Overview

The Integrity Check Monitor (ICM) is a DMA controller that performs SHA-based memory hashing over memory regions. As shown in Figure 42-2, it integrates a DMA interface, a Monitoring Finite State Machine (FSM), an integrity scheduler, a set of context registers, a SHA engine, an interface for configuration and status registers.

The ICM integrates a Secure Hash Algorithm Engine (SHA). This engine requires a message padded according to FIPS180-2 specification when used as a SHA calculation unit only. Otherwise, if the ICM is used as integrated check for memory content, the padding is not mandatory. The SHA module produces an N-bit message digest each time a block is read and a processing period ends. N is 160 for SHA1, 224 for SHA224, 256 for SHA256.

When the ICM module is enabled, it sequentially retrieves a circular list of region descriptors from the memory (Main List described in Figure 42-3). Up to four regions may be monitored. Each region descriptor is composed of four words indicating the layout of the memory region (see Figure 42-4). It also contains the hashing engine configuration on a per region basis. As soon as the descriptor is loaded from the memory and context registers are updated with the data structure, the hashing operation starts. A programmable number of blocks (see TRSIZE field of the ICM_RCTRL structure member) is transferred from the memory to the SHA engine. When the desired number of blocks have been transferred, the digest is whether moved to memory (Write Back function) or compared with a digest reference located in the system memory (Compare function). If a digest mismatch occurs, an interrupt is triggered if unmasked. The ICM module passes through the region descriptor list until the end of the list marked by an End of List bit set to one. To continuously monitor the list of regions, the WRAP bit must be set to one in the last data structure.

Figure 42-3. ICM Region Descriptor and Hash Areas



Each region descriptor supports gathering of data through the use of the Secondary List. Unlike the Main List, the Secondary List cannot modify the configuration attributes of the region. When the end of the Secondary List has been encountered, the ICM returns to the Main List. Memory integrity monitoring can be considered as a background service and the mandatory bandwidth shall be very limited. In order to limit the ICM memory bandwidth, use the BBC field of the ICM_CFG register to control ICM memory load.

Figure 42-4. Region Descriptor

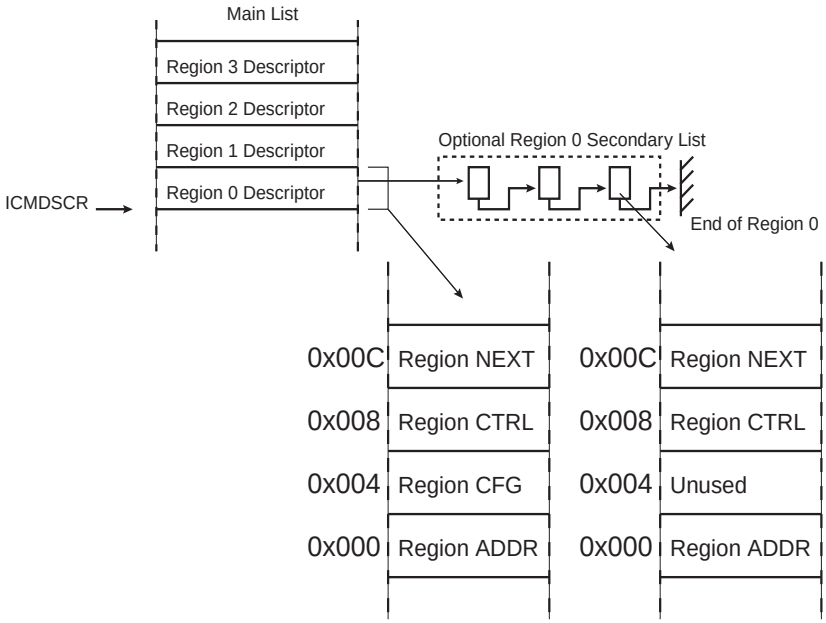
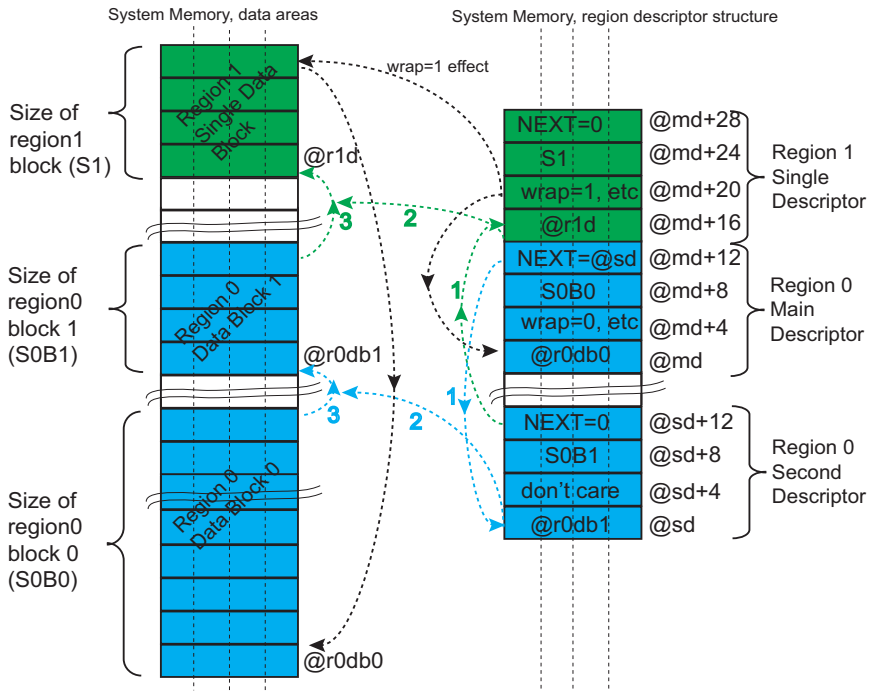


Figure 42-5 shows an example of the mandatory ICM settings to monitor three memory data blocks of the system memory (defined as two regions) with one region being not contiguous (two separate areas) and one contiguous memory area. For each said region, the SHA algorithm may be independently selected (different for each region). The wrap allows continuous monitoring.

Figure 42-5. Example: Monitoring of 3 Memory Data Blocks (Defined as 2 Regions)



42.5.2 ICM Region Descriptor Structure

The ICM Region Descriptor Area is a contiguous area of system memory that the controller and the processor can access. When the ICM controller is activated, the controller performs a descriptor fetch operation at *(ICM_DSCR) address. If the Main List contains more than one descriptor (i.e., more than one region is to be monitored), the fetch address is *(ICM_DSCR) + (RID<<4) where RID is the region identifier.

Table 42-2. Region Descriptor Structure (Main List)

Offset	Structure Member	Name
ICM_DSCR+0x000+RID*(0x10)	ICM Region Start Address	ICM_RADDR
ICM_DSCR+0x004+RID*(0x10)	ICM Region Configuration	ICM_RCFG
ICM_DSCR+0x008+RID*(0x10)	ICM Region Control	ICM_RCTRL
ICM_DSCR+0x00C+RID*(0x10)	ICM Region Next Address	ICM_RNEXT

42.5.2.1 ICM Region Start Address Structure Member

Name: ICM_RADDR

Address: ICM_DSCR+0x000+RID*(0x10)

Access: Read/Write

31	30	29	28	27	26	25	24
RADDR							

23	22	21	20	19	18	17	16
RADDR							

15	14	13	12	11	10	9	8
RADDR							

7	6	5	4	3	2	1	0
RADDR							

- **RADDR: Region Start Address**

This field indicates the first byte address of the region.

42.5.2.2 ICM Region Configuration Structure Member

Name: ICM_RCFG

Address: ICM_DSCR+0x004+RID*(0x10)

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
–	ALGO			–	PROCDLY	SUIEN	ECIEN

7	6	5	4	3	2	1	0
WCIEN	BEIEN	DMIEN	RHIEN	–	EOM	WRAP	CDWBN

- **CDWBN: Compare Digest or Write Back Digest**

0: The digest is written to the Hash area.

1: The digest value is compared to the digest stored in the Hash area.

- **WRAP: Wrap Command**

0: The next region descriptor address loaded is the current region identifier descriptor address incremented by 0x10.

1: The next region descriptor address loaded is ICM_DSCR.

- **EOM: End Of Monitoring**

0: The current descriptor does not terminate the monitoring.

1: The current descriptor terminates the Main List. WRAP bit value has no effect.

- **RHIEN: Region Hash Completed Interrupt Disable (Default Enabled)**

0: The ICM_ISR RHC[*i*] flag is set when the field NEXT = 0 in a descriptor of the main or second list.

1: The ICM_ISR RHC[*i*] flag remains cleared even if the setting condition is met.

- **DMIEN: Digest Mismatch Interrupt Disable (Default Enabled)**

0: The ICM_ISR RBE[*i*] flag is set when the hash value just calculated from the processed region differs from expected hash value.

1: The ICM_ISR RBE[*i*] flag remains cleared even if the setting condition is met.

- **BEIEN: Bus Error Interrupt Disable (Default Enabled)**

0: The flag is set when an error is reported on the system bus by the bus MATRIX.

1: The flag remains cleared even if the setting condition is met.

- **WCIEN: Wrap Condition Interrupt Disable (Default Enabled)**

0: The ICM_ISR RWC[*i*] flag is set when the WRAP bit is set in a descriptor of the main list.

1: The ICM_ISR RWC[*i*] flag remains cleared even if the setting condition is met.

- **ECIEN: End Bit Condition Interrupt (Default Enabled)**

0: The ICM_ISR REC[*i*] flag is set when the descriptor having the EOM bit set is processed.

1: The ICM_ISR REC[*i*] flag remains cleared even if the setting condition is met.

- **SUIEN: Monitoring Status Updated Condition Interrupt (Default Enabled)**

0: The ICM_ISR RSU[*i*] flag is set when the corresponding descriptor is loaded from memory to ICM.

1: The ICM_ISR RSU[*i*] flag remains cleared even if the setting condition is met.

- **PROCDLY: Processing Delay**

Value	Name	Description
0	SHORTEST	SHA processing runtime is the shortest one
1	LONGEST	SHA processing runtime is the longest one

When SHA1 algorithm is processed, the runtime period is either 85 or 209 clock cycles.

When SHA256 or SHA224 algorithm is processed, the runtime period is either 72 or 194 clock cycles.

- **ALGO: SHA Algorithm**

Value	Name	Description
0	SHA1	SHA1 algorithm processed
1	SHA256	SHA256 algorithm processed
4	SHA224	SHA224 algorithm processed

Values which are not listed in the table must be considered as “reserved”.

42.5.2.3 ICM Region Control Structure Member

Name: ICM_RCTRL

Address: ICM_DSCR+0x008+RID*(0x10)

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
TRSIZE							

7	6	5	4	3	2	1	0
TRSIZE							

- **TRSIZE: Transfer Size for the Current Chunk of Data**

ICM performs a transfer of (TRSIZE + 1) blocks of 512 bits.

42.5.2.4 ICM Region Next Address Structure Member

Name: ICM_RNEXT

Address: ICM_DSCR+0x00C+RID*(0x10)

Access: Read/Write

31	30	29	28	27	26	25	24
NEXT							

23	22	21	20	19	18	17	16
NEXT							

15	14	13	12	11	10	9	8
NEXT							

7	6	5	4	3	2	1	0
NEXT						–	–

- **NEXT: Region Transfer Descriptor Next Address**

When configured to 0, this field indicates that the current descriptor is the last descriptor of the Secondary List, otherwise it points at a new descriptor of the Secondary List.

42.5.3 ICM Hash Area

The ICM Hash Area is a contiguous area of system memory that the controller and the processor can access. The physical location is configured in the ICM hash area start address register. This address is a multiple of 128 bytes. If the CDWBN bit of the context register is cleared (i.e., Write Back activated), the ICM controller performs a digest write operation at the following starting location: *(ICM_HASH) + (RID<<5), where RID is the current region context identifier. If the CDWBN bit of the context register is set (i.e., Digest Comparison activated), the ICM controller performs a digest read operation at the same address.

42.5.3.1 Message Digest Example

Considering the following **512 bits message** (example given in FIPS 180-2):

“61626380018”

The message is written to memory in a Little Endian (LE) system architecture.

Table 42-3. 512 bits Message Memory Mapping

Memory Address	Address Offset / Byte Lane			
	0x3 / 31:24	0x2 / 23:16	0x1 / 15:8	0x0 / 7:0
0x000	80	63	62	61
0x004 - 0x038	00	00	00	00
0x03C	18	00	00	00

The digest is stored at the memory location pointed at by the ICM_HASH pointer with a Region Offset.

Table 42-4. LE Resulting SHA-160 Message Digest Memory Mapping

Memory Address	Address Offset / Byte Lane			
	0x3 / 31:24	0x2 / 23:16	0x1 / 15:8	0x0 / 7:0
0x000	36	3e	99	a9
0x004	6a	81	06	47
0x008	71	25	3e	ba
0x00C	6c	c2	50	78
0x010	9d	d8	d0	9c

Table 42-5. Resulting SHA-224 Message Digest Memory Mapping

Memory Address	Address Offset / Byte Lane			
	0x3 / 31:24	0x2 / 23:16	0x1 / 15:8	0x0 / 7:0
0x000	22	7d	09	23
0x004	22	d8	05	34
0x008	77	a4	42	86
0x00C	b3	55	a2	bd
0x010	e4	bc	ad	2a
0x014	f7	b3	a0	bd
0x018	a7	9d	6c	e3

Table 42-6. Resulting SHA-256 Message Digest Memory Mapping

Memory Address	Address Offset / Byte Lane			
	0x3 / 31:24	0x2 / 23:16	0x1 / 15:8	0x0 / 7:0
0x000	bf	16	78	ba
0x004	ea	cf	01	8f
0x008	de	40	41	41
0x00C	23	22	ae	5d
0x010	a3	61	03	b0
0x014	9c	7a	17	96
0x018	61	ff	10	b4
0x01C	ad	15	00	f2

Considering the following **1024 bits message** (example given in FIPS 180-2):

"61626380000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000018"

The message is written to memory in a Little Endian (LE) system architecture.

Table 42-7. 1024 bits Message Memory Mapping

Memory Address	Address Offset / Byte Lane			
	0x3 / 31:24	0x2 / 23:16	0x1 / 15:8	0x0 / 7:0
0x000	80	63	62	61
0x004 - 0x078	00	00	00	00
0x07C	18	00	00	00

42.5.4 Using ICM as SHA Engine

The ICM can be configured to only calculate a SHA1, SHA224, SHA256 digest value.

42.5.4.1 Settings for Simple SHA Calculation

The start address of the system memory containing the data to hash must be configured in the transfer descriptor of the DMA embedded in the ICM.

The transfer descriptor is a system memory area integer multiple of 4 x 32-bit word and the start address of the descriptor must be configured in ICM_DSCR (the start address must be aligned on 64-bytes; six LSB must be cleared). If the data to hash is already padded according to SHA standards, only a single descriptor is required, and the EOM bit of ICM_RCFG must be written to 1. If the data to hash does not contain a padding area, it is possible to define the padding area in another system memory location, the ICM can be configured to automatically jump from a memory area to another one by configuring the descriptor register ICM_RNEXT with a value that differs from 0. Configuring the field NEXT of the ICM_RNEXT with the start address of the padding area forces the ICM to concatenate both areas, thus providing the SHA result from the start address of the hash area configured in ICM_HASH.

Whether the system memory is configured as a single or multiple data block area, the bits CDWBN and WRAP must be cleared in the region descriptor structure member ICM_RCFG. The bits WBDIS, EOMDIS, SLBDIS must be cleared in ICM_CFG.

The bits RHIEN or ECIEN must be written to 1 in the region descriptor structure member ICM_RCTRL. The flag RHC[*i*], *i* being the region index, is set (if RHIEN is set) when the hash result is available at address defined in

ICM_HASH. The flag REC[*i*], *i* being the region index, is set (if ECIEN is set) when the hash result is available at the address defined in ICM_HASH.

An interrupt is generated if the bit RHC[*i*] is written to 1 in the ICM_IER (if RHC[*i*] is set in ICM_RCTRL of region *i*) or if the bit REC[*i*] is written to 1 in the ICM_IER (if REC[*i*] is set in ICM_RCTRL of region *i*).

42.5.4.2 Processing Period

The SHA engine processing period can be configured.

The short processing period allows to allocate bandwidth to the SHA module whereas the long processing period allocates more bandwidth on the system bus to other applications.

In SHA mode, the shortest processing period is 85 clock cycles + 2 clock cycles for start command synchronization. The longest period is 209 clock cycles + 2 clock cycles.

In SHA256 and SHA224 modes, the shortest processing period is 72 clock cycles + 2 clock cycles for start command synchronization. The longest period is 194 clock cycles + 2 clock cycles.

42.5.5 ICM Automatic Monitoring Mode

The ASCD bit of the ICM_CFG register is used to activate the ICM Automatic mode. When ICM_CFG.ASCD is set, the ICM performs the following actions:

- The ICM controller passes through the Main List once with CDWBN bit in the context register at 0 (i.e., Write Back activated) and EOM bit in context register at 0.
- When WRAP = 1 in ICM_RCFG, the ICM controller enters active monitoring with CDWBN bit in context register now set and EOM bit in context register cleared. Bits CDWBN and EOM in ICM_RCFG have no effect.

42.5.6 Programming the ICM for Multiple Regions

Table 42-8. Region Attributes

Transfer Type		Main List	ICM_RCFG			ICM_RNEXT	Comments
			CDWBN	WRAP	EOM	NEXT	
Single Region	Contiguous list of blocks Digest written to memory Monitoring disabled	1 item	0	0	1	0	The Main List contains only one descriptor. The Secondary List is empty for that descriptor. The digest is computed and saved to memory.
	Non contiguous list of blocks Digest written to memory Monitoring disabled	1 item	0	0	1	Secondary List address of the current region identifier	The Main List contains only one descriptor. The Secondary List describes the layout of the non contiguous region.
	Contiguous list of blocks Digest comparison enabled Monitoring enabled	1 item	1	1	0	0	When the hash computation is terminated, the digest is compared with the one saved in memory.
Multiple Regions	Contiguous list of blocks Digest written to memory Monitoring disabled	More than one item	0	0	1 for the last, 0 otherwise	0	ICM passes through the list once.
	Contiguous list of blocks Digest comparison is enabled Monitoring is enabled	More than one item	1	1 for the last, 0 otherwise	0	0	ICM performs active monitoring of the regions. If a mismatch occurs, an interrupt is raised.
	Non contiguous list of blocks Digest is written to memory Monitoring is disabled	More than one item	0	0	1	Secondary List address	ICM performs hashing and saves digests to the Hash area.
	Non contiguous list of blocks Digest comparison is enabled Monitoring is enabled	More than one item	1	1	0	Secondary List address	ICM performs data gathering on a per region basis.

42.5.7 Security Features

When an undefined register access occurs, the URAD bit in the Interrupt Status Register (ICM_ISR) is set if unmasked. Its source is then reported in the Undefined Access Status Register (ICM_UASR). Only the first undefined register access is available through the ICM_UASR.URAT field.

Several kinds of unspecified register accesses can occur:

- Unspecified structure member set to one detected when the descriptor is loaded.
- Configuration register (ICM_CFG) modified during active monitoring.
- Descriptor register (ICM_DSCR) modified during active monitoring.
- Hash register (ICM_HASH) modified during active monitoring.
- Write-only register read access.

The URAD bit and the URAT field can only be reset by writing a 1 to the ICM_CTRL.SWRST bit.

42.6 Integrity Check Monitor (ICM) User Interface

Table 42-9. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Configuration Register	ICM_CFG	Read/Write	0x0
0x04	Control Register	ICM_CTRL	Write-only	–
0x08	Status Register	ICM_SR	Read-only	–
0x0C	Reserved	–	–	–
0x10	Interrupt Enable Register	ICM_IER	Write-only	–
0x14	Interrupt Disable Register	ICM_IDR	Write-only	–
0x18	Interrupt Mask Register	ICM_IMR	Read-only	0x0
0x1C	Interrupt Status Register	ICM_ISR	Read-only	0x0
0x20	Undefined Access Status Register	ICM_UASR	Read-only	0x0
0x24 - 0x2C	Reserved	–	–	–
0x30	Region Descriptor Area Start Address Register	ICM_DSCR	Read/Write	0x0
0x34	Region Hash Area Start Address Register	ICM_HASH	Read/Write	0x0
0x38	User Initial Hash Value 0 Register	ICM_UIHVAL0	Write-only	–
...	...	...	...	...
0x54	User Initial Hash Value 7 Register	ICM_UIHVAL7	Write-only	–
0x58 - 0xFC	Reserved	–	–	–

42.6.1 ICM Configuration Register

Name: ICM_CFG

Address: 0x40044000

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
UALGO			UIHASH	–	–	DUALBUFF	ASCD

7	6	5	4	3	2	1	0
BBC				–	SLBDIS	EOMDIS	WBDIS

- **WBDIS: Write Back Disable**

0: Write Back Operations are permitted.

1: Write Back Operations are forbidden. Context register CDWBN bit is internally set to one and cannot be modified by a linked list element. The CDWBN bit of the ICM_RCFG structure member has no effect.

When ASCD bit of the ICM_CFG register is set, WBDIS bit value has no effect.

- **EOMDIS: End of Monitoring Disable**

0: End of Monitoring is permitted.

1: End of Monitoring is forbidden. The EOM bit of the ICM_RCFG structure member has no effect.

- **SLBDIS: Secondary List Branching Disable**

0: Branching to the Secondary List is permitted.

1: Branching to the Secondary List is forbidden. The NEXT field of the ICM_RNEXT structure member has no effect and is always considered as zero.

- **BBC: Bus Burden Control**

This field is used to control the burden of the ICM system bus. The number of system clock cycles between the end of the current processing and the next block transfer is set to 2^{BBC}. Up to 32,768 cycles can be inserted.

- **ASCD: Automatic Switch To Compare Digest**

0: Automatic mode is disabled.

1: When this mode is enabled, the ICM controller automatically switches to active monitoring after the first Main List pass. Both CDWBN and WBDIS bits have no effect. A one must be written to the EOM bit in ICM_RCFG to terminate the monitoring.

- **DUALBUFF: Dual Input Buffer**

0: Dual Input Buffer mode is disabled.

1: Dual Input Buffer mode is enabled (better performances, higher bandwidth required on system bus).

- **UIHASH: User Initial Hash Value**

0: The secure hash standard provides the initial hash value.

1: The initial hash value is programmable. Field UALGO provides the SHA algorithm. The ALGO field of the ICM_RCFG structure member has no effect.

- **UALGO: User SHA Algorithm**

Value	Name	Description
0	SHA1	SHA1 algorithm processed
1	SHA256	SHA256 algorithm processed
4	SHA224	SHA224 algorithm processed

42.6.2 ICM Control Register

Name: ICM_CTRL

Address: 0x40044004

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
RMEN				RMDIS			

7	6	5	4	3	2	1	0
REHASH				–	SWRST	DISABLE	ENABLE

- **ENABLE: ICM Enable**

0: No effect.

1: When set to one, the ICM controller is activated.

- **DISABLE: ICM Disable Register**

0: No effect.

1: The ICM controller is disabled. If a region is active, this region is terminated.

- **SWRST: Software Reset**

0: No effect.

1: Resets the ICM controller.

- **REHASH: Recompute Internal Hash**

0: No effect.

1: When REHASH[*i*] is set to one, Region *i* digest is re-computed. This bit is only available when region monitoring is disabled.

- **RMDIS: Region Monitoring Disable**

0: No effect.

1: When bit RMDIS[*i*] is set to one, the monitoring of region with identifier *i* is disabled.

- **RMEN: Region Monitoring Enable**

0: No effect.

1: When bit RMEN[*i*] is set to one, the monitoring of region with identifier *i* is activated.

Monitoring is activated by default.

42.6.3 ICM Status Register

Name: ICM_SR

Address: 0x40044008

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
RMDIS				RAWRMDIS			

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	ENABLE

- **ENABLE: ICM Controller Enable Register**

0: ICM controller is disabled.

1: ICM controller is activated.

- **RAWRMDIS: Region Monitoring Disabled Raw Status**

0: Region *i* monitoring has been activated by writing a 1 in RMEN[*i*] of ICM_CTRL.

1: Region *i* monitoring has been deactivated by writing a 1 in RMDIS[*i*] of ICM_CTRL.

- **RMDIS: Region Monitoring Disabled Status**

0: Region *i* monitoring is being monitored (occurs after integrity check value has been calculated and written to Hash area).

1: Region *i* monitoring is not being monitored.

42.6.4 ICM Interrupt Enable Register

Name: ICM_IER

Address: 0x40044010

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	URAD

23	22	21	20	19	18	17	16
RSU				REC			

15	14	13	12	11	10	9	8
RWC				RBE			

7	6	5	4	3	2	1	0
RDM				RHC			

- **RHC: Region Hash Completed Interrupt Enable**

0: No effect.

1: When RHC[*i*] is set to one, the Region *i* Hash Completed interrupt is enabled.

- **RDM: Region Digest Mismatch Interrupt Enable**

0: No effect.

1: When RDM[*i*] is set to one, the Region *i* Digest Mismatch interrupt is enabled.

- **RBE: Region Bus Error Interrupt Enable**

0: No effect.

1: When RBE[*i*] is set to one, the Region *i* Bus Error interrupt is enabled.

- **RWC: Region Wrap Condition detected Interrupt Enable**

0: No effect.

1: When RWC[*i*] is set to one, the Region *i* Wrap Condition interrupt is enabled.

- **REC: Region End bit Condition Detected Interrupt Enable**

0: No effect.

1: When REC[*i*] is set to one, the region *i* End bit Condition interrupt is enabled.

- **RSU: Region Status Updated Interrupt Enable**

0: No effect.

1: When RSU[*i*] is set to one, the region *i* Status Updated interrupt is enabled.

- **URAD: Undefined Register Access Detection Interrupt Enable**

0: No effect.

1: The Undefined Register Access interrupt is enabled.

42.6.5 ICM Interrupt Disable Register

Name: ICM_IDR

Address: 0x40044014

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	URAD

23	22	21	20	19	18	17	16
RSU				REC			

15	14	13	12	11	10	9	8
RWC				RBE			

7	6	5	4	3	2	1	0
RDM				RHC			

- **RHC: Region Hash Completed Interrupt Disable**

0: No effect.

1: When RHC[*i*] is set to one, the Region *i* Hash Completed interrupt is disabled.

- **RDM: Region Digest Mismatch Interrupt Disable**

0: No effect.

1: When RDM[*i*] is set to one, the Region *i* Digest Mismatch interrupt is disabled.

- **RBE: Region Bus Error Interrupt Disable**

0: No effect.

1: When RBE[*i*] is set to one, the Region *i* Bus Error interrupt is disabled.

- **RWC: Region Wrap Condition Detected Interrupt Disable**

0: No effect.

1: When RWC[*i*] is set to one, the Region *i* Wrap Condition interrupt is disabled.

- **REC: Region End bit Condition detected Interrupt Disable**

0: No effect.

1: When REC[*i*] is set to one, the region *i* End bit Condition interrupt is disabled.

- **RSU: Region Status Updated Interrupt Disable**

0: No effect.

1: When RSU[*i*] is set to one, the region *i* Status Updated interrupt is disabled.

- **URAD: Undefined Register Access Detection Interrupt Disable**

0: No effect.

1: Undefined Register Access Detection interrupt is disabled.

42.6.6 ICM Interrupt Mask Register

Name: ICM_IMR

Address: 0x40044018

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	URAD

23	22	21	20	19	18	17	16
RSU				REC			

15	14	13	12	11	10	9	8
RWC				RBE			

7	6	5	4	3	2	1	0
RDM				RHC			

- **RHC: Region Hash Completed Interrupt Mask**

0: When RHC[i] is set to zero, the interrupt is disabled for region i.

1: When RHC[i] is set to one, the interrupt is enabled for region i.

- **RDM: Region Digest Mismatch Interrupt Mask**

0: When RDM[i] is set to zero, the interrupt is disabled for region i.

1: When RDM[i] is set to one, the interrupt is enabled for region i.

- **RBE: Region Bus Error Interrupt Mask**

0: When RBE[i] is set to zero, the interrupt is disabled for region i.

1: When RBE[i] is set to one, the interrupt is enabled for region i.

- **RWC: Region Wrap Condition Detected Interrupt Mask**

0: When RWC[i] is set to zero, the interrupt is disabled for region i.

1: When RWC[i] is set to one, the interrupt is enabled for region i.

- **REC: Region End bit Condition Detected Interrupt Mask**

0: When REC[i] is set to zero, the interrupt is disabled for region i.

1: When REC[i] is set to one, the interrupt is enabled for region i.

- **RSU: Region Status Updated Interrupt Mask**

0: When RSU[i] is set to zero, the interrupt is disabled for region i.

1: When RSU[i] is set to one, the interrupt is enabled for region i.

- **URAD: Undefined Register Access Detection Interrupt Mask**

0: Interrupt is disabled.

1: Interrupt is enabled.

42.6.7 ICM Interrupt Status Register

Name: ICM_ISR

Address: 0x4004401C

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	URAD

23	22	21	20	19	18	17	16
RSU				REC			

15	14	13	12	11	10	9	8
RWC				RBE			

7	6	5	4	3	2	1	0
RDM				RHC			

- **RHC: Region Hash Completed**

When RHC[*i*] is set, it indicates that the ICM has completed the region with identifier *i*.

- **RDM: Region Digest Mismatch**

When RDM[*i*] is set, it indicates that there is a digest comparison mismatch between the hash value of the region with identifier *i* and the reference value located in the Hash Area.

- **RBE: Region Bus Error**

When RBE[*i*] is set, it indicates that a bus error has been detected while hashing memory region *i*.

- **RWC: Region Wrap Condition Detected**

When RWC[*i*] is set, it indicates that a wrap condition has been detected.

- **REC: Region End bit Condition Detected**

When REC[*i*] is set, it indicates that an end bit condition has been detected.

- **RSU: Region Status Updated Detected**

When RSU[*i*] is set, it indicates that a region status updated condition has been detected.

- **URAD: Undefined Register Access Detection Status**

0: No undefined register access has been detected since the last SWRST.

1: At least one undefined register access has been detected since the last SWRST.

The URAD bit is only reset by the SWRST bit in the ICM_CTRL register.

The URAT field in the ICM_UASR indicates the unspecified access type.

42.6.8 ICM Undefined Access Status Register

Name: ICM_UASR

Address: 0x40044020

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
–	–	–	–	–	URAT		

- **URAT: Undefined Register Access Trace**

Value	Name	Description
0	UNSPEC_STRUCT_MEMBER	Unspecified structure member set to one detected when the descriptor is loaded
1	ICM_CFG_MODIFIED	ICM_CFG modified during active monitoring
2	ICM_DSCR_MODIFIED	ICM_DSCR modified during active monitoring
3	ICM_HASH_MODIFIED	ICM_HASH modified during active monitoring
4	READ_ACCESS	Write-only register read access

Only the first Undefined Register Access Trace is available through the URAT field.

The URAT field is only reset by the SWRST bit in the ICM_CTRL register.

42.6.9 ICM Descriptor Area Start Address Register

Name: ICM_DSCR

Address: 0x40044030

Access: Read/Write

31	30	29	28	27	26	25	24
DASA							

23	22	21	20	19	18	17	16
DASA							

15	14	13	12	11	10	9	8
DASA							

7	6	5	4	3	2	1	0
DASA		–	–	–	–	–	–

- **DASA: Descriptor Area Start Address**

The start address is a multiple of the total size of the data structure (64 bytes).

42.6.10 ICM Hash Area Start Address Register

Name: ICM_HASH

Address: 0x40044034

Access: Read/Write

31	30	29	28	27	26	25	24
HASA							

23	22	21	20	19	18	17	16
HASA							

15	14	13	12	11	10	9	8
HASA							

7	6	5	4	3	2	1	0
HASA	–	–	–	–	–	–	–

- **HASA: Hash Area Start Address**

This field points at the Hash memory location. The address must be a multiple of 128 bytes.

42.6.11 ICM User Initial Hash Value Register

Name: ICM_UIHVALx [x=0..7]

Address: 0x40044038

Access: Write-only

31	30	29	28	27	26	25	24
VAL							

23	22	21	20	19	18	17	16
VAL							

15	14	13	12	11	10	9	8
VAL							

7	6	5	4	3	2	1	0
VAL							

- **VAL: Initial Hash Value**

When UIHASH bit of IMC_CFG register is set, the Initial Hash Value is user-programmable.

To meet the desired standard, use the following example values.

For ICM_UIHVAL0 field:

Example	Comment
0x67452301	SHA1 algorithm
0xC1059ED8	SHA224 algorithm
0x6A09E667	SHA256 algorithm

For ICM_UIHVAL1 field:

Example	Comment
0xEFCDAB89	SHA1 algorithm
0x367CD507	SHA224 algorithm
0xBB67AE85	SHA256 algorithm

For ICM_UIHVAL2 field:

Example	Comment
0x98BADCFE	SHA1 algorithm
0x3070DD17	SHA224 algorithm
0x3C6EF372	SHA256 algorithm

For ICM_UIHVAL3 field:

Example	Comment
0x10325476	SHA1 algorithm
0xF70E5939	SHA224 algorithm
0xA54FF53A	SHA256 algorithm

For ICM_UIHVAL4 field:

Example	Comment
0xC3D2E1F0	SHA1 algorithm
0xFFC00B31	SHA224 algorithm
0x510E527F	SHA256 algorithm

For ICM_UIHVAL5 field:

Example	Comment
0x68581511	SHA224 algorithm
0x9B05688C	SHA256 algorithm

For ICM_UIHVAL6 field:

Example	Comment
0x64F98FA7	SHA224 algorithm
0x1F83D9AB	SHA256 algorithm

For ICM_UIHVAL7 field:

Example	Comment
0xBEFA4FA4	SHA224 algorithm
0x5BE0CD19	SHA256 algorithm

Example of Initial Value for SHA-1 Algorithm

Register Address	Address Offset / Byte Lane			
	0x3 / 31:24	0x2 / 23:16	0x1 / 15:8	0x0 / 7:0
0x000 ICM_UIHVAL0	01	23	45	67
0x004 ICM_UIHVAL1	89	ab	cd	ef
0x008 ICM_UIHVAL2	fe	dc	ba	98
0x00C ICM_UIHVAL3	76	54	32	10
0x010 ICM_UIHVAL4	f0	e1	d2	c3

43. Classical Public Key Cryptography Controller (CPKCC)

43.1 Description

The Classical Public Key Cryptography Controller (CPKCC) is an Atmel macrocell that processes public key cryptography algorithm calculus in both GF(p) and GF(2^n) fields. The ROMed CPKCL, the Classical Public Key Cryptography Library, is the library built on the top of the CPKCC.

The Classical Public Key Cryptography Library includes complete implementation of the following public key cryptography algorithms:

- RSA, DSA:
  - Modular Exponentiation with CRT up to 6144 bits.
  - Modular Exponentiation without CRT up to 5408 bits.
  - Prime generation.
  - Utilities: GCD/modular Inverse, Divide, Modular reduction, Multiply…
- Elliptic Curves:
  - ECDSA up to 1504 bits.
  - Point Multiply.
  - Point Add/Doubling.
  - Elliptic Curves in GF(p) or GF(2^n).
  - Choice of the curves parameters so compatibility with NIST Curves or others.
- Deterministic Random Number Generation (DRNG ANSI X9.31) for DSA.

43.2 Product Dependencies

43.2.1 Power Management

The CPKCC is not continuously clocked. The CPCKCC interface is clocked through the Power Management Controller (PMC).

43.2.2 Interrupt Sources

The CPKCC has an interrupt line connected to the Nested Vector Interrupt Controller (NVIC). Handling interrupts requires programming the NVIC before configuring the CPKCC.

43.3 Functional Description

The CPKCC macrocell is managed by the CPKCL Library available in the ROM memory of the microcontroller. The user interface of the CPKCC is not described in this chapter.

The usage description of the CPKCC and its associated Library is provided in a separate document. Contact an Atmel Sales Representative for further details.

44. True Random Number Generator (TRNG)

44.1 Description

The True Random Number Generator (TRNG) passes the American *NIST Special Publication 800-22 (A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications)* and the *Diehard Suite of Tests.*

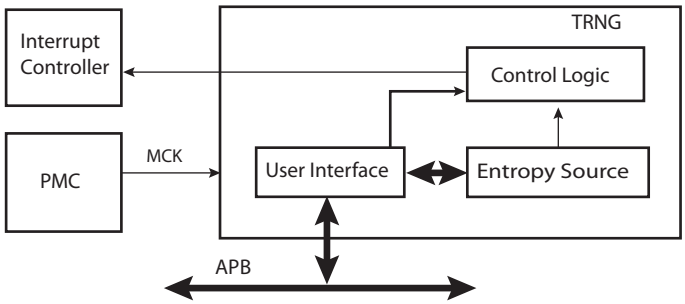
The TRNG may be used as an entropy source for seeding an NIST approved DRNG (Deterministic RNG) as required by FIPS PUB 140-2 and 140-3.

44.2 Embedded Characteristics

- Passes NIST Special Publication 800-22 Test Suite.
- Passes Diehard Suite of Tests.
- May be Used as Entropy Source for Seeding a NIST-approved DRNG (Deterministic RNG) as required by FIPS PUB 140-2 and 140-3.
- Provides a 32-bit Random Number Every 84 Clock Cycles.

44.3 Block Diagram

Figure 44-1. TRNG Block Diagram



44.4 Product Dependencies

44.4.1 Power Management

The TRNG interface may be clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the TRNG user interface clock. The user interface clock is independent from any clock that may be used in the entropy source logic circuitry. The source of entropy can be enabled before enabling the user interface clock.

44.4.2 Interrupt Sources

The TRNG interface has an interrupt line connected to the Interrupt Controller. In order to handle interrupts, the Interrupt Controller must be programmed before configuring the TRNG.

Table 44-1. Peripheral IDs

Instance	ID
TRNG	33

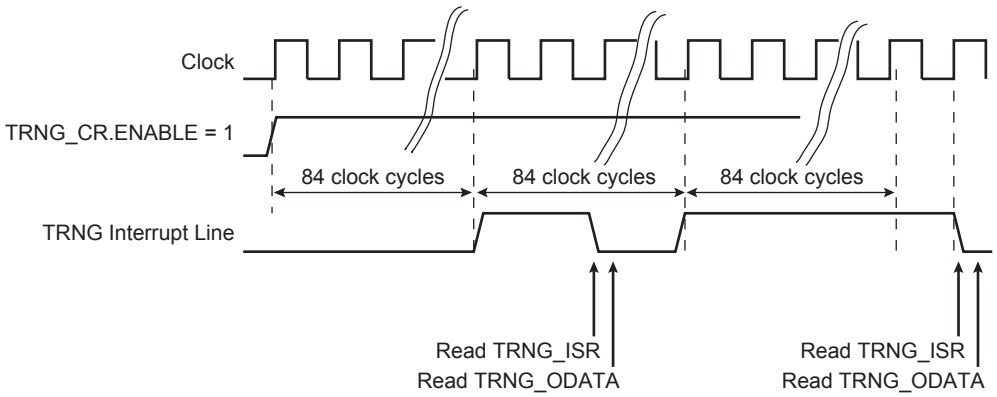
44.5 Functional Description

As soon as the TRNG is enabled in the Control register (TRNG_CR), the generator provides one 32-bit value every 84 clock cycles.

The TRNG interrupt line can be enabled in the Interrupt Enable register (TRNG_IER), and disabled in the Interrupt Disable register (TRNG_IDR). This interrupt is set when a new random value is available and is cleared when the Status register (TRNG_ISR) is read. The flag TRNG_ISR.DATRDY is set when the random data is ready to be read out on the 32-bit Output Data register (TRNG_ODATA).

The normal mode of operation checks that the flag in TRNG_ISR equals '1' before reading TRNG_ODATA when a 32-bit random value is required by the software application.

Figure 44-2. TRNG Data Generation Sequence



44.6 True Random Number Generator (TRNG) User Interface

Table 44-2. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	TRNG_CR	Write-only	–
0x04 - 0x0C	Reserved	–	–	–
0x10	Interrupt Enable Register	TRNG_IER	Write-only	–
0x14	Interrupt Disable Register	TRNG_IDR	Write-only	–
0x18	Interrupt Mask Register	TRNG_IMR	Read-only	0x0000_0000
0x1C	Interrupt Status Register	TRNG_ISR	Read-only	0x0000_0000
0x20 - 0x4C	Reserved	–	–	–
0x50	Output Data Register	TRNG_ODATA	Read-only	0x0000_0000
0x54 - 0xFC	Reserved	–	–	–

44.6.1 TRNG Control Register

Name: TRNG_CR

Address: 0x40048000

Access: Write-only

31	30	29	28	27	26	25	24
KEY							

23	22	21	20	19	18	17	16
KEY							

15	14	13	12	11	10	9	8
KEY							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	ENABLE

- **ENABLE: Enables the TRNG to Provide Random Values**

0: Disables the TRNG.

1: Enables the TRNG if 0x524E47 (“RNG” in ASCII) is written in KEY field at the same time.

- **KEY: Security Key**

Value	Name	Description
0x524E47	PASSWD	Writing any other value in this field aborts the write operation.

44.6.2 TRNG Interrupt Enable Register

Name: TRNG_IER

Address: 0x40048010

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	DATRDY

- **DATRDY: Data Ready Interrupt Enable**

0: No effect.

1: Enables the corresponding interrupt.

44.6.3 TRNG Interrupt Disable Register

Name: TRNG_IDR

Address: 0x40048014

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	DATRDY

- **DATRDY: Data Ready Interrupt Disable**

0: No effect.

1: Disables the corresponding interrupt.

44.6.4 TRNG Interrupt Mask Register

Name: TRNG_IMR

Address: 0x40048018

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	DATRDY

- **DATRDY: Data Ready Interrupt Mask**

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

44.6.5 TRNG Interrupt Status Register

Name: TRNG_ISR

Address: 0x4004801C

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
		–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	DATRDY

- **DATRDY: Data Ready (Cleared on Read)**

0: Output data is not valid or TRNG is disabled.

1: New random value is completed since the last read of TRNG_ODATA.

44.6.6 TRNG Output Data Register

Name: TRNG_ODATA

Address: 0x40048050

Access: Read-only

31	30	29	28	27	26	25	24
ODATA							

23	22	21	20	19	18	17	16
ODATA							

15	14	13	12	11	10	9	8
ODATA							

7	6	5	4	3	2	1	0
ODATA							

- **ODATA: Output Data**

The 32-bit Output Data register contains the 32-bit random data.

45. Electrical Characteristics

45.1 Absolute Maximum Ratings

Table 45-1. Absolute Maximum Ratings*

Storage Temperature ..... -55°C to + 125°C
Power Supply inputs with respect to ground pins:
VDDCORE, VDDPLL, VDDOUT PLC, VDDPLL PLC, VDDOUT AN 1.4V
VDDBU, VDDIO, VDDIN, VDDLCD, VDDIN PLC, VDDIN AN 4.0V
Voltage on VDDIO Digital Input Pins with Respect to ground ..... -0.3V to VDDIO +0.3V
Voltage on VDDBU Digital Input Pins with Respect to ground ..... -0.3V to VDDBU +0.3V

*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.**

45.2 Recommended Operating Conditions

Table 45-2. Recommended Operating Conditions on Power supply Inputs

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
VDDCORE	Core logic power supply		1.08	1.20	1.32	V
VDDBU	Backup region power supply		1.62	3.3	3.6	V
VDDIO	I/Os power supply		3.0	3.3	3.6	V
VDDIN	Analog cells (Voltage Regulators, 10-bit ADC, temperature sensor) power supply		2.5	3.3	3.6	V
VDDLCD	LCD output buffers power supply		2.4	—	3.6	V
VDDPLL	PLLs and Main crystal oscillator power supply		1.08	1.20	1.32	V
VDDIN PLC	PLC Digital Regulator input		3.0	3.3	3.6	V
VDDIN AN	PLC Analog Regulator input		3.0	3.3	3.6	V
VDDPLL PLC	PLC PLL Power Supply		1.08	1.20	1.32	V
f_{MCK}	Master clock frequency	VDDCORE @ 1.20V, T_A = 85°C VDDCORE @ 1.08V, T_A = 85°C	—	—	120 100	MHz

Table 45-3. Recommended Operating Conditions on Input Pins

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$AD\ [x]_{IN}$	Input voltage range on 10-bit ADC analog inputs	On AD[0..x]	0	—	min (VDDIN, VDDIO)	V
V_{GPIO_IN}	Input voltage range on GPIOs	On any pin configured as a digital input	0	—	VDDIO	V
V_{VDDBU_IN}	Input voltage range on inputs referenced to VDDBU	On FWUP, TMP0 and XIN32 inputs	0	—	VDDBU	V

Table 45-4. Recommended Thermal Operating Conditions

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
T_A	Ambient temperature range		-40	—	85	°C
T_J	Junction temperature range		-40	—	100	
R_{JA}	Junction-to-ambient thermal resistance	LQFP 176 package	—	38	—	°C / W
P_D	Power dissipation	T_A = 70 ºC T_A = 85 ºC	—	—	789 394	mW

45.3 Electrical Parameters Usage

The tables that follow further on in Section 45.4 “I/O Characteristics”, Section 45.5 “Embedded Analog Peripherals Characteristics”, Section 45.6 “Embedded Flash Characteristics” and Section 45.7 “Power Supply Current Consumption” define the limiting values for several electrical parameters. Unless otherwise noted, these values are valid over the ambient temperature range T_A= [-40°C + 85°C]. Note that these limits may be affected by the board on which the MCU is mounted. Particularly, noisy supply and ground conditions must be avoided and care must be taken to provide:

- a PCB with a low impedance ground plane (unbroken ground planes are strongly recommended).
- low impedance decoupling of the MCU power supply inputs. A 100 nF Ceramic X7R (or X5R) capacitor placed very close to each power supply input is a minimum requirement. See special recommendations regarding integrated analog functions like voltage reference or voltage regulators in corresponding sections.
- low impedance power supply decoupling of external components. This recommendation aims at avoiding current spikes travelling into the PCB ground plane.

45.4 I/O Characteristics

45.4.1 I/O DC Characteristics

Table 45-5. I/O DC Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IL} (1)	Low-level input voltage	3.0V < VDDIO < 3.6V	—	—	0.3 x VDDIO	V
V_{IH} (1)	High-level input voltage	3.0V < VDDIO < 3.6V	0.7 x VDDIO	—	—	V
V_{OH} (2)	High-level output voltage	3.0V < VDDIO < 3.6V I_{OH} = 0 I_{OH} > 0 (See IOH details below)	 VDDIO VDDIO - 0.4	—	—	V

Table 45-5. I/O DC Characteristics (Continued)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{OL}[2]	Low-level output voltage	3.0V < VDDIO < 3.6V I_{OL} = 0 I_{OL} > 0 (See IOL details below)	—	—	 0 0.4	V
I_{OH}	High-Level output current	**PA0,PA29,PB13,PC0,PC5 pins**[3]				mA
		VDDIO = 3.0V; V_{OH} = V_{VDDIO} - 0.4 VDDIO = 3.3V; V_{OH} = V_{VDDIO} - 0.4 VDDIO = 3.6V; V_{OH} = V_{VDDIO} - 0.4	—	—	-7 -7 -11	
		Other GPIO pins, Low Drive[2][4]				
		VDDIO = 3.0V; V_{OH} = V_{VDDIO} - 0.4 VDDIO = 3.3V; V_{OH} = V_{VDDIO} - 0.4 VDDIO = 3.6V; V_{OH} = V_{VDDIO} - 0.4	—	—	-3 -5 -6	
		Other GPIO pins, High Drive[4]				
		VDDIO = 3.0V; V_{OH} = V_{VDDIO} - 0.4 VDDIO = 3.3V; V_{OH} = V_{VDDIO} - 0.4 VDDIO = 3.6V; V_{OH} = V_{VDDIO} - 0.4	—	—	-6 -8 -8	
		Relaxed Mode[5] **PA0,PA29,PB13,PC0,PC5 pins**[3]				
		VDDIO = 3.0V; V_{OH} = V_{VDDIO} - 0.6 VDDIO = 3.3V; V_{OH} = 2.2V VDDIO = 3.6V; V_{OH} = 2.4V	—	—	-12 -22 -26	
		Relaxed Mode[5] **Other GPIO pins, Low Drive**[2][4]				
		VDDIO = 3.0V; V_{OH} = V_{VDDIO} - 0.6 VDDIO = 3.3V; V_{OH} = 2.2V VDDIO = 3.6V; V_{OH} = 2.4V	—	—	-5 -12 -13	
		Relaxed Mode[5] **Other GPIO pins, High Drive**[4]				
		VDDIO = 3.0V; V_{OH} = V_{VDDIO} - 0.6 VDDIO = 3.3V; V_{OH} = 2.2V VDDIO = 3.6V; V_{OH} = 2.4V	—	—	-10 -20 -24	
I_{OL}	Low-Level output current	**3.0V < VDDIO < 3.6V; V_{OL} = 0.4V**				mA
		PA0,PA29,PB13,PC0,PC5 pins[3] Other GPIO pins, Low Drive[2][4] Other GPIO pins, High Drive[4]	—	—	9 8 10	
		Relaxed Mode[5] **3.0V < VDDIO < 3.6V; V_{OL} = 0.6V**				
		PA0,PA29,PB13,PC0,PC5 pins[3] Other GPIO pins, Low Drive[2][4] Other GPIO pins, High Drive[4]	—	—	13 10 13	
V_{HYST}	Hysteresis Voltage	Hysteresis mode enabled	150	—	—	mV

Table 45-5. I/O DC Characteristics (Continued)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
I_{IL}	Input Low Leakage Current	No pull-up or pull-down; V_{IN} = GND; V_{VDDIO} Max. (Typ: T_A = 25°C, Max: T_A = 85°C) - PA0,PA29,PB13,PC0,PC5 pins(3) - PB16,PB17,PB18,PB19,PB20,PB21 pins - Other pins	—	 12 5 4	 57 41 7	nA
I_{IH}	Input High Leakage Current	No pull-up or pull-down; V_{IN} = VDD; V_{VDDIO} Max. (Typ: T_A = 25°C, Max: T_A = 85°C) - PA0,PA29,PB13,PC0,PC5 pins(3) - PB16,PB17,PB18,PB19,PB20,PB21 pins - Other pins	—	 15 1.7 5	 150 9 14	nA
R_{PULLUP}	Pull-up Resistor	Digital Input Mode	70	100	130	kΩ
$R_{PULLDOWN}$	Pull-down Resistor		70	100	130	kΩ
R_{ODT}	On-die Series Termination Resistor		—	36	—	Ω
C_{PAD}	Input Capacitance	I/O configured as digital input	—	—	5	pF

Notes:
1. Read VDDBU instead of VDDIO for VDDBU-referenced inputs (XIN32, FWUP, TMP0/WKUP0).
2. Read VDDBU instead of VDDIO for SHDN output referenced to VDDBU.
3. These I/O lines have permanent non-programmable maximum drive (MaxDRV).
4. Refer to “Peripheral Signal Multiplexing on I/O Lines” tables in Section “Peripherals”.
5. The Relaxed mode applies for cases of higher output current on the I/O lines that override standard V_{OL} and V_{OH} definitions.

45.4.2 I/O AC Characteristics

Criteria used to define the maximum frequency of the I/Os:

- Output duty cycle (40% - 60%).
- Minimum output swing: 100 mV to VDDIO - 100 mV.
- Minimum output swing: 100 mV to VDDIO - 100 mV.
- Addition of rising and falling time inferior to 75% of the period.

Table 45-6. Output AC Characteristics

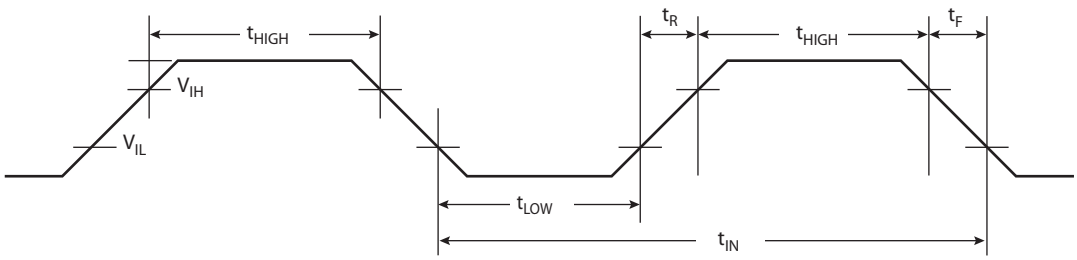
Symbol	Parameter	Conditions		Min	Max	Unit
FreqMax1	Pin Group 1 (1) Maximum output frequency	10 pF	V_{VDDIO} = 3.3V	—	70	MHz
		30 pF		—	45	
FreqMax2	Pin Group 2 (2)Maximum output frequency	10 pF	V_{VDDIO} = 3.3V	—	35	
		25 pF		—	15	
FreqMax3	Pin Group3 (3)Maximum output frequency	10 pF	V_{VDDIO} = 3.3V	—	70	
		25 pF		—	35	

Notes:
1. Pin Group 1 = PA0, PA29, PB13, PC0, PC5 pins.
2. Other pins, low drive settings.
3. Other pins, high drive settings.

Table 45-7 provides the input characteristics of the I/O lines with voltage references to VDDIO. In particular, these values apply when the XIN input is used as a clock input of the device (oscillator set in bypass mode). They do not apply for the XIN32 input which is made for slow signals with frequencies up to 50 kHz. V_{IL} and V_{IH} parameters defined in Table 45-5 apply.

Table 45-7. Input AC Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{IN}	Input frequency		—	—	50	MHz
t_{IN}	Input period		20	—	—	ns
t_{HIGH}	Time at high level		8	—	—	
t_{LOW}	Time at low level		8	—	—	
t_R	Rise Time		—	—	2.2	
t_F	Fall Time		—	—	2.2	



45.4.3 SPI Characteristics

Figure 45-1. SPI Master Mode with (CPOL= NCPHA = 0) or (CPOL= NCPHA= 1)

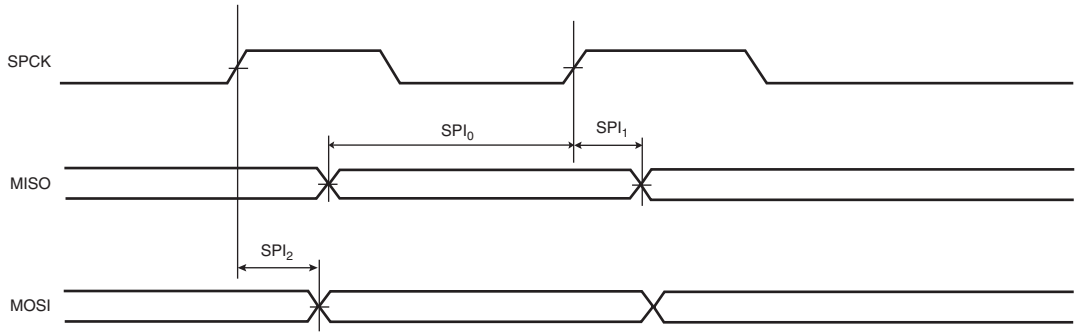


Figure 45-2. SPI Master Mode with (CPOL = 0 and NCPHA=1) or (CPOL=1 and NCPHA= 0)

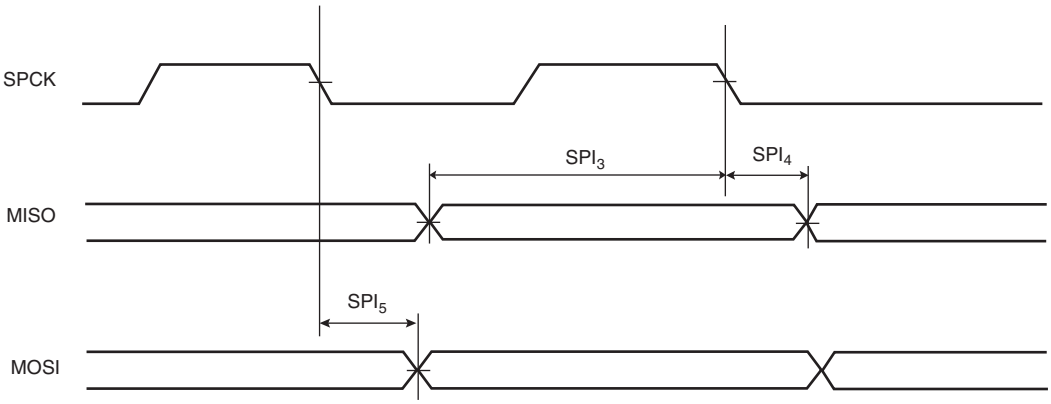


Figure 45-3. SPI Slave Mode with (CPOL=0 and NCPHA=1) or (CPOL=1 and NCPHA=0)

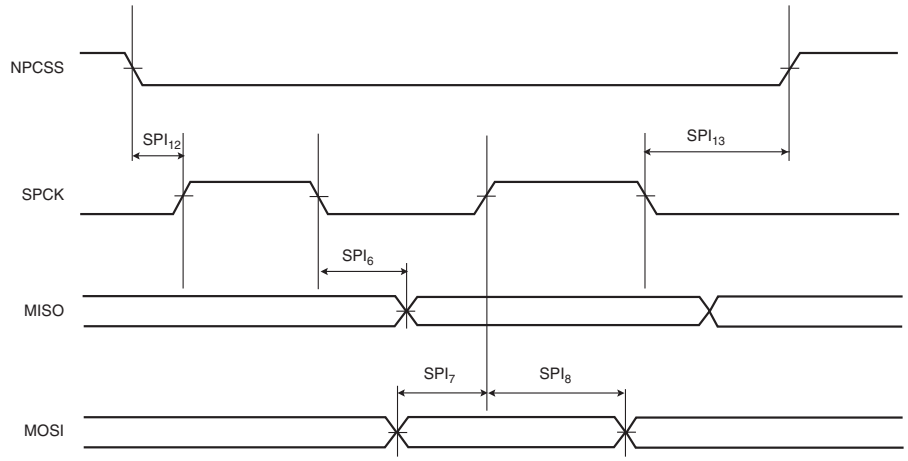
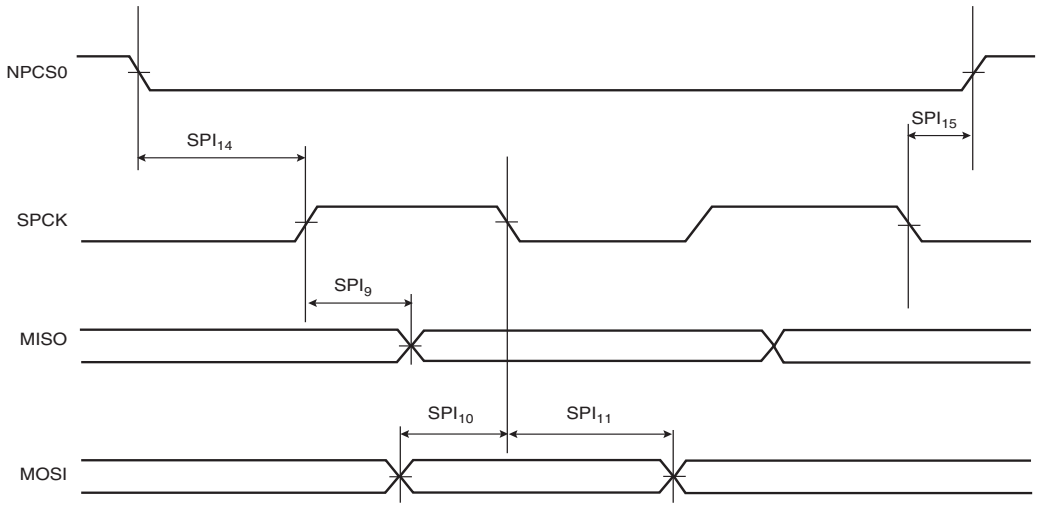


Figure 45-4. SPI Slave Mode with (CPOL = NCPHA = 0) or (CPOL= NCPHA= 1)



45.4.3.1 Maximum SPI Frequency

The formulas that follow give the maximum SPI frequency in Master Write and Read modes, and in Slave Read and Write modes.

Master Write Mode

The SPI is only sending data to a slave device such as an LCD, for example. The limit is given by SPI_2 (or SPI_5) timing. Since it gives a maximum frequency above the maximum pad speed (see Section 45.4.2 “I/O AC Characteristics”), the max SPI frequency is the one from the pad.

Master Read Mode

$$f_{SPCK}Max = \frac{1}{SPI_0(orSPI_3) + t_{valid}}$$

t_{valid} is the slave time response to output data after detecting an SPCK edge. For Atmel SPI DataFlash (AT45DB642D), t_{valid} (or t_v) is 12 ns Max.

In the formula above, f_{SPCK}Max = 40 MHz @ VDDIO = 3.3V.

Slave Read Mode

In Slave mode, SPCK is the input clock for the SPI. The max SPCK frequency is given by setup and hold timings SPI_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 the SPCK pad.

Slave Write Mode

$$f_{SPCK}Max = \frac{1}{2x(SPI_{6max}(orSPI_{9max}) + t_{setup})}$$

t_{setup} is the setup time from the master before sampling data.

45.4.3.2 SPI Timings

Table 45-8. SPI Timings

Symbol	Parameter	Min	Max	Unit
SPI_0	MISO setup time before SPCK rises (master)	15.3	—	ns
SPI_1	MISO hold time after SPCK rises (master)	-3.9	—	
SPI_2	SPCK rising to MOSI delay (master)	-5.5	1.5	
SPI_3	MISO setup time before SPCK falls (master)	21.7	—	
SPI_4	MISO hold time after SPCK falls (master)	-8.1	—	
SPI_5	SPCK falling to MOSI delay (master)	-9.9	-4.2	
SPI_6	SPCK falling to MISO delay (slave)	3.9	13.8	
SPI_7	MOSI setup time before SPCK rises (slave)	0.4	—	
SPI_8	MOSI hold time after SPCK rises (slave)	3.9	—	
SPI_9	SPCK rising to MISO delay (slave)	4.0	14.4	
SPI_{10}	MOSI setup time before SPCK falls (slave)	1.1	—	
SPI_{11}	MOSI hold time after SPCK falls (slave)	2.8	—	
SPI_{12}	NPCS setup to SPCK rising (slave)	3.8	—	
SPI_{13}	NPCS hold after SPCK falling (slave)	-29.9	—	
SPI_{14}	NPCS setup to SPCK falling (slave)	4.5	—	
SPI_{15}	NPCS hold after SPCK falling (slave)	-28.7	—	

Note: V_{VDDIO} from 3.0V to 3.6V, maximum external capacitor = 10 pF.

Note that in SPI Master mode, the device does not sample the data (MISO) on the opposite edge where data clocks out (MOSI) but the same edge is used as shown in Figure 45-1 and Figure 45-2.

45.4.4 USART in SPI Mode Timings

Timings are given in the following domain:

- V_{VDDIO} from 3.0V to 3.6V, maximum external capacitor = 10 pF.

Figure 45-5. USART SPI Master Mode

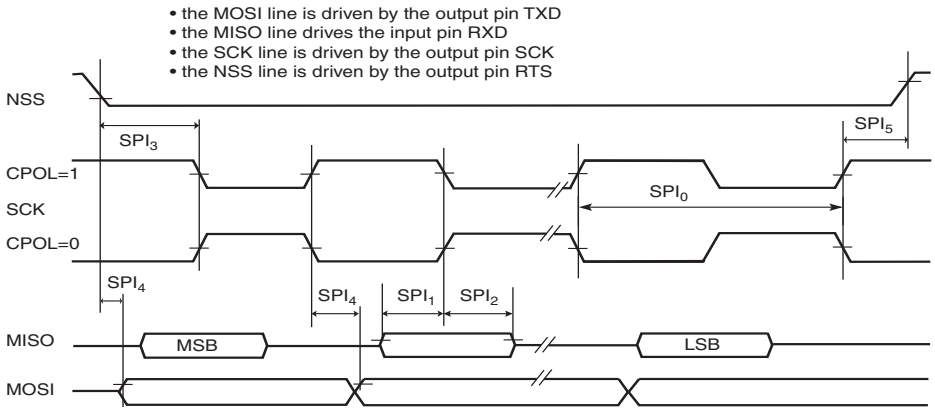


Figure 45-6. USART SPI Slave Mode: (Mode 1 or 2)

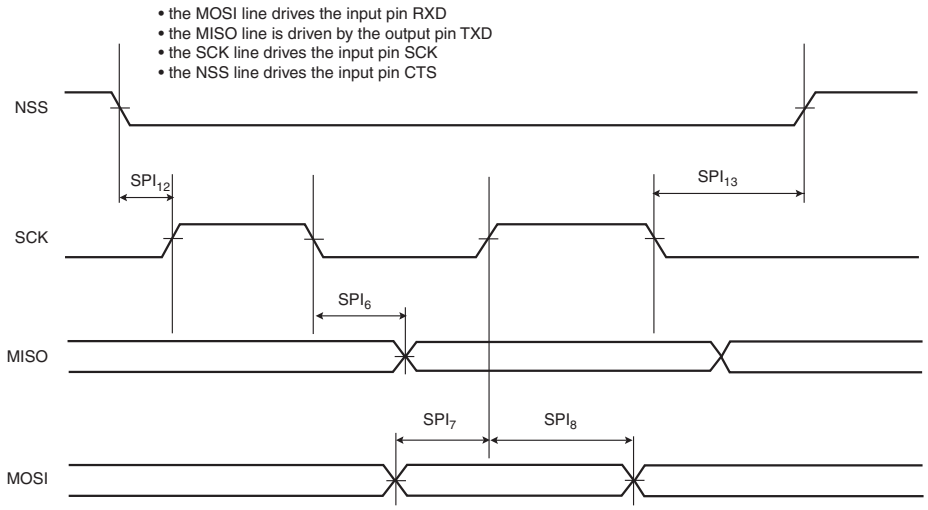
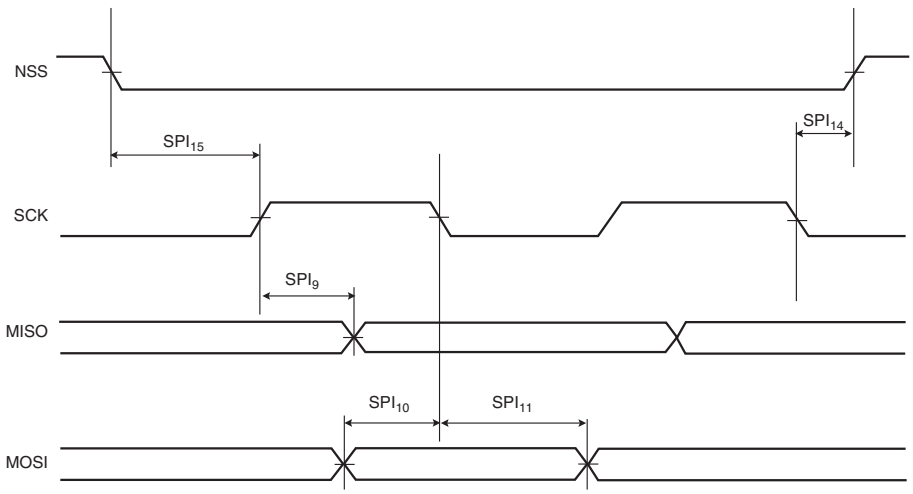


Figure 45-7. USART SPI Slave Mode: (Mode 0 or 3)



45.4.4.1 USART SPI Timings

Table 45-9. USART SPI Timings

Symbol	Parameter	Min	Max	Unit
Master Mode				
SPI_0	SCK period	6 / MCK	—	ns
SPI_1	Input data setup time	0.5 * MCK + 1.1	—	
SPI_2	Input data hold time	1.5 * MCK + 4.8	—	
SPI_3	Chip select active to serial clock	1.5 * SPCK + 0.9	—	
SPI_4	Output data setup time	- 6.7	7.1	
SPI_5	Serial clock to chip select inactive	1 * SPCK - 6.0	—	
Slave Mode				
SPI_6	SCK falling to MISO	6.8	20.7	ns
SPI_7	MOSI setup time before SCK rises	2 * MCK + 0.2	—	
SPI_8	MOSI hold time after SCK rises	4.2	—	
SPI_9	SCK rising to MISO	8.1	19.8	
SPI_{10}	MOSI setup time before SCK falls	2 * MCK + 1	—	
SPI_{11}	MOSI hold time after SCK falls	5.2	—	
SPI_{12}	NPCS0 setup to SCK rising	2.5 * MCK - 0.4	—	
SPI_{13}	NPCS0 hold after SCK falling	1.5 * MCK + 5.5	—	
SPI_{14}	NPCS0 setup to SCK falling	2.5 * MCK + 0.2	—	
SPI_{15}	NPCS0 hold after SCK rising	1.5 * MCK + 4.5	—	

45.5 Embedded Analog Peripherals Characteristics

45.5.1 Core Voltage Regulator

Table 45-10. Core Voltage Regulator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{VDDIN}	Supply Voltage Range (VDDIN)		2.5	3.3	3.6	V
V_{VDDOUT}	DC Output Voltage	Normal Mode Standby Mode	—	1.2 0	—	V
I_{LOAD}	Maximum DC Output Current	V_{VDDIN} > 2.5V T_J= 100ºC	—	—	120	mA
ACC	Output Voltage Total Accuracy	I_{LOAD} = 0.8 mA to 120 mA V_{VDDIN} = 2.5V to 3.6V T_J= [-40ºC to 100ºC]	-5	—	5	%
I_{INRUSH}	Inrush current	I_{LOAD} = 0. See note(1).	—	—	400	mA
I_{VDDIN}	Current Consumption (VDDIN)	Normal Mode; I_{Load} = 0 mA Normal Mode; I_{Load} = 120 mA Standby Mode;	—	5 500 0.02	 1	µA
C_{IN}	Input Decoupling Capacitor(2)		1	—	—	µF
C_{OUT}	Output Capacitor(3)	Capacitance ESR	0.7 0.01	2.2	10 10	µF Ω
t_{ON}	Turn on time	C_{OUT}= 2.2 µF, V_{VDDOUT} reaches 1.2V (+/- 3%)	—	500	—	µs
t_{OFF}	Turn off time	C_{OUT}= 2.2 µF	—	—	40	ms

Notes:
1. Current needed to charge the external bypass/decoupling capacitor network.
2. A ceramic capacitor must be connected between VDDIN and the closest GND pin of the device. This decoupling capacitor is mandatory to reduce inrush current and to improve transient response and noise rejection.
3. To ensure stability, an external output capacitor, C_{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 of 0.01 to 10 Ω.
   Solid tantalum, and multilayer ceramic capacitors are all suitable as output capacitors. An additional 100 nF bypass capacitor between VDDOUT and the closest GND pin of the device helps decrease output noise and improves the load transient response.

45.5.2 PLC DC Characteristics

Table 45-11. PLC DC Characteristics

Parameter	Condition	Symbol	Rating			Unit
			Min	Typ	Max	
Supply Voltage		VDDIO	3.00	3.30	3.60	V
H-level Input Voltage (3.3V CMOS)		VIH	2.0	-	VDDIO+0.3	
L-level Input Voltage (3.3V CMOS)		VIL	-0.3	-	0.8	
H-level Output Voltage	3.3V I/O IOH = -100 µA	VOH	VDDIO-0.2	-	VDDIO	
L-level Output Voltage	3.3V I/O IOL = 100 µA	VOL	0	-	0.2	
H-level Output V - I Characteristics	3.3V I/O VDDIO=3.3±0.3	IOH	*See "V-I curves" section*			mA
L-level Output V - I Characteristics	3.3V I/O VDDIO=3.3±0.3	IOL	*See "V-I curves" section*			
Internal Pull-up Resistor	3.3V I/O	Rpu	15	33	70	kΩ
Internal Pull-down Resistor	3.3V I/O	Rpd	15	33	70	

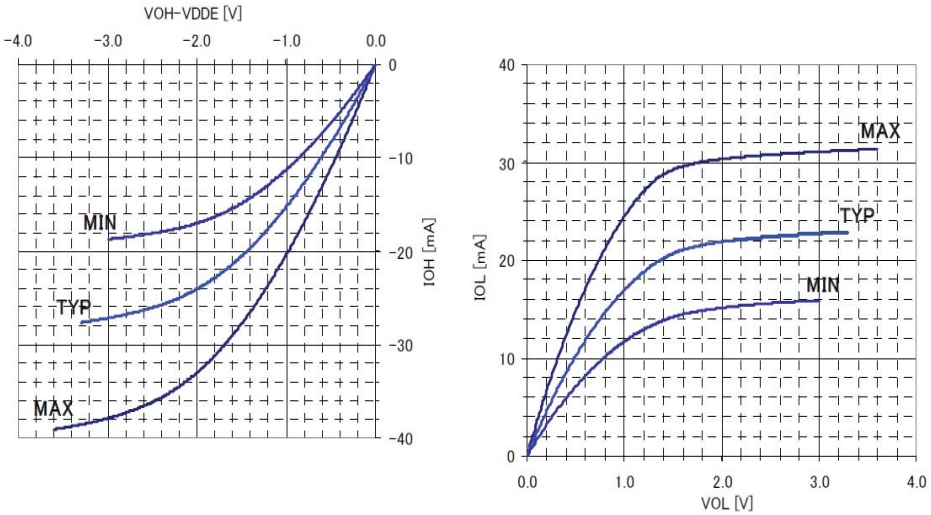
45.5.2.1 V-I curves

V-I Characteristics 3.3V standard CMOS IO L, M type

Apply to pins INTEST4, AGC0, AGC3

Condition:	MIN	Process = Slow	T_J = 125°C	VDDIO = 3.0V
	TYP	Process = Typical	T_J = 25°C	VDDIO = 3.3V
	MAX	Process = Fast	T_J = -40°C	VDDIO = 3.6V

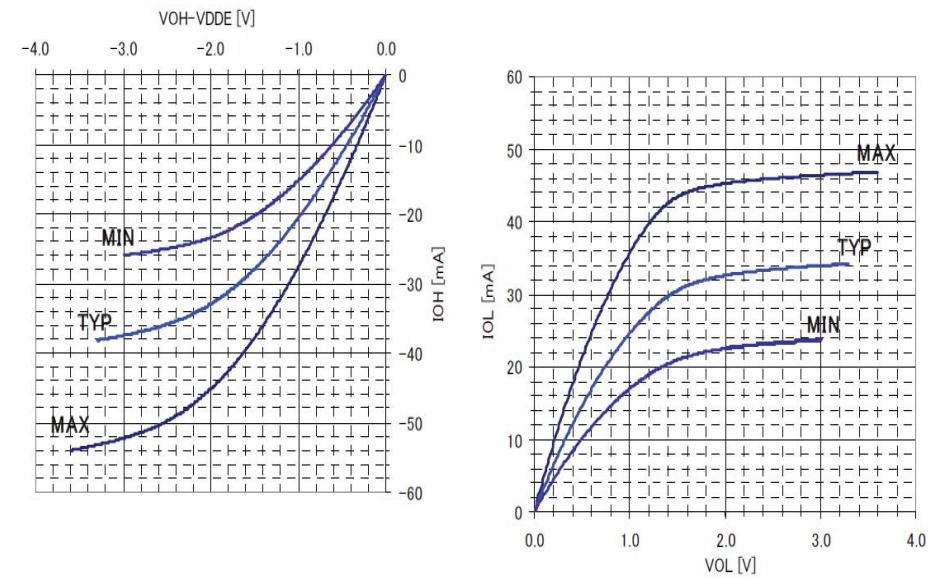
Figure 45-8. V-I curves for pins INTEST4, AGC0, AGC3



Apply to pins INTEST3, AGC1, AGC4

Condition:	MIN	Process = Slow	T_J = 125°C	VDDIO = 3.0V
	TYP	Process = Typical	T_J = 25°C	VDDIO = 3.3V
	MAX	Process = Fast	T_J = -40°C	VDDIO = 3.6V

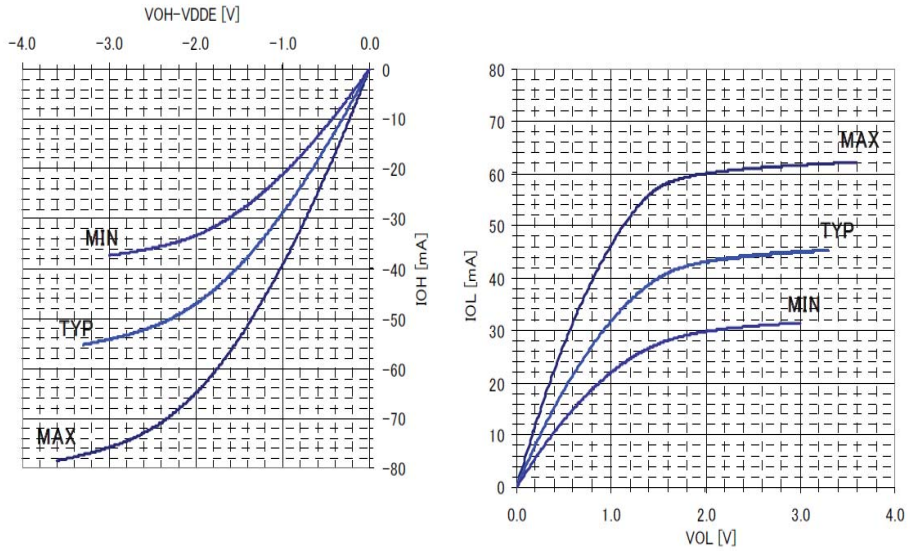
Figure 45-9. V-I curves for pins INTEST3, AGC1, AGC4



Apply to pins CLKOUT, TXRX0, TXRX1

Condition:	MIN	Process = Slow	T_J = 125°C	VDDIO = 3.0V
	TYP	Process = Typical	T_J = 25°C	VDDIO = 3.3V
	MAX	Process = Fast	T_J = -40°C	VDDIO = 3.6V

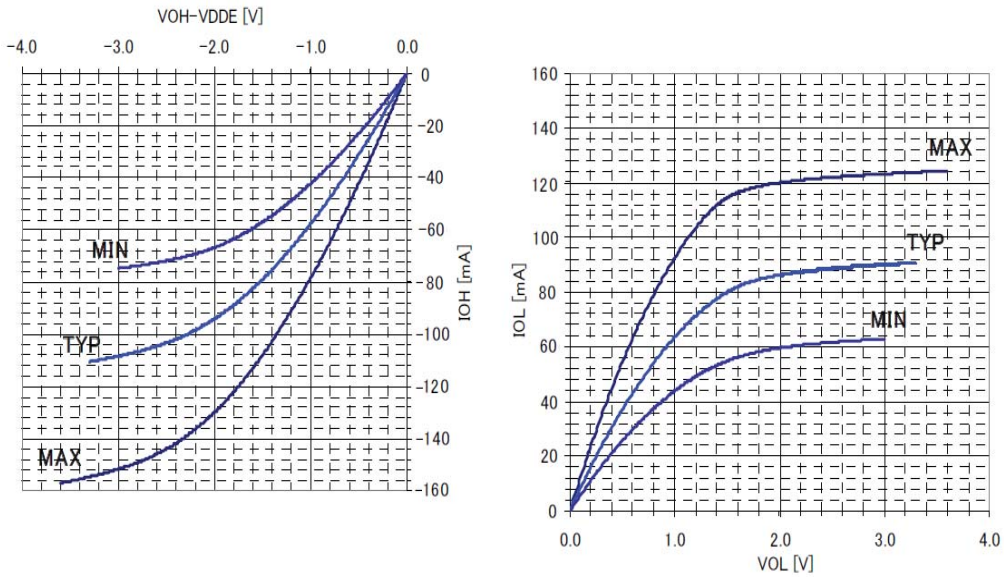
Figure 45-10. V-I curves for pins CLKOUT, TXRX0, TXRX1



Apply to pins EMIT [0:11], AGC2, AGC5

Condition:	MIN	Process = Slow	T_J = 125°C	VDDIO = 3.0V
	TYP	Process = Typical	T_J = 25°C	VDDIO = 3.3V
	MAX	Process = Fast	T_J = -40°C	VDDIO = 3.6V

Figure 45-11. V-I curves for pins EMIT [0:11], AGC2, AGC5



45.5.3 Automatic Power Switch

Table 45-12. Automatic Power Switch Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IT+}	Positive-going input threshold voltage (VDDIO)		1.9	—	2.2	V
V_{IT-}	Negative-going input threshold voltage (VDDIO)		1.8	—	2.1	V
V_{IT_HYST}	Threshold Hysteresis		—	100	—	mV

45.5.4 LCD Voltage Regulator and LCD Output Buffers

The LCD Voltage Regulator is a complete solution to drive an LCD display. It integrates a low-power LDO regulator with programmable output voltage and buffers to drive the LCD lines. A 1 µF capacitor is required at the LDO regulator output (VDDLCD). This regulator can be set in Active (Normal) mode, in Bypass mode (HiZ mode), or in OFF mode.

- In Normal mode, the VDDLCD LDO regulator output can be selected from 2.4V to 3.5V using LCDVROUT bits in the Supply Controller Mode Register (SUPC_MR), with the conditions:
  - VDDLCD ≤ VDDIO and,
  - VDDLCD ≤ VDDIN - 150 mV.
- In Bypass mode (HiZ mode), the VDDLCD is set in high impedance (through the LCDMODE bits in SUPC_MR register), and can be forced externally. This mode can be used to save the LDO operating current (4 µA).
- In OFF mode, the VDDLCD output is pulled down.

IMPORTANT: When using an external or the internal voltage regulator, VDDIO and VDDIN must be still supplied with the conditions: 2.4V ≤ VDDLCD ≤ VDDIO/VDDIN and VDDIO/VDDIN ≥ 2.5V.

Table 45-13. LCD Voltage Regulator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{VDDIN}	Supply Voltage Range (VDDIN)		2.5	—	3.6	V
V_{VDDLCD}	Programmable Output Range	See Table 45-14.	2.4	—	3.6	V
	Output voltage Accuracy		-10	—	+10	%
I_{VDDIN}	Current Consumption (VDDIN)	LDO Enabled	—	—	4	µA
d_{VOUT} / d_{VDDIN}	VDDLCD variation with VDDIN		—	-50	-70	mV/V
I_{LOAD}	Output Current	DC or transient load averaged by the external decoupling capacitor	—	—	2	mA
C_{OUT}	Output Capacitor on VDDLCD		1	—	10	µF
t_{ON}	Start-up Time	C_{OUT} = 1 µF	—	—	1	ms

Table 45-14. VDDLCD Voltage Selection at VDDIN = 3.6V

LCDVROUT	VDDLCD (V)	LCDVROUT	VDDLCD (V)	LCDVROUT	VDDLCD (V)	LCDVROUT	VDDLCD (V)
0	2.86	4	2.57	8	3.45	12	3.16
1	2.79	5	2.50	9	3.38	13	3.09
2	2.72	6	2.43	10	3.31	14	3.02
3	2.64	7	2.36	11	3.23	15	2.95

Table 45-15. LCD Buffers Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
I_{VDDIN}	Current Consumption (VDDIN)	LDO enabled	—	25	35	µA
Z_{OUT}	Buffer Output Impedance	GPIO in LCD mode (SEG or COM)	200	500	1500	Ω
C_{LOAD}	Capacitive Output Load		10p	—	50n	F
t_R / t_F	Rising or falling time 95% convergence	C_{LOAD} = 10 pF C_{LOAD} = 50 nF	—	—	3 225	µs

45.5.5 VDDCORE Brownout Detector

Table 45-16. Core Power Supply Brownout Detector Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IT-}	Negative-going input threshold voltage (VDDCORE) (1)		0.98	1.0	1.04	V
V_{IT+}	Positive-going input threshold voltage (VDDCORE)		0.80	1.0	1.08	V
V_{HYST}	Hysteresis Voltage	V_{IT+} - V_{IT-}	—	25	50	mV
t_{d-}	V_{IT-} detection propagation time	VDDCORE = V_{IT+} to (V_{IT-} - 100mV)	—	200	300	ns
t_{ON}	Start-up Time	From disabled state to enabled state	—	—	300	µs
$I_{VDDCORE}$	Current Consumption (VDDCORE)	Brownout detector enabled	—	—	15	µA
I_{VDDIO}	Current Consumption (VDDIO)	Brownout detector enabled	—	—	18	µA

Notes: 1. The product is guaranteed to be functional at V_{IT-}

Figure 45-12. Core Brownout Output Waveform

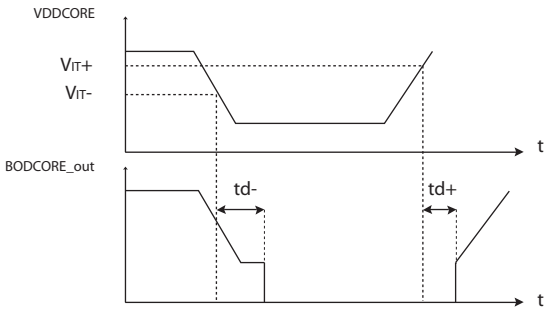
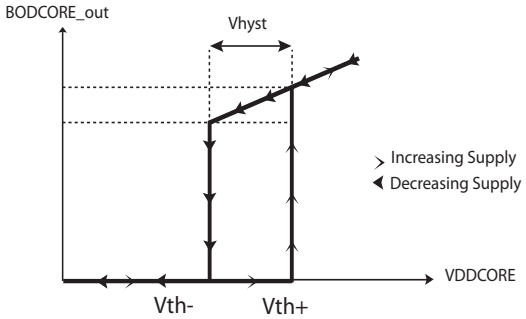


Figure 45-13. Core Brownout Transfer Characteristics



45.5.6 VDDCORE Power-On-Reset

Table 45-17. Core Power Supply Power-On-Reset Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IT-}	Negative-going input threshold voltage (VDDCORE)		0.71	0.9	1.02	V
V_{IT+}	Positive-going input threshold voltage (VDDCORE)		0.80	1.0	1.08	V
V_{HYST}	Hysteresis Voltage	V_{IT+} - V_{IT-}	—	60	110	mV
t_{d-}	V_{IT-} detection propagation time	VDDCORE = V_{IT+} to (V_{IT-} - 100mV)	—	—	15	µs
t_{ON}	Start-up Time	VDDCORE rising from 0 to final value. Time to release reset signal.	—	—	300	µs
$I_{VDDCORE}$	Current Consumption (VDDCORE)		—	—	6	µA
I_{VDDIO}	Current Consumption (VDDIO)		—	—	9	µA

45.5.7 VDDIO Supply Monitor

Table 45-18. VDDIO Supply Monitor

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{TH-}	Programmable range of negative-going input threshold voltage (VDDIO)	4 selectable steps	3.0	—	3.4	V
ACC	V_{TH-} Accuracy	With respect to programmed value	-2.5	—	+2.5	%
V_{HYST}	Hysteresis(2)		—	30	40	mV
I_{DDON}	Current Consumption (VDDIO)(1)	On with a 100% duty cycle	—	20	40	µA
t_{ON}	Start-up Time	From OFF to ON	—	—	300	µs

Notes: 1. The average current consumption can be reduced by using the supply monitor in Sampling mode. See the Supply Controller section.

2. V_{HYST} = V_{TH+} - V_{TH-}. V_{TH+} is the positive-going input threshold voltage (VDDIO).

Table 45-19. VDDIO Supply Monitor V_{TH-} Threshold Selection

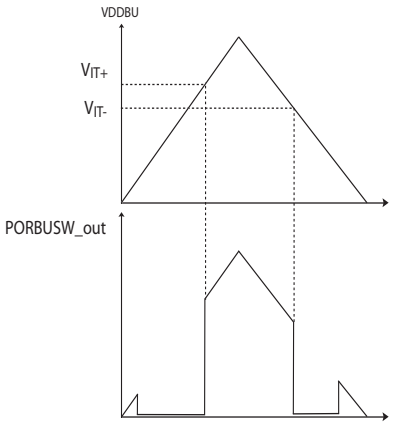
Digital Code	Threshold typ (V)
1100	3.04
1101	3.16
1110	3.28
1111	3.40

45.5.8 VDDBU Power-On-Reset

Table 45-20. Zero-Power-On POR (Backup POR) Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{TH+}	Positive-going input threshold voltage (VDDBU)	At Startup	1.45	1.53	1.59	V
V_{TH-}	Negative-going input threshold voltage (VDDBU)		1.35	1.45	1.55	V
I_{VDDBU}	Current Consumption	Enabled	—	300	700	nA
t_{RES}	Reset Time-out Period		100	240	500	µs

Figure 45-14. Zero-Power-On Reset Characteristics



45.5.9 VDDIO Power-On-Reset

Table 45-21. Zero-Power-On POR (VDDIO POR) Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{TH+}	Positive-going input threshold voltage (VDDIO)	At Startup	1.45	1.53	1.59	V
V_{TH-}	Negative-going input threshold voltage (VDDIO)		1.35	1.45	1.55	V
I_{VDDIO}	Current Consumption		—	300	700	nA
t_{RES}	Reset Time-out Period		100	240	500	µs

45.5.10 Oscillator Characteristics

45.5.10.1 32 kHz RC Oscillator

Table 45-22. 32 kHz RC Oscillator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{VDDBU}	Supply voltage range (VDDBU)		1.62	3.3	3.6	V
f_0	Frequency initial accuracy	VDDBU = 3.3V, T_A = 27°C	26	32	39	kHz
dF/dV	Frequency drift with VDDBU	VDDBU from 1.6V to 3.6V	0.5	1.5	2.5	% / V
dF/dT	Frequency drift with temperature	T_A = [-40°C to +27°C]	—	+8	+17	%
		T_A = [27°C to 85°C]	—	+6	+13	
Duty	Duty cycle		48	50	52	%
t_{ON}	Start-up time		—	—	100	µs
I_{DDON}	Current consumption (VDDBU)		—	150	300	nA

45.5.10.2 4/8/12 MHz RC Oscillators

Table 45-23. 4/8/12 MHz RC Oscillators Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$V_{VDDCORE}$	Supply voltage range (VDDCORE)		1.08	1.2	1.32	V
f_{Range}	Output frequency range(1)		4	—	12	MHz
ACC_4	4 MHz range; total accuracy	-40°C < T_A < +85°C	—	—	± 30	%
ACC_8	8 MHz range; total accuracy	VDDCORE from 1.08V to 1.32V, T_A = 25°C 0°C < T_A < +70°C -40°C < T_A < +85°C	—	—	± 1.0 ± 3.0 ± 5.0	%
ACC_{12}	12 MHz range; total accuracy	VDDCORE from 1.08V to 1.32V, T_A = 25°C 0°C < T_A < +70°C -40°C < T_A < +85°C	—	—	± 1.0 ± 3.0 ± 5.0	%
f_{STEP}	Frequency trimming step size	8 MHz 12 MHz	—	47 64	—	kHz
Duty	Duty cycle		45	50	55	%
t_{ON}	Start-up time, MOSCRCEN from 0 to 1		—	—	10	µs
t_{STAB}	Stabilization time on RC Frequency change (MOSCRCF)		—	—	5	µs
I_{DDON}	Active Current Consumption (VDDCORE)	4 MHz 8 MHz 12 MHz	—	50 65 82	68 86 102	µA

Notes: 1. The frequency range can be configured in the PMC Clock Generator.

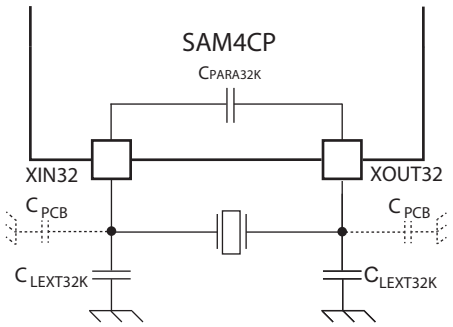
45.5.10.3 32.768 kHz Crystal Oscillator

Table 45-24. 32.768 kHz Crystal Oscillator Characteristics

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
V_{VDDBU}	Supply voltage range (VDDBU)			1.62	3.3	3.6	V
f_{REQ}	Operating frequency	Normal mode with crystal		—	—	32.768	kHz
Duty	Duty cycle			40	50	60	%
t_{ON}	Start-up time	$Rs^{(1)}$ < 50 kΩ	$C_{CRYSTAL}$ = 12.5pF $C_{CRYSTAL}$ = 6pF	—	—	900 300	ms
		$Rs^{(1)}$ < 100 kΩ	$C_{CRYSTAL}$ = 12.5pF $C_{CRYSTAL}$ = 6pF			1200 500	
I_{DDON}	Current consumption (VDDBU)	$Rs^{(1)}$ < 65 kΩ	$C_{CRYSTAL}$ = 12.5pF $C_{CRYSTAL}$ = 6pF	—	450 280	950 850	nA
		$Rs^{(1)}$ < 100 kΩ	$C_{CRYSTAL}$ = 6pF		350	1050	
		$Rs^{(1)}$ < 20 kΩ	$C_{CRYSTAL}$ = 6pF		220	—	
P_{ON}	Drive level			—	—	0.1	µW
R_F	Internal resistor	Between XIN32 and XOUT32		—	10	—	MΩ
$C_{CRYSTAL}$	Allowed crystal capacitive load	From crystal specification		6	—	12.5	pF
$C_{LEXT32K}$	External capacitor on XIN32 and XOUT32			—	—	24	pF
$C_{PARA32K}$	Internal parasitic capacitance	Between XIN32 and XOUT32		0.6	0.7	0.8	pF

Notes: 1. R_S is the series resistor.

Figure 45-15. 32.768 kHz Crystal Oscillator Schematic



$$C_{LEXT32K} = 2 \times (C_{CRYSTAL} - C_{PARA32K} - C_{PCB} / 2)$$

where C_{PCB} is the ground referenced parasitic capacitance of the printed circuit board (PCB) on XIN32 and XOUT32 tracks. As an example, if the crystal is specified for a 12.5 pF load, with C_{PCB} = 1 pF (on XIN32 and on XOUT32), $C_{LEXT32K}$ = 2 x (12.5 - 0.7 - 0.5) = 22.6 pF

Table 45-25 summarizes recommendations for 32.768 kHz crystal selection.

Table 45-25. Recommended 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

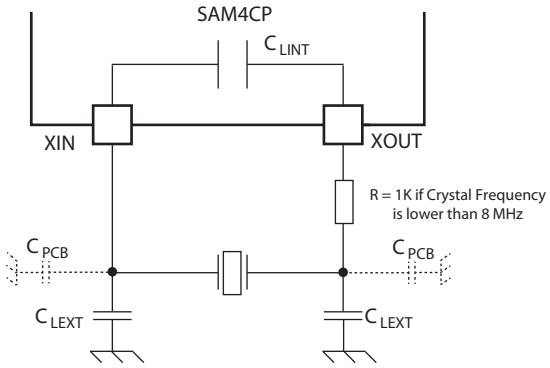
45.5.10.4 3 to 20 MHz Crystal Oscillator

Table 45-26. 3 to 20 MHz Crystal Oscillator Characteristics(1)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{VDDIO}	Supply Voltage Range (VDDIO)		3.0	3.3	3.6	V
V_{VDDPLL}	Supply Voltage Range (VDDPLL)		1.08	1.2	1.32	V
f_{OSC}	Operating Frequency range	Normal mode with crystal	3	16	20	MHz
Duty	Duty Cycle		40	50	60	%
t_{ON}	Start-up time	3 MHz, C_{SHUNT} = 3 pF 8 MHz, C_{SHUNT} = 7 pF 16 MHz, C_{SHUNT} = 7 pF with C_M = 8 fF 16 MHz, C_{SHUNT} = 7 pF with C_M = 1.6 fF 20 MHz, C_{SHUNT} = 7 pF	—	—	14.5 4 1.4 2.5 1	ms
I_{DDON}	Current consumption On VDDIO	3 MHz(2) 8 MHz(3) 16 MHz(4) 20 MHz(5)	—	230 300 390 450	350 400 470 560	µA
	On VDDPLL	3 MHz(2) 8 MHz(3) 16 MHz(4) 20 MHz(5)		6 12 20 24	7 14 23 30	
P_{ON}	Drive level	3 MHz 8 MHz 16 MHz, 20 MHz	—	—	15 30 50	µW
R_F	Internal resistor	Between XIN and XOUT	—	0.5	—	MΩ
$C_{CRYSTAL}$	Allowed crystal capacitive load	From crystal specification	12	—	18	pF
C_{LEXT}	External capacitor on XIN and XOUT		—	—	18	pF
C_{LINT}	Integrated load capacitance	Between XIN and XOUT	7.5	9.5	10.5	pF

Notes:
1. See "Crystal Oscillator Design Considerations" .
2. R_S = 100 - 200 Ω; C_S = 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_S = 2.0 - 2.5 pF; C_M = 4 - 3 fF(typ, worst case).
4. R_S = 25 - 50 Ω; C_S = 2.5 - 3.0 pF; C_M = 7 - 5 fF (typ, worst case).
5. R_S = 20 - 50 Ω; C_S = 3.2 - 4.0 pF; C_M = 10 - 8 fF(typ, worst case).

Figure 45-16. 3 to 20 MHz Crystal Oscillator Schematic



$$C_{LEXT} = 2 \times (C_{CRYSTAL} - C_{LINT} - C_{PCB} / 2)$$

where C_{PCB} is the ground referenced parasitic capacitance of the printed circuit board (PCB) on XIN and XOUT tracks. As an example, if the crystal is specified for an 18 pF load, with C_{PCB} = 1 pF (on XIN and on XOUT), C_{LEXT} = 2 x (18 - 9.5 - 0.5) = 16 pF

Table 45-27 summarizes recommendations to be followed when choosing a crystal.

Table 45-27. Recommend Crystal Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
ESR	Equivalent Series Resistor (Rs)	Fundamental @ 3 MHz Fundamental @ 8 MHz Fundamental @ 12 MHz Fundamental @ 16 MHz Fundamental @ 20 MHz	—	—	200 100 80 80 50	Ω
C_M	Motional capacitance		—	—	8	fF
C_{SHUNT}	Shunt capacitance		—	—	7	pF

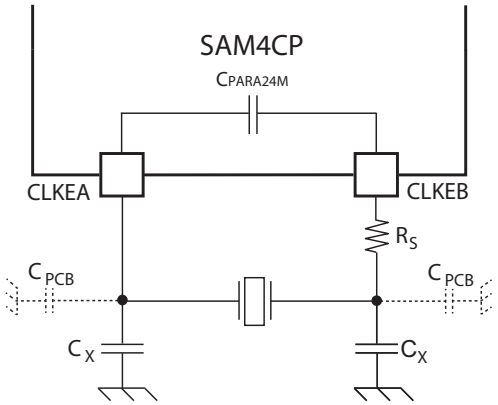
45.5.10.5 24 MHz CLKEA (PLC) Crystal Oscillator Characteristics

Table 45-28. PLC 24 MHz Crystal Oscillator Characteristics

Parameter	Test Condition	Symbol	Rating			Unit
			Min	Typ	Max	
Crystal Oscillator frequency	Fundamental	X_{tal}		24		MHz
External Oscillator Capacitance(2)(3)		C_{XTAL}	-	10	-	pF
External capacitor on CLKEA and CLKEB(2)(3)		C_X	-	17	-	
Internal parasitic capacitance	Between CLKEA and CLKEB	$C_{PARA24M}$	-	1	-	
H-level Input Voltage		XVIH	2	-	VDDIO +0.3	V
L-level Input Voltage		XVIL	-0.3	-	0.8	
External Oscillator Parallel Resistance		Rp		not needed		Ω
External Oscillator Series Resistance(2)		Rs	-	220	-	

Notes:
1. The crystal should be located as close as possible to CLKEB and CLKEA pins.
2. Recommended value for Cx is 17pF and Rs 220Ω. These values may depend on the specific crystal characteristics and PCB layout. See example below. For further information please refer to Atmel doc43084 “Crystal Selection Guidelines” application note.
3. As a requirement of G3 specification, the System Clock tolerance from which transmit frequency and symbol timing are derived shall be ± 25 ppm maximum. Crystal Stability/Tolerance/Ageing values must be selected according to standard G3 requirements.

Figure 45-17. 24 MHz Crystal Oscillator Schematic



$$C_X = 2 \times (C_{XTAL} - C_{PARA24M} - C_{PCB} / 2)$$

where C_{PCB} is the ground referenced parasitic capacitance of the printed circuit board (PCB) on CLKEA and CLKEB tracks.

As a practical example, taking the following crystal part number:

Manufacturer: TXC CORPORATION

Part Number: 7B-24.000MEEQ-T

Frequency: 24.000 MHz

Tolerance: 10 ppm (as low as possible to fullfil G3 specification requirements)

C_{XTAL} = 10 pF

Working in a typical layout / substrate with C_{PCB} = 1 pF

The value of the external capacitors on CLKEA and CLKEB should be C_X = 2 x (10 - 1 - 0.5) = 17 pF

It is strongly recommended to use capacitors with the lowest temperature stability possible. In this practical example, a suitable part number could be:

Manufacturer: MURATA

Part Number: GRM1885C1H150FA01D

Capacitance: 15 pF (commercial value close to 17 pF)

Tolerance: 1 %

Dielectric: C0G / NP0 (0 drift)

45.5.10.6 Crystal Oscillator Design Considerations

When choosing a crystal for the 32768 Hz Slow Clock Oscillator or for the 3 - 20 MHz Oscillator, several parameters must be taken into account. Important parameters are as follows:

- Main Oscillator Input

  CLKOUT signal is intended to be used as XIN (system main oscillator) input. When using an external crystal/oscillator as XIN input instead CLKOUT signal, CLKOUT should be switched off by software to minimize noise due to coupling in close pins.

- Crystal Load Capacitance

  The total capacitance loading the crystal, including the oscillator's internal parasitics and the PCB parasitics, must match the load capacitance for which the crystal's frequency is specified. Any mismatch in the load capacitance with respect to the crystal's specification will lead to inaccurate oscillation frequency.

- Crystal Drive Level

  Use only crystals with the specified drive levels greater than the specified MCU oscillator drive level. Applications that do not respect this criterion may damage the crystal.

- Crystal Equivalent Series Resistor (ESR)

  Use only crystals with the specified ESR lower than the specified MCU oscillator ESR. In applications where this criterion is not respected, the crystal oscillator may not start.

- Crystal Shunt Capacitance

  Use only crystal with the specified shunt capacitance lower than the specified MCU oscillator shunt capacitance. In applications where this criterion is not respected, the crystal oscillator may not start.

- PCB Layout Considerations

  To minimize inductive and capacitive parasitics associated with XIN, XOUT, XIN32, XOUT32, CLKEA, CLKEB nets, it is recommended to route them as short as possible. It is also of prime importance to keep those nets away from noisy switching signals (clock, data, PWM, etc...). A good practice is to shield them with a quiet ground net to avoid coupling to neighboring signals.

45.5.11 PLLA, PLLB Characteristics

Table 45-29. PLLA Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{VDDPLL}	Supply Voltage Range (VDDPLL)		1.08	1.2	1.32	V
f_{IN}	Input Frequency Range		30	32.768	34	kHz
f_{OUT}	Output Frequency Range		7.5	8.192	8.5	MHz
N_{RATIO}	Frequency Multiplying Ratio (MULA +1)		—	250	—	—
J_P	Period Jitter	Peak value	—	4	—	ns
t_{ON}	Start-up time	From OFF to output oscillations (Output frequency within 10% of target frequency)	—	—	250	µs
t_{LOCK}	Lock Time	From OFF to PLL locked	—	—	2.5	ms
I_{PLLON}	Active mode Current Consumption (VDDPLL)	f_{OUT} = 8.192 MHz	—	50	—	µA
I_{PLLOFF}	OFF mode Current Consumption (VDDPLL)	@25°C Over the temperature range	—	0.05 0.05	0.30 5	µA

Table 45-30. PLLB Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{VDDPLL}	Supply Voltage Range (VDDPLL)		1.08	1.2	1.32	V
f_{IN}	Input Frequency Range		3	—	32	MHz
f_{OUT}	Output Frequency Range		80	—	240	MHz
N_{RATIO}	Frequency Multiplying Ratio (MULB +1)		3	—	62	—
Q_{RATIO}	Frequency Dividing Ratio (DIVB)		2	—	24	—
t_{ON}	Start-up time		—	60	150	µs
I_{VDDPLL}	Current Consumption on VDDPLL	Active mode @ 80 MHz @1.2V Active mode @ 96 MHz @1.2V Active mode @ 160 MHz @1.2V Active mode @ 240 MHz @1.2V	—	0.94 1.2 2.1 3.34	1.2 1.5 2.5 4	mA

45.5.12 Temperature Sensor Characteristics

The temperature sensor provides an output voltage (V_T) that is proportional to absolute temperature (PTAT). This voltage can be measured through the channel number 7 of the 10-bit ADC. Improvement of the raw performance of the temperature sensor acquisition can be achieved by performing a single temperature point calibration to remove the initial inaccuracies (V_T and ADC offsets).

Table 45-31. Temperature Sensor Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{VDDIN}	Supply Voltage Range (VDDIN)		2.5	—	3.6	V
V_T	Output voltage	T_J = 27°C	1.34	1.44	1.54	V
dV_T / dT	Output Voltage sensitivity to Temperature		4.2	4.7	5.2	mV/°C
dV_T / dV	V_T variation with VDDIN	VDDIN from 2.5V to 3.6V	—	—	1	mV/V
t_S	V_T Settling time	When V_T is sampled by the 10-bit ADC, the required track time to ensure 1ºC accurate settling	—	—	1	µs
T_{ACC}	Temperature accuracy(1)	After offset calibration Over T_J range [-40°C to +85°C]	—	±5	±7	°C
		After offset calibration Over T_J range [0°C to +80°C]	—	±4	±6	°C
t_{ON}	Start-up time		—	5	10	µs
I_{VDDIN}	Current Consumption		50	70	80	µA

Note: 1. Does not include errors due to A/D conversion process.

45.5.13 Optical UART RX Transceiver Characteristics

Table 45-32 gives the description of the optical link transceiver for electrically isolated serial communication with hand-held equipment, such as calibrators compliant with standards ANSI-C12.18 or IEC62056-21 (only available on UART1).

Table 45-32. Transceiver Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{VDDIO}	Supply voltage range (VDDIO)		3	3.3	3.6	V
I_{DD}	Current consumption	ON OFF	—	25 —	35 0.1	µA
V_{TH}	Comparator threshold	According to the programmed threshold. See the OPT_CMPTH bit in the UART Mode Register (UART1)	-20	—	+20	mV
V_{HYST}	Hysteresis		10	20	40	mV
t_{PROP}	Propagation time	With 100 mVpp square wave input around threshold	—	—	5	µs
t_{ON}	Start-up time		—	—	100	µs

45.5.14 10-bit ADC Characteristics

Table 45-33. ADC Power Supply Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{VDDIN}	Supply voltage range (VDDIN)		2.5	3.3	3.6	V
I_{VDDIN}	Current consumption on VDDIN	ADC ON(1), internal voltage reference ON generating ADVREF = 3.0V	—	450	700	µA
		ADC ON(1), internal voltage reference OFF, ADVREF externally supplied	—	220	350	

Note: 1. Average current consumption performing conversion in free run mode @ 16 MHz ADC Clock. F_S = 510 kS/s.

Table 45-34. ADC Voltage Reference Input Characteristics (ADVREF pin)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{ADVREF}	ADVREF input voltage range (1)	Internal voltage reference OFF	2.4	—	V_{VDDIN}	V
R_{ADVREF}	ADVREF input resistance	ADC ON, internal voltage reference OFF	9	14	19	kΩ
I_{ADVREF}	Current consumption on ADVREF	ADVREF = 2.4V	-35%	170	+35%	µA
		ADVREF = 3.3V		235		
		ADVREF = 3.6V		260		
C_{ADVREF}	Decoupling capacitor on ADVREF		100	—	—	nF

Note: 1. ADVREF input range limited to VDDIO if VDDIO < VDDIN.

Table 45-35. ADC Timing Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{CK_ADC}	ADC clock frequency	3.0V ≤ VDDIN ≤ 3.6V	—	—	16	MHz
		2.5V ≤ VDDIN ≤ 3.0V			14	
t_{CONV}	ADC conversion time(1)	f_{CK_ADC} = 16 MHz, t_{TRACK} = 500 ns	1.95	—	—	µs
F_S	Sampling rate(2)	V_{VDDIN} > 3.0V, f_{CK_ADC} = 16 MHz	—	—	510	kS/s
		V_{VDDIN} > 2.5V, f_{CK_ADC} = 14 MHz			380	
t_{ON}	Start-up time	ADC only	—	—	40	µs
t_{TRACK}	Track and hold time(3)	2.5V ≤ VDDIN ≤ 3.0V	1000	—	—	ns
		3.0V < VDDIN < 3.6V	500			

Notes: 1. t_{CONV} = (TRACKTIM + 24) / f_{CK_ADC}.
2. F_S = 1 / t_{CONV}.
3. Refer to Section 45.5.14.1 “Track and Hold Time versus Source Output Impedance, Effective Sampling Rate”.

Table 45-36. ADC Analog Input Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
FSR	Analog Input Full Scale Range(1)		0	—	V_{ADVREF}	V
C_{IN}	Input capacitance(2)	Accounts for I/O input capacitance + ADC Sampling capacitor	—	—	10	pF

Notes: 1. If V_{VDDIO} < V_{ADVREF}, full scale range is limited to VDDIO.
2. See Figure 45-18 “Simplified Acquisition Path” .

Table 45-37. Static Performance Characteristics(1)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
R_{ADC}	Native ADC Resolution		—	10	—	Bits
R_{ADC_AV}	Resolution with Digital Averaging	See ”ADC Controller” section	10	—	12	Bits
INL	Integral Non Linearity	f_{CK_ADC} = 16 MHz	-2	—	+2	LSB
DNL	Differential Non Linearity		-1	—	+1	LSB
OE	Offset Error		-5	—	5	LSB
GE	Gain Error	Errors with respect to the best fit line method	-3	—	+3	LSB

Notes: 1. In this table, values expressed in LSB refer to the Native ADC resolution (i.e., a 10-bit LSB).

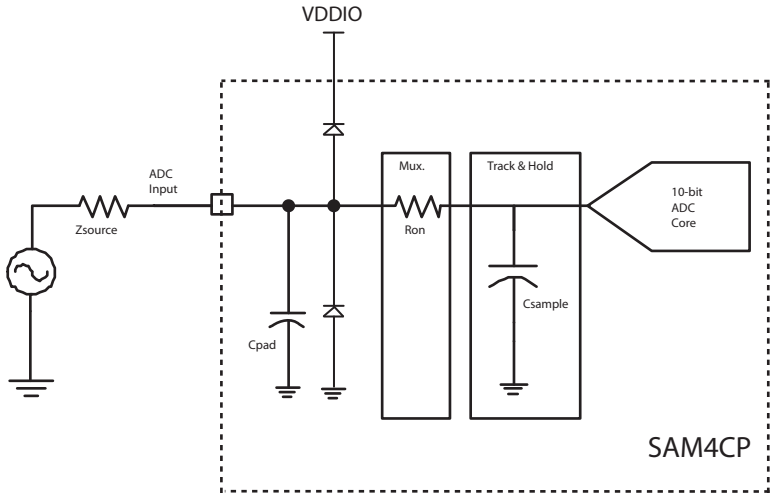
Table 45-38. Dynamic Performance Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
SNR	Signal to Noise Ratio	f_{CK_ADC} = 16 MHz V_{ADVREF} = V_{VDDIN} f_{IN} = 50 kHz V_{INPP} = 0.95 x V_{ADVREF}	57	60	—	dB
THD	Total Harmonic Distortion		—	-68	-55	dB
SINAD	Signal to Noise and Distortion		52	59	—	dB
ENOB	Effective Number of Bits		8.3	9.6	—	Bits

45.5.14.1 Track and Hold Time versus Source Output Impedance, Effective Sampling Rate

The following figure gives a simplified view of the acquisition path.

Figure 45-18. Simplified Acquisition Path



During its tracking phase, the 10-bit ADC charges its sampling capacitor through various serial resistors: source output resistor, multiplexer series resistor and the sampling switch series resistor. In case of high output source resistance (low power resistive divider, for example), the track time must be increased to ensure full settling of the sampling capacitor voltage. The following formulas give the minimum track time that guarantees a 10-bit accurate settling:

- $V_{VDDIN} > 3.0V$: t_{TRACK} (ns) = 0.12 x R_{SOURCE} (Ω)+ 500.
- $V_{VDDIN} \leq 3.0V$: t_{TRACK} (ns) = 0.12 x R_{SOURCE} (Ω)+ 1000.

According to the calculated track time (t_{TRACK}), the actual track time of the ADC must be adjusted through the TRACKTIM field in the ADC_MR register. TRACKTIM is obtained by the following formula:

$$TRACKTIM = floor\ (t_{TRACK} / t_{CK\_ADC})$$

with $t_{CK\_ADC} = 1 / f_{CK\_ADC}$ and floor (x) the mathematical function that rounds x to the greatest previous integer.

The actual conversion time of the converter is obtained by the following formula:

$$t_{CONV} = (TRACKTIM + 24) \times t_{CK\_ADC}$$

When converting in free run mode, the actual sampling rate of the converter is ($1/ t_{CONV}$) or as defined by the following formula:

$$F_S = f_{CK\_ADC} / (TRACKTIM + 24)$$

The maximum source resistance with the actual TRACKTIM setting is:

- R_{SOURCE_MAX} (Ω) = ((TRACKTIM + 1) x t_{CK_ADC} (ns) - 500) / 0.12 for V_{VDDIN} > 3.0V; or
- R_{SOURCE_MAX} (Ω) = ((TRACKTIM + 1) x t_{CK_ADC} (ns) - 1000) / 0.12 for $V_{VDDIN} \leq$ 3.0V.

Example: Calculated track time is lower than actual ADC clock period

- Assuming: f_{CK_ADC} = 1 MHz (t_{CK_ADC} = 1 µs), R_{SOURCE} = 100Ω and V_{VDDIN} = 3.3V.
- The minimum required track time is: t_{TRACK} = 0.12 X 100 + 500 = 512 ns.
- t_{TRACK} begin less than t_{CK_ADC}, TRACKTIM is set to 0. Actual track time is t_{CK_ADC} = 1 µs.
- The calculated sampling rate is: F_S = 1 MHz / 24 = 41.7 kHz.
- The maximum allowable source resistance is: R_{SOURCE_MAX} = (1000 - 500) / 0.12 = 4.1 kΩ.

Example: Calculated track time is greater than actual ADC clock period

- Assuming: f_{CK_ADC} = 16 MHz (t_{CK_ADC} = 62.5 ns), R_{SOURCE} = 600Ω and V_{VDDIN} = 2.8V.
- The minimum required track time is: t_{TRACK} = 0.12 X 600 + 1000 = 1072 ns.
- TRACKTIM = floor (1072/62.5) = 17. Actual track time is: (17 + 1) x t_{CK_ADC} = 1.125 µs.
- The calculated sampling rate is: F_S = 16 MHz / (24 + 17) = 390.2 kHz.
- The maximum allowable source resistance is: R_{SOURCE_MAX} = (1125 - 1000) / 0.12 = 1.04 kΩ.

45.5.15 Programmable Voltage Reference Characteristics

SAM4CP embeds a programmable voltage reference designed to drive the 10-bit ADC ADVREF input. Table 45-39 shows the electrical characteristics of this internal voltage reference. If necessary, this voltage reference can be bypassed with some level of configurability: the user can either choose to feed the ADVREF input with an external voltage source or with the VDDIO internal power rail. See programming details in the ADC Analog Control Register (ADC_ACR) in Section Analog-to-Digital Converter (ADC) in this datasheet.

Table 45-39. Programmable Voltage Reference Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{VDDIN}	Voltage Reference Supply Range		2.5	—	3.6	V
V_{ADVREF}	Programmable Output Range	See Table 45-40 $V_{VDDIN} > V_{ADVREF} + 100mV$	1.6	—	3.4	V
ACC	Reference Voltage Accuracy	With respect to the programmed value V_{VDDIN} = 3.3V; T_J = 25ºC	-3	—	3	%
T_C	Temperature coefficient	Box method(1)	—	—	250	ppm/ºC
t_{ON}	Start-up time	V_{VDDIN} = 2.5V V_{VDDIN} = 3.0V V_{VDDIN} = 3.6V	—	—	100 70 40	µs
Z_{LOAD}	Load Impedance	Resistive	4	—	—	kΩ
		Capacitive	0.1	—	1	µF
I_{VDDIN}	Current Consumption on VDDIN(2)	ADC is OFF	—	20	30	µA

Notes: 1. T_C = (max (V_{ADVREF}) - min (V_{ADVREF})) / ((T_{MAX} - T_{MIN}) * V_{ADVREF}(25ºC)).
2. Does not include the current consumed by the ADC ADVREF input if ADC is ON.

Table 45-40. Programmable Voltage Reference Selection Values

Sel. Value(1)	ADVREF	Notes
0	2.40	Default value
1	2.28	—
2	2.16	—
3	2.04	—
4	1.92	—
5	1.80	—
6	1.68	—
7	1.55	Min value
8	3.38	Max value
9	3.25	—
A	3.13	—
B	3.01	—
C	2.89	—
D	2.77	—
E	2.65	—
F	2.53	—

Notes: 1. Voltage reference values are configurable in ADC_ACR.IRVS.

45.6 Embedded Flash Characteristics

45.6.1 Embedded Flash DC Characteristics

Table 45-41. 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 Maximum Read Frequency onto VDDIO = 3.3V @ 25°C Random 64-bit Read: Maximum Read Frequency onto VDDCORE = 1.2V @ 25°C Maximum Read Frequency onto VDDIO = 3.3V @ 25°C	 16 3 10 3	 25 5 18 5	mA
		Program: - Onto VDDCORE = 1.2V @ 25°C - Onto VDDIO = 3.3V @ 25°C	 3 10	 5 15	mA
		Erase: - Onto VDDCORE = 1.2V @ 25°C - Onto VDDIO = 3.3V @ 25°C	 3 10	 5 15	mA

45.6.2 Embedded Flash AC Characteristics

Table 45-42.AC Flash Characteristics

Parameter	Conditions	Min	Typ	Max	Unit
Program/ Erase Operation Cycle Time	Write page (512 bytes)	—	1.5	3	ms
	Erase page	—	10	50	ms
	Erase block (4 Kbytes)	—	50	200	ms
	Erase sector	—	400	950	ms
	Full chip erase (1 MByte)	—	9	18	s
	Lock/Unlock time per region	—	1.5	3	ms
Data Retention	Not powered or powered	—	20	—	years
Endurance	Write/Erase cycles per page, block or sector @ 85°C	10K	—	—	cycles

45.6.2.1 Flash Wait States and Operating Frequency

The maximum operating frequency given in Table 45-43 below is limited by the Embedded Flash access time when the processor is fetching code out of it. The table gives the device maximum operating frequency depending on the FWS field of the EFC_FMR register. This field defines the number of wait states required to access the Embedded Flash Memory.

Table 45-43. Flash Wait State Versus Operating Frequency

FWS (Flash Wait State)	Maximum Operating Frequency (MHz) @ T° = 85C			
	VDDCORE = 1.08V	VDDIO = 3.0V to 3.6V	VDDCORE = 1.2V	VDDIO = 3.0V to 3.6V
0	16		17	
1	33		35	
2	51		52	
3	67		70	
4	85		87	
5	100		105	
6	—		121	

45.7 Power Supply Current Consumption

This section provides information about the current consumption on different power supply rails of the device. It gives current consumption in low-power modes (Backup mode, Wait mode, Sleep mode) and in Active mode (the application running from memory, by peripheral). All the consumption values in sections 45.7.1 to 45.7.4 are measured with PLC transceiver disabled (see Table 45-54 in Section 45.7.5 for PLC consumption values).

45.7.1 Backup Mode Current Consumption

Backup mode configurations and measurements are defined as follows:

- Configuration A is used to achieve the lowest possible current consumption.
- Configurations B, C and D are typical use cases with crystal oscillator, LCD and anti-tamper pins enabled.

Reminder: In Backup mode, the core voltage regulator is off and thus all the digital functions powered by VDDCORE are off.

45.7.1.1 Backup Mode Configuration A: Embedded Slow Clock RC Oscillator Enabled

- POR backup on VDDBU is disabled.
- RTC running.
- RTT enabled on 1 Hz mode.
- Force wake-up (FWUP) enabled.
- PLC in shutdown.
- Current measurement as per Figure 45-19.

45.7.1.2 Backup Mode Configuration B: 32.768 kHz Crystal Oscillator Enabled

- POR backup on VDDBU is disabled.
- RTC running.
- RTT enabled on 1 Hz mode.
- Force wake-up (FWUP) enabled.
- Anti-tamper input TMP0 enabled.
- PLC in shutdown.
- Current measurement as per Figure 45-19.

Figure 45-19. Measurement Setup for Configurations A and B

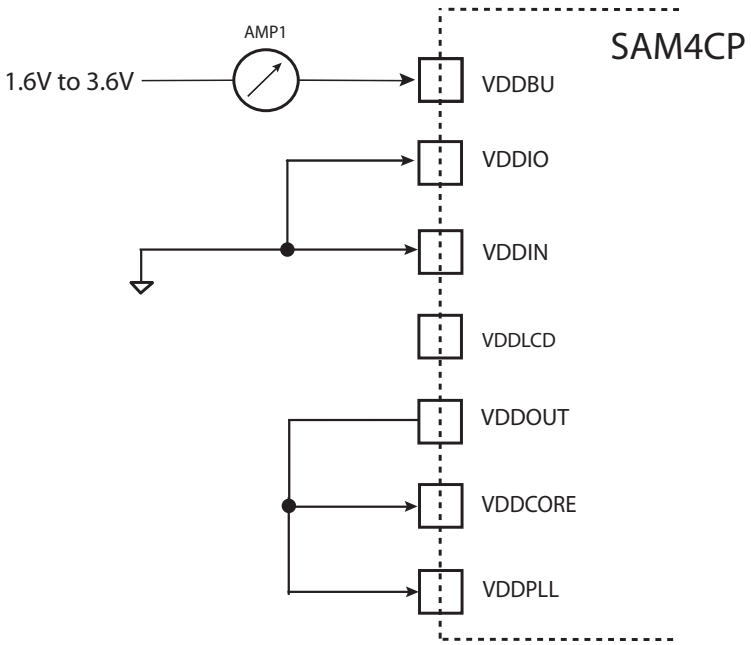
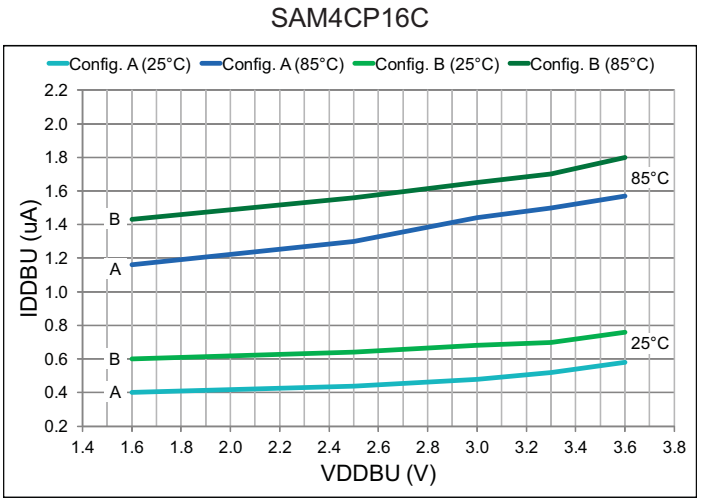


Table 45-44. Typical Current Consumption Values for Backup Mode Configurations A and B

Conditions	Configuration A	Configuration B	Unit
VDDBU = 3.6V @25°C	580	760	nA
VDDBU = 3.3V @25°C	520	700	
VDDBU = 3.0V @25°C	480	680	
VDDBU = 2.5V @25°C	440	640	
VDDBU = 1.6V @25°C	400	600	
VDDBU = 3.6V @85°C	1.57	1.80	µA
VDDBU = 3.3V @85°C	1.50	1.70	
VDDBU = 3.0V @85°C	1.44	1.65	
VDDBU = 2.5V @85°C	1.30	1.56	
VDDBU = 1.6V @85°C	1.16	1.43	

Figure 45-20. Typical Current Consumption in Backup Mode for Configurations A and B



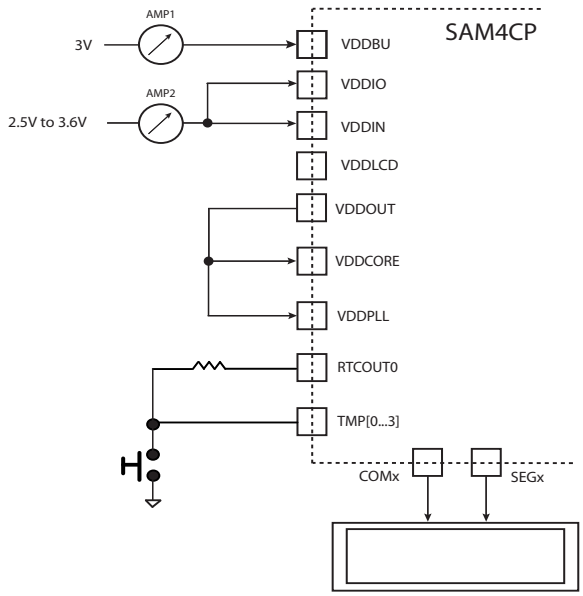
45.7.1.3 Backup Mode Configuration C: 32.768 kHz Crystal Oscillator Enabled

- POR backup on VDDBU is disabled.
- RTC running.
- RTT enabled on 1 Hz mode.
- Force wake-up (FWUP) enabled.
- Anti-tamper input TMP0, TMP1, TMP2, TMP3 and RTCOUT0 enabled.
- Main crystal oscillator disabled.
- System IO lines PA30, PA31, PB[0...3] in GPIO Input Pull-up mode.
- PLC in shutdown.
- All other GPIO lines in default state (see PIO Multiplexing table).
- Current measurement as per Figure 45-21.

45.7.1.4 Backup Mode Configuration D: 32.768 kHz Crystal Oscillator and LCD Enabled

- POR backup on VDDBU is disabled.
- RTC running.
- RTT enabled on 1 Hz mode.
- LCD controller in Low power mode, static bias and x64 slow clock buffer on-time drive time.
- LCD voltage regulator used.
- Force wake-up (FWUP) enabled.
- Anti-tamper input TMP0, TMP1, TMP2, TMP3 and RTCOUT0 enabled.
- Main crystal oscillator disabled.
- System IO lines PA30, PA31, PB[0...3] in GPIO Input Pull-up mode.
- PLC in shutdown.
- All other GPIO lines in default state (see PIO Multiplexing table).
- Current measurement as per Figure 45-21.

Figure 45-21. Measurement Setup for Configurations C and D

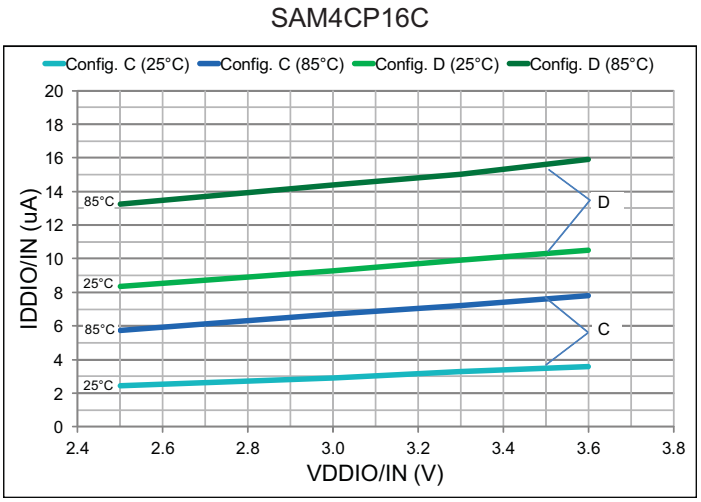


Note: No current is drawn on VDDIN power input in Backup mode. The pin VDDIN can be left unpowered in Backup mode. For Configuration C, if LCD is not used, VDDIN is not required.

Table 45-45. Typical Current Consumption Values for Backup Mode Configurations C and D

Conditions	Configuration C		Configuration D		Unit
	IDD_BU - AMP1	IDD_IN/IO - AMP2	IDD_BU - AMP1	IDD_IN/IO - AMP2	
VDDIO = 3.6V @25°C	0.05	2.6	0.05	9.5	µA
VDDIO = 3.3V @25°C		2.4		9.0	
VDDIO = 3.0V @25°C		2.1		8.5	
VDDIO = 3.6V @85°C	0.09	6.7	0.10	14.8	
VDDIO = 3.3V @85°C		6.2		14.0	
VDDIO = 3.0V @85°C		5.8		13.5	

Figure 45-22. Typical Current Consumption in Backup Mode for Configurations C and D



45.7.2 Wait Mode Current Consumption

Wait mode configuration and measurements are defined in Section 45.7.2.1 “Wait Mode Configuration”.

Reminder: In Wait mode, the core voltage regulator is on, but the device is not clocked. Flash power mode can be either in Standby mode or Deep power-down mode. Wait mode provides a much faster wake-up compared to Backup mode.

45.7.2.1 Wait Mode Configuration

- 32.768 kHz crystal oscillator running.
- 4 MHz RC oscillator running.
- Main crystal and PLLs stopped.
- RTT enabled on 1 Hz mode.
- One wake-up pin (WKUPx) used in Fast Wake-up mode.
- Anti-tamper inputs TMP0, TMP1, TMP2, TMP3 and RTCOUT0 enabled.
- System IO lines PA30, PA31, PB[0...3] in GPIO Input Pull-up mode.
- PLC in shutdown.
- All other GPIO lines in default state.
- Current measurement as per Figure 45-23.

Figure 45-23. Measurement Setup for Wait Mode Configuration

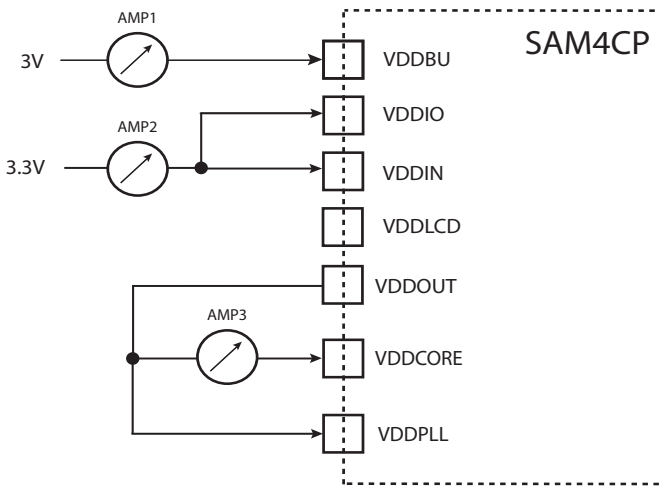


Table 45-46. Typical Current Consumption in Wait Mode

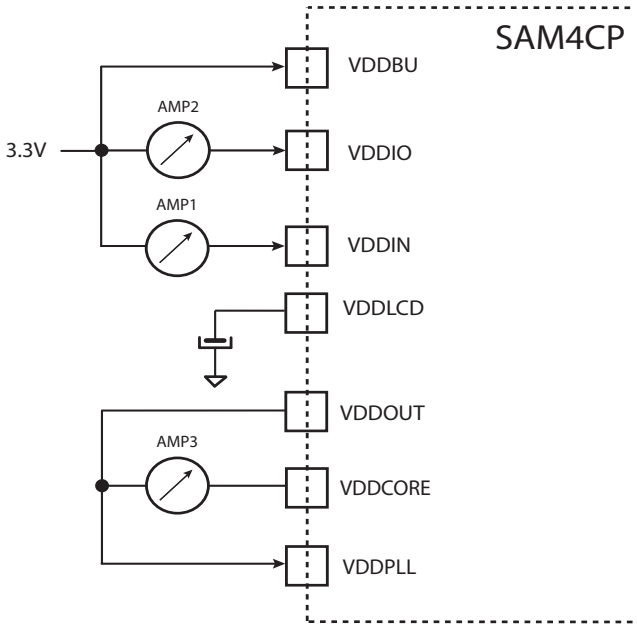
Conditions	IDD_BU - AMP1		IDD_IN/IO - AMP2		IDD_CORE - AMP3		Unit
	@25°C	@85°C	@25°C	@85°C	@25°C	@85°C	
Flash in Read-Idle mode	0.003	0.09	68	500	45	470	µA
Flash in Standby mode	0.003	0.09	66	500	45	470	
Flash in Deep Power Down mode	0.003	0.09	62	500	45	470	

45.7.3 Sleep Mode Current Consumption

Sleep mode configuration and measurements are defined in this section.

Reminder: The purpose of Sleep mode is to optimize power consumption of the device versus response time. In this mode, only the core clocks of CM4P0 and/or CM4P1 are stopped.

Figure 45-24. Measurement Setup for Sleep Mode

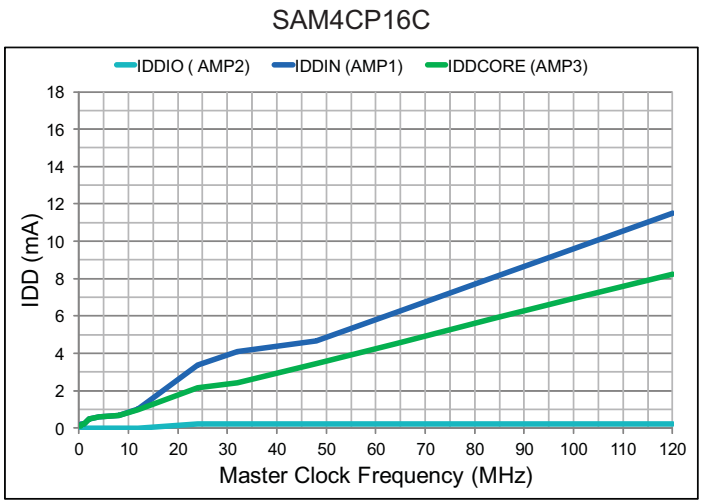


- VDDIO = VDDIN = 3.3V.
- VDDCORE = 1.2V (internal voltage regulator used).
- T_A = 25°C.
- Core 0 clock (HCLK) and Core 1 (CPHCLK) clock stopped.
- Sub-system 0 Master Clock (MCK), Sub-system 1 Master Clock (CPBMCK) running at various frequencies (PLLB used for frequencies above 12 MHz, fast RC oscillator at 12 MHz for the 12 MHz point, and fast RC oscillator at 8 MHz divided by 1/2/4/8/16/32 for lower frequencies).
- All peripheral clocks deactivated.
- No activity on I/O lines.
- PLC in shutdown.
- VDDPLL not taken into account. See PLL characteristics for further details.
- Current measurement as per Figure 45-24.

Table 45-47. Typical Sleep Mode Current Consumption Versus Frequency

Master Clock (MHz)	IDD_IN - AMP1	IDD_IO - AMP2	IDD_CORE - AMP3	Unit
120	14.26	0.22	10.83	mA
100	11.96	0.22	9.09	
84	10.1	0.22	7.68	
64	7.78	0.22	5.92	
48	5.93	0.22	4.48	
32	5.02	0.22	3.16	
24	3.85	0.22	2.4	
12	1.26	0.03	1.21	
8	0.88	0.03	0.83	
4	0.50	0.03	0.45	
2	0.32	0.03	0.27	
1	0.26	0.03	0.22	
0.5	0.22	0.03	0.20	
0.25	0.19	0.03	0.18	

Figure 45-25. Typical Current Consumption in Sleep Mode

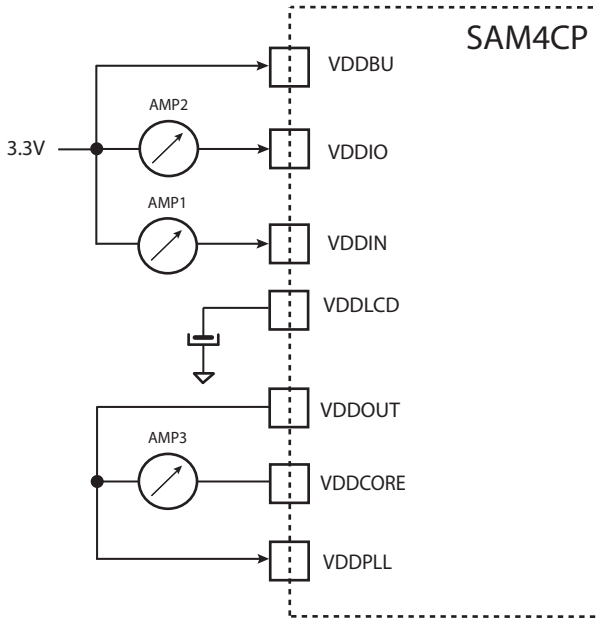


45.7.4 Active Mode Power Consumption

The current consumption configuration for Active mode, i.e., Core executing codes, is as follows:

- VDDIO = VDDIN = 3.3V.
- VDDCORE = 1.2V (internal voltage regulator used).
- T_A = 25°C.
- Sub-system 0 Master Clock (MCK) and Core Clock (HCLK), Sub-system 1 Master Clock (CPBMCK) and Core Clock (CPHCLK) running at various frequencies (PLLB used for frequencies above 8 MHz and fast RC oscillator at 8 MHz divided by 1/2/4/8/16/32 for lower frequencies).
- All peripheral clocks deactivated.
- No activity on I/O lines.
- PLC in shutdown.
- Flash Wait State (FWS) in EEFC_FMR adjusted versus core frequency.
- Current measurement as per Figure 45-26.

Figure 45-26. Measurement Setup for Active Mode



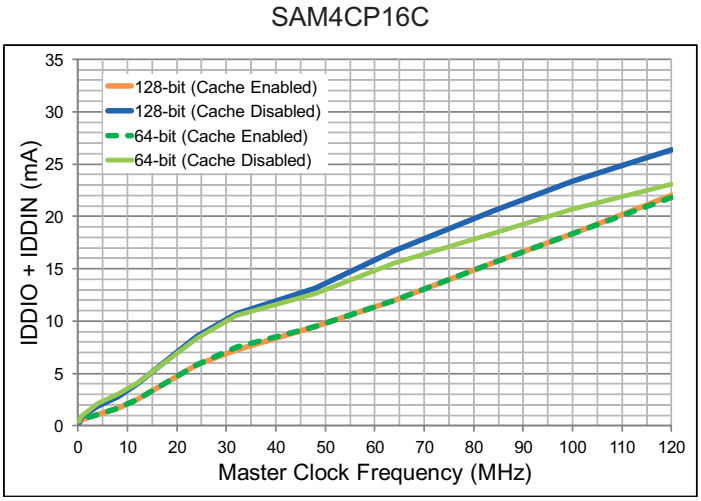
45.7.4.1 Test Setup 1: CoreMark™

- CoreMark on Core 0 (CM4P0) running out of flash in 128-bit or 64-bit Access mode with and without Cache Enabled. Cache is enabled above 0 WS.
- Sub-system 1 Master Clock (CPBMCK) and Core Clock (CPHCLK) stopped and in reset state.

Table 45-48. Test Setup 1 Current Consumption

Clock (MHz)	128-bit Flash Access						64-bit Flash Access						Unit
	Cache Enabled			Cache Disabled			Cache Enabled			Cache Disabled			
	IDD_IN (AMP1)	IDD_I0 (AMP2)	IDD_CORE (AMP3)	IDD_IN (AMP1)	IDD_I0 (AMP2)	IDD_CORE (AMP3)	IDD_IN (AMP1)	IDD_I0 (AMP2)	IDD_CORE (AMP3)	IDD_IN (AMP1)	IDD_I0 (AMP2)	IDD_CORE (AMP3)	
120	21.8	0.27	18.5	24.4	2.0	21.1	21.5	0.27	18.3	21.2	1.9	17.9	mA
100	18.1	0.27	15.4	21.6	1.8	18.9	18.1	0.27	15.4	19.0	1.8	16.3	
84	15.3	0.27	13.0	18.8	1.7	16.6	15.3	0.27	13.0	16.8	1.7	14.5	
64	11.8	0.27	10.1	15.2	1.5	13.5	11.8	0.27	10.1	14.1	1.4	12.5	
48	9.2	0.27	7.9	11.7	1.4	10.5	9.2	0.27	7.9	11.3	1.3	10.0	
32	7.2	0.27	5.6	9.5	1.2	7.9	7.2	0.27	5.6	9.3	1.2	7.7	
24	5.6	0.27	4.3	7.5	1.1	6.2	5.6	0.27	4.3	7.2	1.2	5.9	
12	2.4	0.09	2.4	3.1	0.9	3.1	2.4	0.09	2.4	3.1	1.0	3.1	
8	1.6	0.09	1.6	2.1	0.7	2.1	1.6	0.09	1.6	2.1	0.9	2.1	
4	1.0	0.09	1.0	1.4	0.5	1.4	1.0	0.09	1.0	1.4	0.8	1.4	
2	0.70	0.09	0.69	0.90	0.40	0.90	0.70	0.09	0.69	0.70	0.70	0.70	
1	0.54	0.09	0.53	0.65	0.30	0.65	0.55	0.09	0.54	0.65	0.40	0.65	
0.5	0.47	0.09	0.46	0.50	0.20	0.50	0.47	0.09	0.46	0.60	0.20	0.60	
0.25	0.25	0.09	0.24	0.26	0.10	0.25	0.25	0.09	0.24	0.36	0.10	0.25	

Figure 45-27. Typical Current Consumption in Active Mode (Test Setup 1)



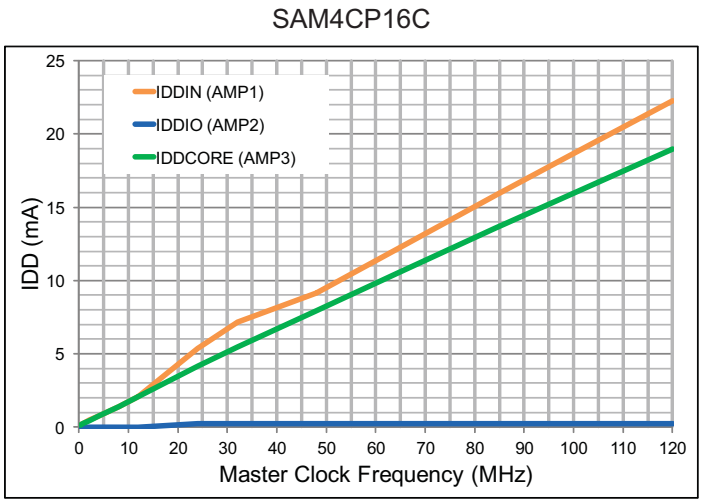
45.7.4.2 Test Setup 2: CoreMark

- CoreMark on Core 1 (CM4P1) running out of SRAM1 (Code) / SRAM2 (Data).
- Core 0 (CM4P0) in Sleep mode.

Table 45-49. Test Setup 2 Current Consumption

Clock (MHz)	SRAM1, SRAM2			Unit
	IDD_IN (AMP1)	IDD_I0 (AMP2)	IDD_CORE (AMP3)	
120	22.3	0.22	19.0	mA
100	18.7	0.22	16.0	
84	15.8	0.22	13.6	
64	12.1	0.22	10.5	
48	9.2	0.22	7.9	
32	7.1	0.22	5.5	
24	5.4	0.22	4.2	
12	2.1	0.01	2.1	
8	1.4	0.01	1.4	
4	0.78	0.01	0.77	
2	0.46	0.01	0.45	
1	0.29	0.01	0.28	
0.5	0.21	0.01	0.20	
0.25	0.13	0.01	0.12	

Figure 45-28. Typical Current Consumption in Active Mode (Test Setup 2)



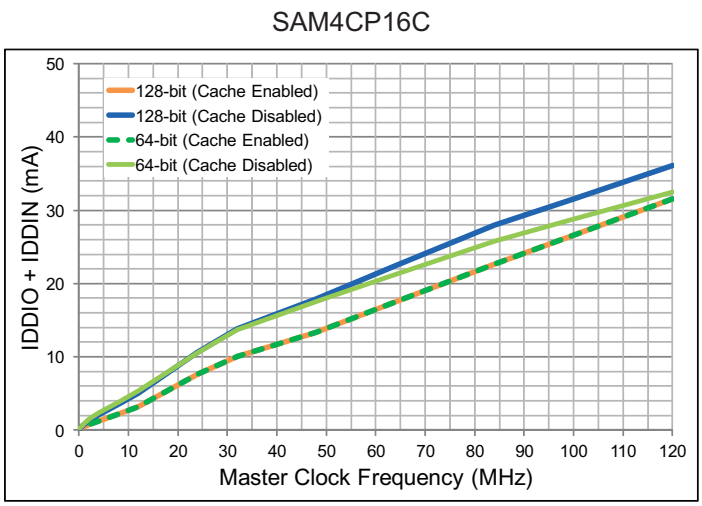
45.7.4.3 Test Setup 3: CoreMark

- CoreMark on Core 0 (CM4P0) running out of Flash in 128-bit or 64-bit Access mode with and without Cache Enabled. Cache is enabled above 0 WS.
- CoreMark on Core 1 (CM4P1) running out of SRAM1 (Code) / SRAM2 (Data).

Table 45-50. Test Setup 3 Current Consumption

	128-bit Flash Access						64-bit Flash Access						Unit
	Cache Enabled			Cache Disabled			Cache Enabled			Cache Disabled			
Clock (MHz)	IDD_IN (AMP1)	IDD_I0 (AMP2)	IDD_CORE (AMP3)	IDD_IN (AMP1)	IDD_I0 (AMP2)	IDD_CORE (AMP3)	IDD_IN (AMP1)	IDD_I0 (AMP2)	IDD_CORE (AMP3)	IDD_IN (AMP1)	IDD_I0 (AMP2)	IDD_CORE (AMP3)	
120	31.3	0.28	28.0	34.2	1.9	30.9	31.3	0.28	28.0	30.7	1.8	27.4	mA
100	26.4	0.28	23.6	29.8	1.8	27.1	26.4	0.28	23.6	27.0	1.8	24.3	
84	22.4	0.28	20.1	26.3	1.7	24.0	22.4	0.28	20.1	24.1	1.7	21.8	
64	17.2	0.28	15.6	21.0	1.5	19.3	17.2	0.28	15.6	19.6	1.6	18.0	
48	13.1	0.28	11.8	16.6	1.4	15.3	13.1	0.28	11.8	16.0	1.6	14.7	
32	9.8	0.28	8.1	12.6	1.2	10.9	9.8	0.28	8.1	12.3	1.4	10.6	
24	7.4	0.28	6.2	9.5	1.1	8.3	7.4	0.28	6.2	9.4	1.3	8.1	
12	3.1	0.11	3.1	4.2	0.88	4.2	3.1	0.11	3.1	4.2	1.2	4.2	
8	2.1	0.11	2.1	2.8	0.78	2.8	2.1	0.11	2.1	2.8	1.0	2.8	
4	1.1	0.11	1.1	1.5	0.58	1.5	1.1	0.11	1.1	1.5	0.9	1.5	
2	0.63	0.11	0.61	0.82	0.40	0.81	0.63	0.11	0.61	0.82	0.66	0.81	
1	0.38	0.11	0.37	0.47	0.26	0.46	0.38	0.11	0.37	0.47	0.38	0.46	
0.5	0.25	0.11	0.24	0.30	0.18	0.29	0.25	0.11	0.24	0.30	0.23	0.29	
0.25	0.14	0.11	0.13	0.16	0.12	0.15	0.14	0.11	0.13	0.16	0.14	0.15	

Figure 45-29. Typical Current Consumption in Active Mode (Test Setup 3)



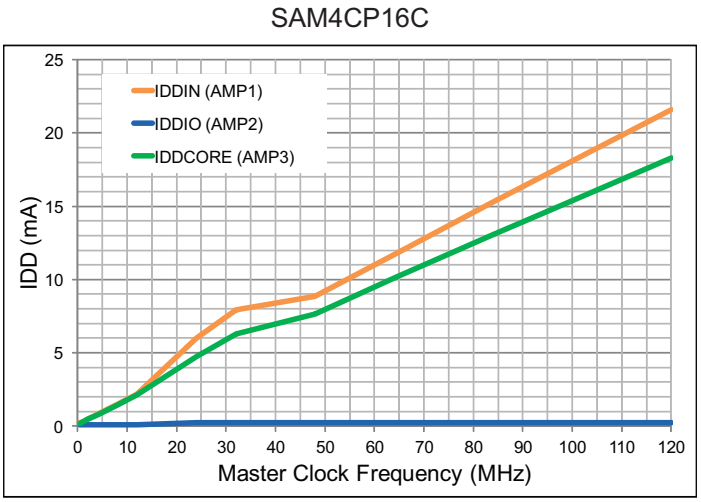
45.7.4.4 Test Setup 4: DSP Application (Five Cascaded 4th Order Biquad Filters) from ARM CMSIS DSP Library

- Application running on Core 1 (CM4P1) out of SRAM1 (Code) / SRAM2 (Data).
- Core 0 (CM4P0) in Sleep mode.

Table 45-51. Test Setup 4 Current Consumption

Clock (MHz)	DSP Application			Unit
	IDD_IN (AMP1)	IDD_I0 (AMP2)	IDD_CORE (AMP3)	
120	21.6	0.22	18.3	mA
100	18.1	0.22	15.4	
84	15.3	0.22	13.1	
64	11.7	0.22	10.1	
48	8.9	0.22	7.6	
32	7.9	0.22	6.3	
24	6.0	0.22	4.8	
12	2.2	0.08	2.1	
8	1.5	0.08	1.5	
4	0.80	0.08	0.76	
2	0.47	0.08	0.46	
1	0.30	0.08	0.29	
0.5	0.22	0.08	0.20	

Figure 45-30. Typical Current Consumption in Active Mode (Test Setup 4)



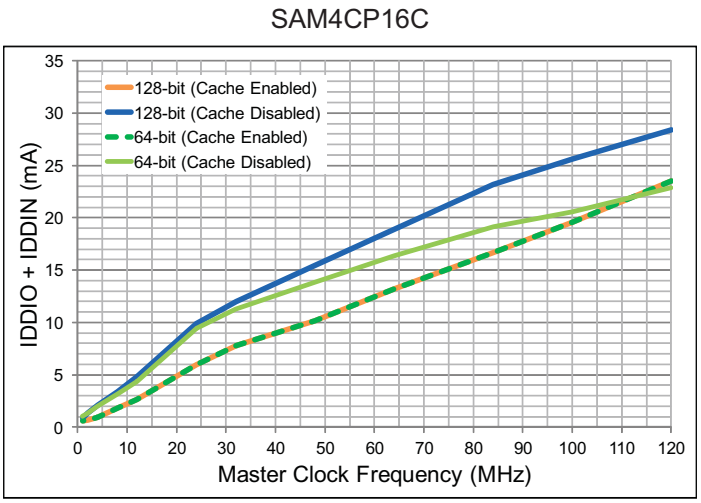
45.7.4.5 Test Setup 5: DSP Application (Five Cascaded 4th Order Biquad Filters) from ARM CMSIS DSP Library

- Application running on Core 0 (CM4P0) out of Flash in 128-bit or 64-bit Access mode with and without cache enabled. Cache is enabled above 0 WS.
- Sub-system 1 Master Clock (CPBMCK) and Core Clock (CPHCLK) stopped and in reset.
- VDDIO = VDDIN = 3.0V

Table 45-52. Test Setup 5 Current Consumption

Clock (MHz)	128-bit Flash Access						64-bit Flash Access						Unit
	Cache Enabled			Cache Disabled			Cache Enabled			Cache Disabled			
	IDD_IN (AMP1)	IDD_I0 (AMP2)	IDD_CORE (AMP3)	IDD_IN (AMP1)	IDD_I0 (AMP2)	IDD_CORE (AMP3)	IDD_IN (AMP1)	IDD_I0 (AMP2)	IDD_CORE (AMP3)	IDD_IN (AMP1)	IDD_I0 (AMP2)	IDD_CORE (AMP3)	
120	23.2	0.31	19.9	26.3	2.1	23.1	23.2	0.31	19.9	21.2	1.7	18.0	mA
100	19.3	0.31	16.6	23.7	2.0	21.0	19.3	0.31	16.6	18.9	1.7	16.2	
84	16.3	0.31	14.1	21.2	1.9	19.0	16.3	0.31	14.1	17.5	1.7	15.3	
64	12.9	0.31	11.2	17.2	1.8	15.5	12.9	0.31	11.2	14.8	1.6	13.1	
48	9.9	0.31	8.6	13.9	1.6	12.7	9.9	0.31	8.6	12.3	1.6	11.1	
32	7.5	0.31	5.8	10.6	1.4	9.0	7.5	0.31	5.8	9.9	1.4	8.2	
24	5.7	0.31	4.4	8.7	1.2	7.5	5.7	0.31	4.4	8.1	1.3	6.9	
12	2.6	0.08	2.6	4.0	0.82	3.9	2.6	0.08	2.6	3.5	0.8	3.4	
8	1.7	0.08	1.7	2.7	0.70	2.7	1.7	0.08	1.7	2.4	0.8	2.4	
4	0.89	0.08	0.88	1.6	0.51	1.6	0.89	0.08	0.88	1.3	0.7	1.3	
2	0.56	0.08	0.55	0.96	0.39	0.95	0.56	0.08	0.55	0.78	0.54	0.76	
1	0.55	0.08	0.54	0.67	0.20	0.66	0.55	0.08	0.54	0.68	0.37	0.67	

Figure 45-31. Typical Current Consumption in Active Mode (Test Setup 5)



45.7.5 Peripheral Power Consumption in Active Mode

Table 45-53. Power Consumption on $V_{VDDCORE}$[1]

Peripheral	Consumption (Typical)	Unit
PIO Controller	4.0	µA / MHz
UART0	5.4	
UART1	5.4	
USART[0 - 4]	7.7	
PWM	3.9	
TWI	5.3	
SPI	5.0	
Timer Counter (TCx)	2.7	
ADC	3.9	
SLCD	0.16	
AES: Performing AES256 Encryption	164	
TRNG	6.2	
ICM	5.2	

Notes: 1. V_{VDDIO} = 3.3V, $V_{VDDCORE}$ = 1.2V, T_A = 25°C.

Table 45-54. PLC Power Consumption

Parameter	Condition	Symbol	Rating			Unit
			Min	Typ	Max	
Power Consumption	T_J = 25ºC VDDIO = 3.3V VDDIN = 3.3V VDDIN AN = 3.3V	P_{25}	-	245	-	mW
Power Consumption (worst case)	T_J = 125ºC VDDIO = 3.6V VDDIN = 3.6V VDDIN AN = 3.6V	P_{125}	-	-	330	

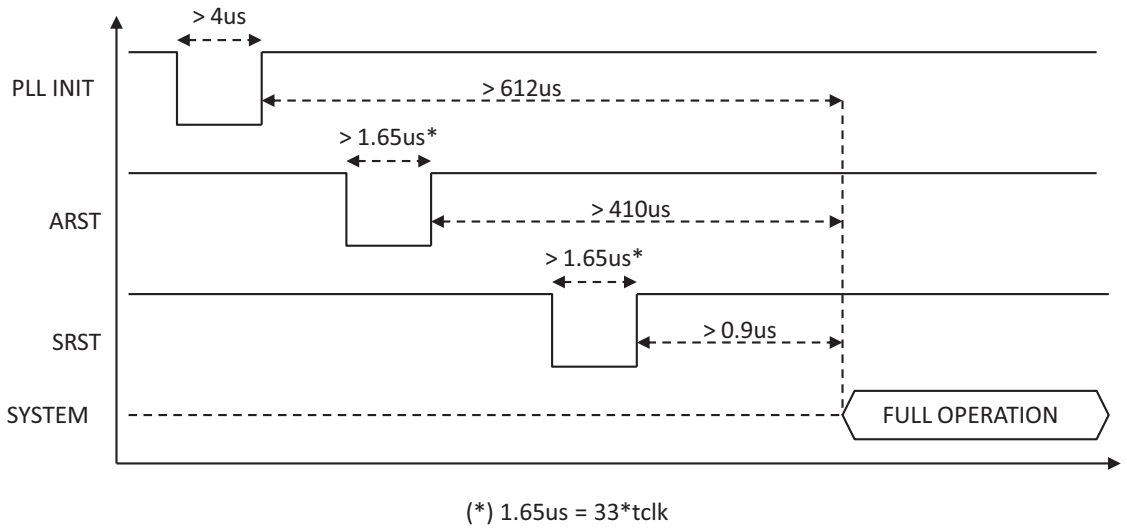
45.8 Power On Considerations

During power-on, PLL INIT pin should be tied to ground during 4 µs at least, in order to ensure proper system start up. After releasing PLL INIT, the system will start no later than 612 µs.

After power-up system can be restarted by means of low active pulse (min 1.65 µs) in ARST or SRST. System full operation starts after 410 µs (ARST pulse) or after 0.9 µs (SRST pulse).

In case of simultaneous tie down of more than one initialization pin the longest time for operation must be respected.

Figure 45-32. Power On timing diagram

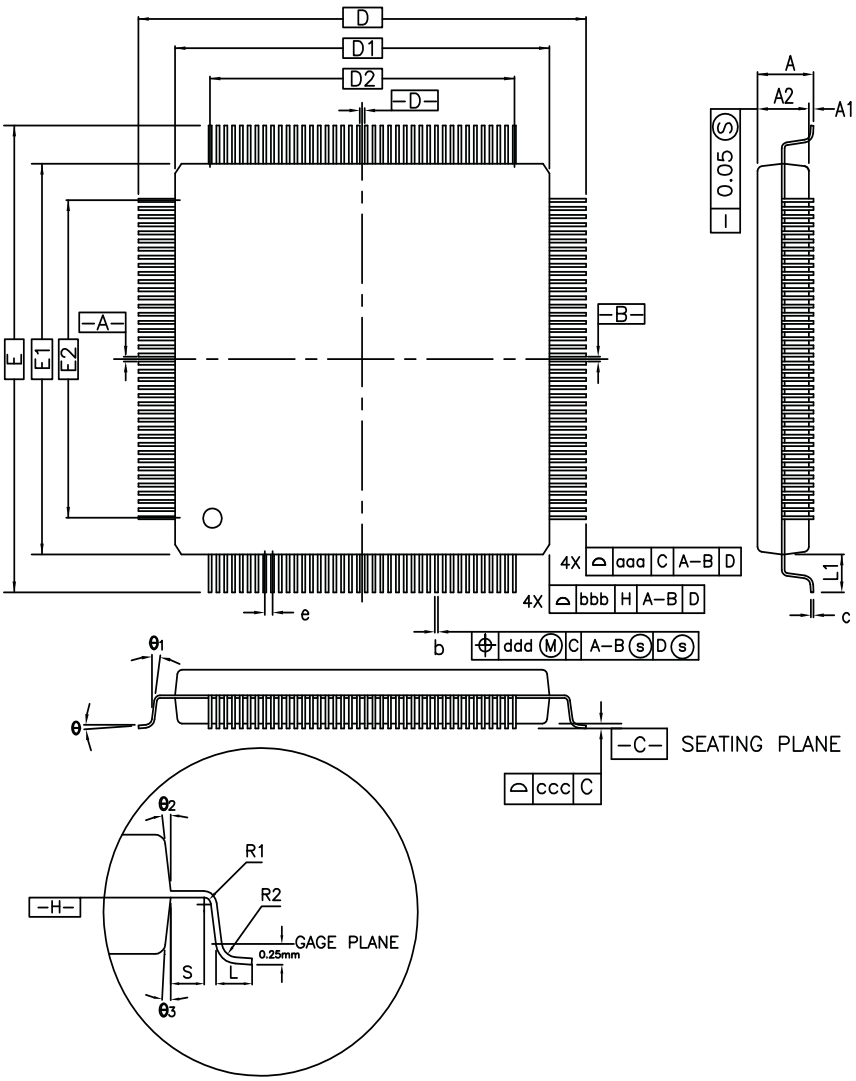


(*) 1.65us = 33*tclk

46. Mechanical Characteristics

46.1 176-lead LQFP Package

Figure 46-1. 176-lead LQFP Package Drawing



CONTROL DIMENSIONS ARE IN MILLIMETERS.

SYMBOL	MILLIMETER			INCH		
	MIN.	NOM.	MAX.	MIN.	NOM.	MAX.
A	—	—	1.60	—	—	0.063
A_1	0.05	—	0.15	0.002	—	0.006
A_2	1.35	1.40	1.45	0.053	0.055	0.057
D	26.00 BSC.			1.024 BSC.		
D_1	24.00 BSC.			0.945 BSC.		
E	26.00 BSC.			1.024 BSC.		
E_1	24.00 BSC.			0.945 BSC.		
R_2	0.08	—	0.20	0.003	—	0.008
R_1	0.08	—	—	0.003	—	—
θ	0°	3.5°	7°	0°	3.5°	7°
$θ_1$	0°	—	—	0°	—	—
$θ_2$	11°	12°	13°	11°	12°	13°
$θ_3$	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
L_1	1.00 REF			0.039 REF		
S	0.20	—	—	0.008	—	—

SYMBOL	176L					
	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	21.50			0.846		
E2	21.50			0.846		
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.

Table 46-1. LQFP Package Reference

JEDEC Drawing Reference	MS-026

Table 46-2. LQFP Package Characteristics

Moisture Sensitivity Level	3

This package respects the recommendations of the NEMI User Group.

46.2 Soldering Profile

Table 46-3 gives the recommended soldering profile from J-STD-020C.

Table 46-3. Soldering Profile

Profile Feature	Green Package
Average Ramp-up Rate (217°C to Peak)	3°C/sec. max.
Preheat Temperature 175°C ± 25°C	180 sec. max.
Temperature Maintained Above 217°C	60 sec. to 150 sec.
Time within 5°C of Actual Peak Temperature	20 sec. to 40 sec.
Peak Temperature Range	260°C
Ramp-down Rate	6°C/sec. max.
Time 25°C to Peak Temperature	8 min. max.

Note: The package is certified to be backward compatible with Pb/Sn soldering profile.

A maximum of three reflow passes is allowed per component.

46.3 Packaging Resources

This section provides land pattern definition.

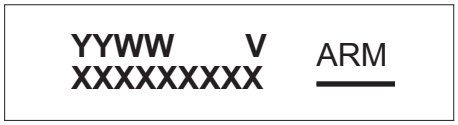
Refer to the following IPC standards:

- IPC-7351A and IPC-782 (*Generic Requirements for Surface Mount Design and Land Pattern Standards*) http://landpatterns.ipc.org/default.asp
- Atmel Green and RoHS Policy and Package Material Declaration Data Sheet http://www.atmel.com/about/quality/package.aspx

47. Marking

All devices are marked with the Atmel logo and the ordering code.

Additional marking is as follows:



where

- “YY”: Manufactory year
- “WW”: Manufactory week
- “V”: Revision
- “XXXXXXXXX”: Lot number

48. Ordering Information

Table 48-1. Ordering Codes for SAM4CP Devices

Ordering Code	MRL	Flash (Kbytes)	Package	Package Type	Temperature Operating Range
ATSAM4CP16C-AHU-Y	A	1*1024	LQFP176	Green	Industrial (-40°C to +85°C)

49. Errata

49.1 Supply Controller (SUPC)

49.1.1 SUPC: Supply Monitor (SM) on VDDIO

The Supply Monitor (SM) Sampling mode reducing the average current consumption on VDDIO is not functional.

Problem Fix/Workaround

Use the Supply Monitor in Continuous mode only.

49.1.2 SUPC: Core Voltage Regulator Standby Mode Control

The Core Voltage Regulator Standby mode controlled by the ONREG bit in SUPC_MR is not functional. This does not prevent to power VDDCORE and VDDPLL by using an external voltage regulator.

Problem Fix/Workaround

None. Do not use the ONREG Bit.

49.1.3 SUPC: Core Brownout Detector. Unpredictable Behavior if BOD is Disabled, VDDCORE is Lost and VDDIO is Powered

In Active mode or in Wait mode, if the Brownout Detector (BOD) is disabled (SUPC_MR: BODDIS=1) and power is lost on VDDCORE while VDDIO is powered, the device can be reset incorrectly and its behavior becomes then unpredictable.

Problem Fix/Workaround

When the Brownout Detector is disabled in Active or in Wait mode, VDDCORE must be always powered.

49.2 Parallel Input Output (PIO) Controller

49.2.1 PIO: Schmitt Trigger

- Schmitt triggers on all PIO controllers are not enabled by default (after reset) as stated in the product datasheet.
- Enable and disable values in the PIO Schmitt Trigger Register (for all PIO controllers) are inverted. The definition of PIO_SCHMITT fields must be as follows:
  - 0: Schmitt Trigger is disabled.
  - 1: Schmitt Trigger is enabled.

Problem Fix/Workaround

None. It is up to the application to enable Schmitt Trigger mode and to take into account the inverted values of the PIO_SCHMITT fields.

49.3 Watchdog (WDT) / Reinforced Safety Watchdog (RSWDT)

49.3.1 WDT / RSWDT not stopped in WAIT mode

When the Watchdog (WDT) or the Reinforced Safety Watchdog (RSWDT) is enabled and the WAITMODE bit set to 1 is used to enter Low-power Wait mode, the WDT/RSWDT is not halted. If the time spent in Wait mode is longer than the Watchdog (Reinforced Safety Watchdog) time-out, the device is reset provided that the WDT/RSWDT reset is enabled.

Problem Fix/Workaround

When entering Wait mode, the WaitForEvent (WFE) instruction of the Cortex-M4 processor must be used while the SLEEPDEEP bit of the Cortex-M System Control Register (SCB_SCR) is set to 0.

49.3.2 RSWDT Windowing Mode

When the RSWDT is configured in Windowing mode (WDD set lower than WDV in RSWDT_MR), an unexpected watchdog reset order may be sent to the Reset Controller (RSTC).

Problem Fix/Workaround

Do not use the Windowing mode of the RSWDT and set WDD to 4095 in RSWDT_MR.

49.4 Enhanced Embedded Flash Controller (EEFC)

49.4.1 EEFC: Erase Sector (ES) Command Cannot be Performed if a Subsector is Locked (Only in Flash sector 0)

If one of the subsectors

- small sector 0
- small sector 1
- larger sector

is locked within the Flash sector 0, the erase sector (ES) command cannot be processed on non-locked subsectors. Refer to “Flash Overview” in Section “Memories”.

Problem Fix/Workaround

All the lock bits of the sector 0 must be cleared prior to issuing the ES command. After the ES command has been issued, the lock bits must be reverted to the state before clearing them.

49.5 Wait For Interrupt (WFI)

49.5.1 Unpredictable Software Behavior When Entering Sleep Mode

When entering Sleep mode, if an interrupt occurs during WFI or WFE instruction (with PMC_FSMR.LPM=0), the ARM core may read a wrong data, thus leading to unpredictable behavior of the software. This issue is not present in Wait mode.

Problem Fix/Workaround

The slave interface for the Flash must be set to no default master in the Bus Matrix Controller.

This is done by setting the field DEFMSTR_TYPE in the register MATRIX_SCFG to NO_DEFAULT.

```
MATRIX_SCFG[2] = MATRIX_SCFG.SLOT_CYCLE(0x1FF) | MATRIX_SCFG.DEFMSTR_TYPE(0x0);
```

This operation must be done once in the software or the instruction before WFI or WFE.

49.6 Power Supply and Power Control / Clock System

49.6.1 CORE 1 SysTick Counter Erratic Behavior

If the CORE 0 processor clock (HCLK) frequency is higher than four times the frequency of the CORE 1 processor clock (CPHCLK), the SysTick counter behavior is erratic.

Problem Fix/Workaround

Always ensure that $f_{HCLK} < 4 \times f_{CPHCLK}$

49.7 Power Management Controller (PMC)

49.7.1 SRCB Bit in CKGR_PLLB Register

The SRCB bit is programmed in bit 29 of the CKGR_PLLB register but must be read in bit 27 of this register.

Problem Fix/Workaround

For SRCB, read bit 27 of the CKGR_PLLB register.

50. Revision History

In the table that follow, the most recent version of the document appears first.

Doc. Rev. 43080	Comments	Change Request Ref.
	G3 Power Line Communications (GPLC) Figure 27-5 and Figure 27-7: updated.	
	Clock Generator Section 29.4 ”Slow Clock”: deleted “This allows the slow clock to be valid in a short time (about 100 µs).” Section 29.4.2 ”32.768 kHz Crystal Oscillator”: deleted Figure 29-2. Typical 32.768 kHz Crystal Oscillator Connection.	
	Power Management Controller (PMC) Instances of “32768 Hz” changed to “32.768 kHz”. Section 30.10 ”Main Processor Fast Startup”: minor editorial changes. Figure 30-4: corrected “CDFEV” to “CFDEV” and “CDFS” to “CFDS”. Section 30.18.10 ”PMC Clock Generator PLLB Register”: changed DIVB description.	
I	**Timer Counter (TC)** Section 37.6.14.2 ”Input Preprocessing”: added limitation on maximum pulse duration. Section 37.6.14.4 ”Position and Rotation Measurement”: in 3rd paragraph, added “The process must be started by configuring TC_CCR.CLKEN and TC_CCR.SWTRG.” Section 37.6.14.6 ”Detecting a Missing Index Pulse”: corrected value of TC_RC0.RC in example in 2nd paragraph. Added Section 37.6.14.7 ”Detecting Contamination/Dust at Rotary Encoder Low Speed”. Section 37.7.14 ”TC Block Mode Register”: updated MAXFILT field description.	
	Advanced Encryption Standard (AES) Section 41.1 ”Description”: corrected index of AES_KEYWRx registers from 3 to 7. Section 41.2 ”Embedded Characteristics”: replaced “12/14/16 Clock Cycles Encryption/Decryption Processing Time” with “10/12/14 Clock Cycles Encryption/Decryption Inherent Processing Time”. Removed sentence “Values which are not listed in the table must be considered as “reserved”.” in Section 41.5.2 ”AES Mode Register”.	

Doc. Rev. 43080	Comments	Change Request Ref.
H	**Power Supply and Power Control** Section 5.5.1.1 ”Entering and Exiting Backup Mode”: added note in step(7).	
	Peripherals Table 11-1: replaced "Watchdog Timer/Reinforced Watchdog Timer" with "Watchdog Timer".	
	Reset Controller (RSTC) Reworked Section 15.1 ”Description” and Section 15.2 ”Embedded Characteristics”. Figure 15-1, Figure 15-2, Figure 15-3, Figure 15-4, Figure 15-5 and Figure 15-6: updated. Updated Section 15.4.1 ”Overview”, Section 15.4.2.1 ”NRST Signal or Interrupt”, Section 15.4.3.3 ”Watchdog Reset” and Section 15.4.3.5 ”User Reset”. Replaced “is set” with “is written to 1” and “is reset” with “is written to 0”. Added Section 15.4.5 ”Managing Reset at Application Level”.	
	Real-time Clock (RTC) Reworked Section 17.5.6 ”Updating Time/Calendar”.	
	Reinforced Safety Watchdog Timer (RSWDT) Removed all references to interrupt by modifying: Figure 19-1, Section 19.2 ”Embedded Characteristics” and Section 19.4 ”Functional Description”. Section 19.5.2 ”Reinforced Safety Watchdog Timer Mode Register” (bit WDFIEN is reserved now).	
	Supply Controller (SUPC) Section 20.2 ”Embedded Characteristics”: bullet “A Supply Monitor Detection on VDDBU_SW Triggers a System Reset” changed to “A Zero-power Power-on-reset on VDDBU_SW Triggers a System Reset”. Figure 20-2: added SMOS waveform. Section 20.4.8.1 ”Supply Monitor Reset”: updated second paragraph. Section 20.5 ”Register Write Protection”: removed “Supply Controller Wake-up Inputs Register” from list of protectable registers. Section 20.6.3 ”Supply Controller Control Register”, Section 20.6.4 ”Supply Controller Supply Monitor Mode Register”, Section 20.6.5 ”Supply Controller Mode Register” and Section 20.6.6 ”Supply Controller Wake-up Mode Register”: added sentence about write-protection. Section 20.6.5 ”Supply Controller Mode Register”: added value “1” to LCDMODE field description. Section 20.6.9 ”System Controller Write Protection Mode Register”: updated WPEN bit description.	
	Enhanced Embedded Flash Controller (EEFC) Section 22.4.3.6 ”Calibration Bit”: changed information on oscillators that are calibrated in production. Table 22-5: updated.	
	Fast Flash Programming Interface (FFPI) Figure 23-1: changed input source for XIN. Table 23-1: updated XIN information. Section 23.3.3 ”Entering Parallel Programming Mode”: deleted note on device clocking and reworded steps 2 and 3. Section 23.3.5.6 ”Flash Security Bit Command”: reworded step 3.	

Doc. Rev. 43080	Comments	Change Request Ref.
H	**G3 Power Line Communications (GPLC)** Section 27.5.3 ”Zero-crossing detector”: updated. Added Section 27.5.3.2 ”Zero Crossing Config register”.	
	Clock Generator Updated Figure 29-1.	
	Power Management Controller (PMC) Updated Figure 30-1. Section 30.10 ”Main Processor Fast Startup”: inserted warning “The duration of the WKUPx pins active level must be greater than four main clock cycles.” Section 30.17 ”Register Write Protection”: added “PMC Clock Generator Main Clock Frequency Register” to list of protectable registers.	
	Serial Peripheral Interface (SPI) Section 33.8.1 ”SPI Control Register”: Removed TXFCLR, RXFCLR, FIFOEN and FIFODIS bits. Section 33.7.3 ”Master Mode Operations”: Modified transmission condition description.	
	Universal Synchronous Asynchronous Receiver Transmitter (USART) At end of Section 36.6.1.2 ”Fractional Baud Rate in Asynchronous Mode” added warning “When the value of field FP is greater than 0...”. Removed sentence “This feature is only available when using USART normal mode”. Section 36.6.3.11 ”Receiver Time-out”: deleted redundant paragraphs on STTTO and RETTO. Section 36.6.4 ”ISO7816 Mode”: corrected USART_MODE value for protocol T = 1. Section 36.6.7.5 ”Character Transmission”: corrected bit names: “RTSEN” to “RCS” and “RTSDIS” to “FCS”. Section 36.7.3 ”USART Mode Register”: updated description for row 0xE, SPI_MASTER. Section 36.7.15 ”USART Baud Rate Generator Register”: added warning “When the value of field FP is greater than 0...” to FP field description.	
	Timer Counter (TC) Section 37.6.3 ”Clock Selection”: updated notes (1) and (2). Replaced TIOA, TIOB, TCLK with TIOAx, TIOBx, TCLKx. Section 37.6.14.4 ”Position and Rotation Measurement”: updated. Added Section 37.6.14.6 ”Detecting a Missing Index Pulse”.	
	Pulse Width Modulation Controller (PWM) Section 38.1 ”Description”: deleted sentence “All PWM macrocell accesses are made through APB mapped registers.” Section 38.6.3 ”PWM Controller Operations”: removed section “Interrupts” (content redundant with information provided in descriptions of registers PWM_IER, PWM_IDR, PWM_IMR, and PWM_ISR). Section 38.7.1 ”PWM Mode Register”: updated bit descriptions.	
	Segment Liquid Crystal Display Controller (SLCDC) Removed all information related to Register Write Protection: - “Register Write Protection” in Section 39.2 ”Embedded Characteristics”. - Section 39.6.11 “Register Write Protection”. - SLCDC Write Protection Mode Register (SLCDC_WPMR) bit. - SLCDC Write Protection Status Register (SLCDC_WPSR) bit.	

Doc. Rev. 43080	Comments	Change Request Ref.
H	**Analog-to-Digital Converter (ADC)** Figure 40-1: updated. Reworked Section 40.5.5 ”I/O Lines”. Section 40.7.8 ”ADC Interrupt Enable Register”: updated FORCEREF and ONREF bit descriptions.	
	Integrity check monitor (ICM) Section 42.5.2.2 ”ICM Region Configuration Structure Member”: removed MRPROT field. Section 42.6.1 ”ICM Configuration Register”: removed fields DAPROT and HAPROT; updated description DUALBUFF field.	
	Electrical Characteristics Updated: Table 45-31.	
G	**Block Diagram** Updated Figure 2-1.	
	Package and Pinout Table 4-1: added WKUP0 in pin number 71.	
	Power Supply and Power Control Updated Figure 5-1. Updated Section 5.1.5.2 ”Single Supply Operation with Backup Battery”.	
	Input/Output Lines Updated Section 6.9 ”ERASE Pin”.	
	Real-time Event Management Updated Table 9-1.	
	System Controller Deleted Figure 10-1 “System Controller Block Diagram”.	
	SAM4CP Boot Program Section 14.3 ”Flow Diagram” and Section 14.4 ”Device Initialization”: updated.	
	Reset Controller (RSTC) Table 15-1: added note (1) to reset value of register RSTC_SR.	
	Real-time Clock (RTC) Figure 17-5: corrected two instances of “3,906 ms” to “3.906 ms”. Section 17.2 ”Embedded Characteristics”: merged bullet on asynchronous design with bullet on ultra low-power. Table 17-2: added offset 0xCC as reserved. Section 17.6.1 ”RTC Control Register”: updated descriptions for bits UPDTIM and UPDCAL. Deleted “All non-significant bits read zero” from the following registers: Section 17.6.3 ”RTC Time Register”, Section 17.6.4 ”RTC Calendar Register”, Section 17.6.13 ”RTC TimeStamp Time Register 0”, Section 17.6.14 ”RTC TimeStamp Time Register 1”, Section 17.6.15 ”RTC TimeStamp Date Register” and Section 17.6.16 ”RTC TimeStamp Source Register”. Added sentence on register report of timestamp to Section 17.6.13 ”RTC TimeStamp Time Register 0”, Section 17.6.14 ”RTC TimeStamp Time Register 1” and Section 17.6.15 ”RTC TimeStamp Date Register”.	

Doc. Rev. 43080	Comments	Change Request Ref.
G	**Watchdog Timer (WDT)** Section 18.5.1 ”Watchdog Timer Control Register”: added note on modification of WDT_CR values. Section 18.5.2 ”Watchdog Timer Mode Register”: updated note on modification of WDT_MR values.	
	Inter-processor Communication (IPC) Updated Section 25.1 ”Description”, Section 25.3.2 ”Interrupt Line” and Section 25.4.1.2 ”Interrupt Source Control”. Updated Figure 25-1 and Figure 25-3.	
	Serial Peripheral Interface (SPI) Section 33.8.1 ”SPI Control Register”: added bits FIFODIS; FIFOEN, RXFCLR, TXFCLR and REQCLR.	
	Two-wire Interface (TWI) Replaced “Figure 34-13 below shows a byte write to an Atmel AT24LC512 EEPROM” with “Figure 34-13 below shows a byte write to a memory device”. Section 34.7.3.4 ”Master Transmitter Mode” and Section 34.7.5.4 ”Receiving Data” added note on clearing TXRDY flag. Section 34.8.5 ”TWI Clock Waveform Generator Register” in CKDIV field description, changed “The TWCK is used to increase both SCL high and low periods” to “The CKDIV field is used to increase both TWCK high and low periods”.	
	Universal Synchronous Asynchronous Receiver Transmitter (USART) Section 36.6.7.5 ”Character Transmission” after first paragraph, inserted new paragraph “The chip select line is de-asserted for a period equivalent to three bits between the transmission of two data”. Section 36.7.6 ”USART Interrupt Enable Register (SPI_MODE)”, Section 36.7.8 ”USART Interrupt Disable Register (SPI_MODE)”, Section 36.7.10 ”USART Interrupt Mask Register (SPI_MODE)” : added bit NSSE (register bit 19). Section 36.7.12 ”USART Channel Status Register (SPI_MODE)” : added bit NSSE (register bit 19) and bit NSS (register bit 23). Figure 36-2 and Figure 36-3: added label “Selected Clock” to USCLKS mux output. Section 36.6.1.1 ”Baud Rate in Asynchronous Mode” in baud rate calculation formula, replaced “$f_{peripheral\ clock}$” with “Selected Clock”. Section 36.6.1.3 ”Baud Rate in Synchronous Mode or SPI Mode” in second paragraph, replaced “$f_{peripheral\ clock}$” with “Selected Clock”.	
	Electrical Characteristics Updated Table 45-1 and Table 45-3. Updated Figure 45-22. Added (1) and (2) in Table 45-5.	

Doc. Rev. 43080	Comments	Change Request Ref.
F	**Features** Updated “Shared System Controller” and “Shared Peripherals”	
	Power Supply and Power Control Table 5-1: updated VDDIO and VDDIN comments. Figure 5-2, Figure 5-3 and Figure 5-4: modified.	
	Input/Output Lines Section 6.2 ”System I/O Lines” replaced note 2 “Refer to the section “3 to 20 MHz Crystal Oscillator” with Refer to “3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator” in the section “Clock Generator”. Updated Section 6.4 ”NRST Pin”.	
	Product Mapping and Peripheral Access Added Figure 7-1.	
	Memories Deleted “Flash Organization” in Figure 8-1 and Figure 8-2.	
	Peripherals Section 11.4.2 ”Reset State” PIO or signal name: “the PIO line is in I/O mode” changed to “the PIO line is in general-purpose I/O (GPIO)”.	
	Real-time Clock (RTC) Updated Section 17.5.7 ”RTC Accurate Clock Calibration”	
	Enhanced Embedded Flash Controller (EEFC) Updated Section 22.2 ”Embedded Characteristics”. Updated Section 22.4.1 ”Embedded Flash Organization” and added Figure 22-1. Updated Section 22.4.3.7 ”Security Bit Protection”, Section 22.4.3.8 ”Unique Identifier Area” and Section 22.4.3.9 ”User Signature Area”.	
	Bus Matrix (MATRIX) Updated Section 26.8 ”Register Write Protection”. In Section 26.9.5 ”Core Debug Configuration Register” bit descriptions, replaced “CROSS_TRIGx” with “CROSS_TRGx”. Section 26.9.7 ”Write Protection Status Register”: modified WPVS bit description.	
	Clock Generator Aligned naming of clock source oscillators in Section 29.2 ”Embedded Characteristics” and Figure 29-1. Section 29.4 ”Slow Clock” and Section 29.5 ”Main Clock”: updated all the subsections of these sections. Section 29.6.1 ”Divider and Phase Lock Loop Programming”: updated.	

Doc. Rev. 43080	Comments	Change Request Ref.
F	**Power Management Controller (PMC)** Section 30.1 ”Description”, Section 30.2 ”Embedded Characteristics”, Section 30.10 ”Main Processor Fast Startup”, Section 30.11 ”Main Processor Startup from Embedded Flash”, Section 30.13 ”Main Clock Failure Detector”, Section 30.14 ”32.768 kHz Crystal Oscillator Frequency Monitor” and Section 30.15 ”Programming Sequence”: updated. Section 30.18.7 ”PMC Clock Generator Main Oscillator Register”, Section 30.18.8 ”PMC Clock Generator Main Clock Frequency Register”, Section 30.18.13 ”PMC Interrupt Enable Register”, Section 30.18.14 ”PMC Interrupt Disable Register”, Section 30.18.15 ”PMC Status Register” and Section 30.18.16 ”PMC Interrupt Mask Register”: updated.	
	True Random Number Generator (TRNG) Section 44.1 ”Description” and Section 44.2 ”Embedded Characteristics”: updated names of referenced test suites. Section 44.5 ”Functional Description”: updated terminology in text and in Figure 44-2. Table 44-2: “Register Mapping”: defined offset ranges 0x04 - 0x0C, 0x20 - 0x4C and 0x54 - 0xFC as reserved.	
E	**Real-time Clock (RTC)** Updated Section 17.1 ”Description” and Section 17.5 ”Functional Description” (removed references to the 20th century). Updated Section 17.5.7 ”RTC Accurate Clock Calibration” (added figures and descriptions).	
	Universal Synchronous Asynchronous Receiver Transmitter (USART) Section 36.5.1 ”I/O Lines”: deleted paragraph “To prevent the TXD line...”.	
	Pulse Width Modulation Controller (PWM) Corrected formula in Section 38.6.2.2 ”Waveform Properties”: “DIVAB” changed to “DIVB”.	
D	**ARM Cortex-M4 Processor** Table 12-32: changed NVIC_IPR10 offset value to 0xE000E428 (was 0xE000E426).	
	Supply Controller (SUPC) Section 20.4.9.2 ”Wake-up Inputs”: added one paragraph and Figure 20-5. Section 20.6.7 ”Supply Controller Wake-up Inputs Register”: updated descriptions of fields WKUPENx and WKUPTx.	
	Power Management Controller (PMC) Section 30.13 ”Main Clock Failure Detector”: modified.	
	Serial Peripheral Interface (SPI) Section 33.7.3.5 ”Peripheral Selection”: in last paragraph, replaced “If LASTXFER is used, the command must be issued before writing the last character” with “If LASTXFER is used, the command must be issued after writing the last character”. Section 33.7.3.8 ”Peripheral Deselection without PDC”: in last paragraph, replaced “the Last Transfer (LASTXFER) bit in the SPI_MR must be set to 1 before writing the last data to transmit into the SPI_TDR” with “the Last Transfer (LASTXFER) bit in SPI_CR must be set after writing the last data to transmit into SPI_TDR”. Section 33.8.9 ”SPI Chip Select Register”: updated SCBR, DLYBS, and DLYBCT field descriptions.	

Doc. Rev. 43080	Comments	Change Request Ref.
D	**Universal Synchronous Asynchronous Receiver Transmitter (USART)** Replaced instances of “Kb/s” and “kbps” with “kbit/s”; replaced instance of “taken into account” with “processed”. Section 36.5.3 ”Interrupt Sources”: changed title (was “Interrupt”). Section 36.6.3.15 ”Hardware Handshaking”: replaced “if the receiver is disabled and if the status RXBUFF” with “if the receiver is disabled or if the status RXBUFF”. Section 36.7.1 ”USART Control Register”: in STTTO bit description, replaced “before clocking the time-out counter” with “before enabling the time-out counter”. Section 36.7.12 ”USART Channel Status Register (SPI_MODE)”: added descriptions for bits ENDRX, ENDTX, TXBUFE, and RXBUFF. Section 36.7.15 ”USART Baud Rate Generator Register”: restructured equations in CD field description. Section 36.7.16 ”USART Receiver Time-out Register”: restructured equation in TO field description. Section 36.7.17 ”USART Transmitter Timeguard Register”: restructured equation in TG bit description.	
	Timer Counter (TC) Replaced all occurrences of “quadrature decoder logic” with “quadrature decoder” or “QDEC” throughout the document. Moved Table 37-1 from Section 37.1 ”Description” to Section 37.3 ”Block Diagram”. Section 37.6.16 ”Register Write Protection”: added “The Timer Counter clock of the first channel must be enabled to access TC_WPMR”. Section 37.7.5 ”TC Counter Value Register”: in CV field description, added notation “IMPORTANT: For 16-bit channels, CV field size is limited to register bits 15:0”. Section 37.7.6 ”TC Register A”: in RA field description, added notation “IMPORTANT: For 16-bit channels, RA field size is limited to register bits 15:0”. Section 37.7.7 ”TC Register B”: in RB field description, added notation “IMPORTANT: For 16-bit channels, RB field size is limited to register bits 15:0”. Section 37.7.8 ”TC Register C”: in RC field description, added notation “IMPORTANT: For 16-bit channels, RC field size is limited to register bits 15:0”. Section 37.7.19 ”TC Write Protection Mode Register”: updated WPEN bit description.	
	Advanced Encryption Standard (AES) Section 41.4.4.1 ”Manual Mode” and Section 41.4.4.3 ”PDC Mode”: reformatting of bullets to numbered steps.	
	Integrity check monitor (ICM) Section 42.4.2 ”Interrupt Sources”: modified title (was “Interrupt”). Section 42.5.1 ”Overview”: reorganized and updated content under this new heading. Section 42.5.2.2 ”ICM Region Configuration Structure Member”: updated descriptions of fields RHIEN, DMIEN, BEIEN, WCIEN, ECIEN and SUIEN. Updated Section 42.5.4 ”Using ICM as SHA Engine”. Added Section 42.5.4.1 ”Settings for Simple SHA Calculation”. Section 42.5.5 ”ICM Automatic Monitoring Mode”: minor editorial changes. Moved Section 42.5.7 ”Security Features” to end of Section 42.5 ”Functional Description”. Section 42.6.3 ”ICM Status Register”: updated descriptions of fields RAWRMDIS and RMDIS.	

Doc. Rev. 43080	Comments	Change Request Ref.
C	**Reset Controller (RSTC)** Section 15.5 ”Reset Controller (RSTC) User Interface”: changed RSTC_SR reset value to 0x0000_0000 (was 0x0001_0000).	
	Real-time Clock (RTC) Section 17.6.11 ”RTC Interrupt Mask Register”: added TDERR field. Figure 17-1: replaced “APB” with “System Bus”.	
	Watchdog Timer (WDT) Section 18.5.3 ”Watchdog Timer Status Register”: appended names of bits WDUNF and WDERR with “(cleared on read)”.	
	Enhanced Embedded Flash Controller (EEFC) Section 22.5.1 ”EEFC Flash Mode Register”: FWS field: Inversed equation.	
	Clock Generator Section 29.4.2 ”32.768 kHz Crystal Oscillator”: deleted redundant content from end of section; appended “OSCBYPASS bit of the Supply Controller Mode Register (SUPC_MR) needs to be set at 1” to read “OSCBYPASS bit of the Supply Controller Mode Register (SUPC_MR) needs to be set at 1 prior to writing a 1 in bit XTALSEL”. Section 29.6 ”Divider and PLL Blocks”: modified the first paragraph.	
	Parallel Input/Output Controller (PIO) Section 32.6.32 ”PIO Pad Pull-Down Status Register”: deleted sentence “This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register”.	
	Universal Asynchronous Receiver Transmitter (UART) Section 35.5.1 ”Baud Rate Generator”: deleted equation for Baud Rate calculation. Section 35.6.9 ”UART Baud Rate Generator Register”: added equation to CD bit description.	
	Timer Counter (TC) Section 37.1 ”Description”: modified first paragraph to read “A Timer Counter (TC) module includes three identical TC channels. The number of implemented TC modules is device-specific”. Section 37.6.14 ”Quadrature Decoder”: removed subsection “Missing Pulse Detection and Auto-correction”. Section 37.6.14.4 ”Position and Rotation Measurement”: modified the second paragraph. Section 37.7.2 ”TC Channel Mode Register: Capture Mode”: in ‘Name’ line, replaced “(WAVE = 0)” with “(CAPTURE_MODE)”. Section 37.7.3 ”TC Channel Mode Register: Waveform Mode”: in ‘Name’ line, replaced “(WAVE = 1)” with “(WAVEFORM_MODE)”. Section 37.7.14 ”TC Block Mode Register”:removed AUTOC bit and MAXCMP field. Section 37.7.18 ”TC QDEC Interrupt Status Register”: removed MPE bit.	
	Advanced Encryption Standard (AES) Section 41.4.1 ”AES Register Endianness”: added. Section 41.5.6 ”AES Interrupt Status Register”: Reworded DATRDY and TAGRDY bit descriptions. Added “(cleared by...)” to ENDRX, ENDTX, RXBUFF, TXBUFE, URAD, URAT and TAGRDY bit descriptions.	
	Electrical Characteristics Section 45.5.10.5 updated: modified Figure 45-17, added equation and information after the figure. Table 45-28 updated: added the values of C_{XTAL} and $C_{PARA24M}$. Modified the notes below the table.	

Doc. Rev. 43080	Comments	Change Request Ref.
B	**Block Diagram** Added pins WKUP[0:15]. Moved Dual Watchdog block out of backup zone. Added power supply for Segment LCD Controller.	
	Signal Description Table 3-1, “Signal Description List” :Added section Supply Controller with WKUP0 and WKUP[1:15].	
	Input/Output Lines Changed Section 6.3 title from “Test Pin” to “TST Pin” .	
	Reset Controller (RSTC) Section 15.5.2 ”RSTC Status Register”: modified bit descriptions.	
	Real-time Timer (RTT) Section 16.4 ”Functional Description”: replaced “... the CRTV field must be read twice at the same value to improve accuracy of the returned value.” with “...the CRTV field must be read twice at the same value to read a correct value”. Section 16.5.4 ”Real-time Timer Status Register” added “(cleared on read)” to each bit description.	
	Enhanced Embedded Flash Controller (EEFC) Replaced “modulo” with “a multiple of” in Section 22.4.3.3 ”Erase Commands”. Section 22.5.1 ”EEFC Flash Mode Register”: replaced “No Flash read should be done during change of this register” with “No Flash read should be done during change of this field”. Corrected FARG description for EPA command in Section 22.5.2 ”EEFC Flash Command Register”. Section 22.5.3 ”EEFC Flash Status Register”: added “(cleared...)” to each bit description.	
	G3 Power Line Communications (GPLC) Sections order redefined. Modified Section 27.5 ”Analog Front-End” (was “PLC coupling circuitry description”).	
	Clock Generator Section 29.6.1 ”Divider and Phase Lock Loop Programming”: Added “Note that there is a delay of two SLCK clock cycles between the disable command and the real disable of the PLL” in the second paragraph.	
	Power Management Controller (PMC) Updated Section 30.10 ”Main Processor Fast Startup”. Section 30.18.9 ”PMC Clock Generator PLLA Register”: added “Unlisted values are forbidden.” to MULA bit description. Section 30.18.10 ”PMC Clock Generator PLLB Register”: added “Unlisted values are forbidden.” to MULB bit description.	
	Parallel Input/Output Controller (PIO) Section 32.5.11 ”Programmable I/O Drive”: Corrected list of configurable pads. Section 32.5.14 ”Register Write Protection”: in list of registers, replaced “PIO Pad Pull-Down Status Register” with “PIO Pad Pull-Down Enable Register”. Modified Section 32.6.48 ”PIO I/O Drive Register” (was PIO I/O Drive Register 1). Removed Section “PIO I/O Drive Register 2”.	

Doc. Rev. 43080	Comments	Change Request Ref.
B	**Serial Peripheral Interface (SPI)** Section 33.7.3 ”Master Mode Operations”: for a better description of the behavior of the TDRE and TXEMPTY flags: - Modified text regarding those flags. - Added note “When the SPI is enabled, the TDRE and TXEMPTY flags are set.” - Added Figure 33-5. Section 33.8.1 ”SPI Control Register”: modified description of bit SPIDIS. Section 33.8.5 ”SPI Status Register”: modified description of the following bits (“cleared by...”): RDRF, TDRE, ENDRX, ENDTX, RXBUFF, TXBUFE, TXEMPTY.	
	Two-wire Interface (TWI) 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”. Section 34.1 ”Description”: removed sentence: “Arbitration of the bus is performed internally and puts the TWIHS in Slave mode automatically if the bus arbitration is lost”. Section 34.7.3.4 ”Master Transmitter Mode”: Modified 2nd paragraph related to NACK. Section 34.8.6 ”TWI Status Register”: rephrased flag clearing descriptions. Corrected Figure 34-25 and Figure 34-26 (replaced EOSVACC with EOSACC). Corrected Figure 34-27 (replaced GCACC with GACC). Corrected Figure 34-32 (replaced “RXRDY= 0?” with “RXRDY= 1?”).	
	Universal Synchronous Asynchronous Receiver Transmitter (USART) Section 36.6.3.4 ”Manchester Decoder”: deleted paragraph referencing RXIDLEV bit in US_MAN register. Table 36-14, “Register Mapping” : added reset value “0x0” for US_MR, US_CSR and US_NER. Section 36.7.3 ”USART Mode Register”: updated USART_MODE and USCLKS bit descriptions. Section 36.7.7 ”USART Interrupt Disable Register”: updated OVRE bit description. Section 36.7.8 ”USART Interrupt Disable Register (SPI_MODE)” and Section 36.7.10 ”USART Interrupt Mask Register (SPI_MODE)”: updated ENDRX, ENDTX, TXBUFE and RXBUFF bit descriptions. Rephrased clearing of flags in Section 36.7.11 ”USART Channel Status Register” and Section 36.7.12 ”USART Channel Status Register (SPI_MODE)”.	
	Timer Counter (TC) Section 37.5.3 ”Interrupt Sources”: rephrased slightly the section. Section 37.7.9 ”TC Status Register”: added “(cleared on read)” to bit descriptions.	
	Segment Liquid Crystal Display Controller (SLCDC) Updated “DIS” bit descriptions.	
	Analog-to-Digital Converter (ADC) Section 40.7.11 ”ADC Interrupt Status Register”: Corrected COMPE bit name and description; changed “error” to “event”.	
	Integrity Check Monitor Section 42.6.4 ”ICM Interrupt Enable Register”: updated RSU bit description.	
	Electrical Characteristics Table 45-54, “PLC Power Consumption” : updated the values of Power Consumption.	
A	First Issue.	

Table of Contents

Description 1

1. Features 2
- 1.1 Configuration Summary 4
- 1.2 SAM4CP application block diagram 5

2. Block Diagram 6

3. Signal Description 7

4. Package and Pinout 12
- 4.1 SAM4CP16C Package and Pinout 12

5. Power Supply and Power Control 14
- 5.1 Power Supplies 14
- 5.2 Clock System 20
- 5.3 System State at Power-up 21
- 5.4 Active Mode 21
- 5.5 Low-power Modes 22
- 5.6 Wake-up Sources 26
- 5.7 Fast Start-up 26

6. Input/Output Lines 27
- 6.1 General-Purpose I/O Lines 27
- 6.2 System I/O Lines 27
- 6.3 TST Pin 28
- 6.4 NRST Pin 28
- 6.5 TMPx Pins: Anti-tamper Pins 28
- 6.6 RTCOUT0 Pin 28
- 6.7 Shutdown (SHDN) Pin 29
- 6.8 Force Wake-up (FWUP) Pin 29
- 6.9 ERASE Pin 29

7. Product Mapping and Peripheral Access 30

8. Memories 33
- 8.1 Embedded Memories 33

9. Real-time Event Management 40
- 9.1 Embedded Characteristics 40
- 9.2 Real-time Event Mapping List 40

10. System Controller 41
- 10.1 System Controller and Peripheral Mapping 41
- 10.2 Power Supply Monitoring 41
- 10.3 Reset Controller 42
- 10.4 Supply Controller (SUPC) 42

11. Peripherals 43

11.1 Peripheral Identifiers . . . 43
11.2 Peripheral DMA Controller (PDC) . . . 44
11.3 APB/AHB Bridge . . . 45
11.4 Peripheral Signal Multiplexing on I/O Lines . . . 45

12. ARM Cortex-M4 Processor . . . 50
12.1 Description . . . 50
12.2 Embedded Characteristics . . . 51
12.3 Block Diagram . . . 51
12.4 Cortex-M4 Models . . . 52
12.5 Power Management . . . 79
12.6 Cortex-M4 Instruction Set . . . 81
12.7 Cortex-M4 Core Peripherals . . . 180
12.8 Nested Vectored Interrupt Controller (NVIC) . . . 181
12.9 System Control Block (SCB) . . . 191
12.10 System Timer (SysTick) . . . 216
12.11 Memory Protection Unit (MPU) . . . 221
12.12 Floating Point Unit (FPU) . . . 243
12.13 Glossary . . . 251

13. Debug and Test Features . . . 255
13.1 Description . . . 255
13.2 Associated Documentation . . . 255
13.3 Embedded Characteristics . . . 255
13.4 Cross Triggering Debug Events . . . 257
13.5 Application Examples . . . 257
13.6 Debug and Test Pin Description . . . 258
13.7 Functional Description . . . 259

14. SAM4CP Boot Program . . . 264
14.1 Description . . . 264
14.2 Hardware and Software Constraints . . . 264
14.3 Flow Diagram . . . 264
14.4 Device Initialization . . . 264
14.5 SAM-BA Monitor . . . 265

15. Reset Controller (RSTC) . . . 268
15.1 Description . . . 268
15.2 Embedded Characteristics . . . 268
15.3 Block Diagram . . . 268
15.4 Functional Description . . . 269
15.5 Reset Controller (RSTC) User Interface . . . 274

16. Real-time Timer (RTT) . . . 279
16.1 Description . . . 279
16.2 Embedded Characteristics . . . 279
16.3 Block Diagram . . . 279
16.4 Functional Description . . . 280
16.5 Real-time Timer (RTT) User Interface . . . 281

17. Real-time Clock (RTC) 286
17.1 Description 286
17.2 Embedded Characteristics 286
17.3 Block Diagram 286
17.4 Product Dependencies 287
17.5 Functional Description 287
17.6 Real-time Clock (RTC) User Interface 295

18. Watchdog Timer (WDT) 315
18.1 Description 315
18.2 Embedded Characteristics 315
18.3 Block Diagram 315
18.4 Functional Description 316
18.5 Watchdog Timer (WDT) User Interface 317

19. Reinforced Safety Watchdog Timer (RSWDT) 322
19.1 Description 322
19.2 Embedded Characteristics 322
19.3 Block Diagram 322
19.4 Functional Description 323
19.5 Reinforced Safety Watchdog Timer (RSWDT) User Interface 324

20. Supply Controller (SUPC) 328
20.1 Description 328
20.2 Embedded Characteristics 328
20.3 Block Diagram 329
20.4 Supply Controller Functional Description 330
20.5 Register Write Protection 339
20.6 Supply Controller (SUPC) User Interface 340

21. General Purpose Backup Registers (GPBR) 352
21.1 Description 352
21.2 Embedded Characteristics 352
21.3 General Purpose Backup Registers (GPBR) User Interface 352

22. Enhanced Embedded Flash Controller (EEFC) 354
22.1 Description 354
22.2 Embedded Characteristics 354
22.3 Product Dependencies 354
22.4 Functional Description 355
22.5 Enhanced Embedded Flash Controller (EEFC) User Interface 368

23. Fast Flash Programming Interface (FFPI) 375
23.1 Description 375
23.2 Embedded Characteristics 375
23.3 Parallel Fast Flash Programming 375

24. Cortex-M Cache Controller (CMCC) 383
24.1 Description 383
24.2 Embedded Characteristics 383

24.3 Block Diagram . . . 383
24.4 Functional Description . . . 384
24.5 Cortex-M Cache Controller (CMCC) User Interface . . . 385

25. Interprocessor Communication (IPC) . . . 396
25.1 Description . . . 396
25.2 Block Diagram . . . 396
25.3 Product Dependencies . . . 397
25.4 Functional Description . . . 397
25.5 Interprocessor Communication (IPC) User Interface . . . 398

26. Bus Matrix (MATRIX) . . . 406
26.1 Description . . . 406
26.2 Embedded Characteristics . . . 406
26.3 Special Bus Granting Mechanism . . . 410
26.4 No Default Master . . . 410
26.5 Last Access Master . . . 410
26.6 Fixed Default Master . . . 410
26.7 Arbitration . . . 411
26.8 Register Write Protection . . . 413
26.9 AHB Bus Matrix (MATRIX) User Interface . . . 414

27. G3 Power Line Communications (GPLC) . . . 423
27.1 Description . . . 423
27.2 Embedded Characteristics . . . 423
27.3 Block Diagram . . . 424
27.4 Signal Description . . . 425
27.5 Analog Front-End . . . 426

28. Peripheral DMA Controller (PDC) . . . 432
28.1 Description . . . 432
28.2 Embedded Characteristics . . . 432
28.3 Block Diagram . . . 433
28.4 Functional Description . . . 433
28.5 Peripheral DMA Controller (PDC) User Interface . . . 435

29. Clock Generator . . . 446
29.1 Description . . . 446
29.2 Embedded Characteristics . . . 446
29.3 Block Diagram . . . 447
29.4 Slow Clock . . . 447
29.5 Main Clock . . . 449
29.6 Divider and PLL Blocks . . . 453

30. Power Management Controller (PMC) . . . 455
30.1 Description . . . 455
30.2 Embedded Characteristics . . . 455
30.3 Block Diagram . . . 456
30.4 Master Clock Controller . . . 457

30.5 Processor Clock Controller . . . 457
30.6 SysTick Clock. . . . 457
30.7 Peripheral Clock Controller . . . 457
30.8 Free Running Processor Clock . . . 458
30.9 Programmable Clock Output Controller . . . 458
30.10 Main Processor Fast Startup . . . 458
30.11 Main Processor Startup from Embedded Flash . . . 459
30.12 Coprocessor Sleep Mode. . . . 460
30.13 Main Clock Failure Detector. . . . 460
30.14 32.768 kHz Crystal Oscillator Frequency Monitor . . . 461
30.15 Programming Sequence . . . 461
30.16 Clock Switching Details . . . 463
30.17 Register Write Protection . . . 466
30.18 Power Management Controller (PMC) User Interface . . . 467

31. Chip Identifier (CHIPID) . . . 497
31.1 Description . . . 497
31.2 Embedded Characteristics . . . 497
31.3 Chip Identifier (CHIPID) User Interface . . . 497

32. Parallel Input/Output Controller (PIO) . . . 502
32.1 Description . . . 502
32.2 Embedded Characteristics . . . 502
32.3 Block Diagram . . . 503
32.4 Product Dependencies. . . . 504
32.5 Functional Description . . . 505
32.6 Parallel Input/Output Controller (PIO) User Interface. . . . 513

33. Serial Peripheral Interface (SPI) . . . 564
33.1 Description . . . 564
33.2 Embedded Characteristics . . . 564
33.3 Block Diagram . . . 565
33.4 Application Block Diagram . . . 565
33.5 Signal Description . . . 566
33.6 Product Dependencies. . . . 566
33.7 Functional Description . . . 567
33.8 Serial Peripheral Interface (SPI) User Interface. . . . 578

34. Two-wire Interface (TWI) . . . 594
34.1 Description . . . 594
34.2 Embedded Characteristics . . . 594
34.3 List of Abbreviations. . . . 595
34.4 Block Diagram . . . 595
34.5 I/O Lines Description . . . 595
34.6 Product Dependencies. . . . 595
34.7 Functional Description . . . 596
34.8 Two-wire Interface (TWI) User Interface . . . 620

35. Universal Asynchronous Receiver Transmitter (UART) . . . 637

35.1 Description . . . 637
35.2 Embedded Characteristics . . . 637
35.3 Block Diagram . . . 637
35.4 Product Dependencies . . . 638
35.5 Functional Description . . . 638
35.6 Universal Asynchronous Receiver Transmitter (UART) User Interface . . . 645

36. Universal Synchronous Asynchronous Receiver Transmitter (USART) . . . 657
36.1 Description . . . 657
36.2 Embedded Characteristics . . . 657
36.3 Block Diagram . . . 658
36.4 I/O Lines Description . . . 658
36.5 Product Dependencies . . . 659
36.6 Functional Description . . . 660
36.7 Universal Synchronous Asynchronous Receiver Transmitter (USART) User Interface . . . 687

37. Timer Counter (TC) . . . 719
37.1 Description . . . 719
37.2 Embedded Characteristics . . . 719
37.3 Block Diagram . . . 720
37.4 Pin List . . . 721
37.5 Product Dependencies . . . 721
37.6 Functional Description . . . 722
37.7 Timer Counter (TC) User Interface . . . 740

38. Pulse Width Modulation Controller (PWM) . . . 766
38.1 Description . . . 766
38.2 Embedded Characteristics . . . 766
38.3 Block Diagram . . . 767
38.4 I/O Lines Description . . . 767
38.5 Product Dependencies . . . 768
38.6 Functional Description . . . 768
38.7 Pulse Width Modulation Controller (PWM) User Interface . . . 775

39. Segment Liquid Crystal Display Controller (SLCDC) . . . 790
39.1 Description . . . 790
39.2 Embedded Characteristics . . . 790
39.3 Block Diagram . . . 791
39.4 I/O Lines Description . . . 792
39.5 Product Dependencies . . . 792
39.6 Functional Description . . . 794
39.7 Waveform Specifications . . . 805
39.8 Segment LCD Controller (SLCDC) User Interface . . . 805

40. Analog-to-Digital Converter (ADC) . . . 820
40.1 Description . . . 820
40.2 Embedded Characteristics . . . 820
40.3 Block Diagram . . . 821
40.4 Signal Description . . . 821

40.5 Product Dependencies . . . 821
40.6 Functional Description . . . 823
40.7 Analog-to-Digital Converter (ADC) User Interface . . . 834

41. Advanced Encryption Standard (AES) . . . 857
41.1 Description . . . 857
41.2 Embedded Characteristics . . . 857
41.3 Product Dependencies . . . 858
41.4 Functional Description . . . 858
41.5 Advanced Encryption Standard (AES) User Interface . . . 869

42. Integrity Check Monitor (ICM) . . . 889
42.1 Description . . . 889
42.2 Embedded Characteristics . . . 890
42.3 Block Diagram . . . 890
42.4 Product Dependencies . . . 890
42.5 Functional Description . . . 891
42.6 Integrity Check Monitor (ICM) User Interface . . . 903

43. Classical Public Key Cryptography Controller (CPKCC) . . . 917
43.1 Description . . . 917
43.2 Product Dependencies . . . 917
43.3 Functional Description . . . 917

44. True Random Number Generator (TRNG) . . . 918
44.1 Description . . . 918
44.2 Embedded Characteristics . . . 918
44.3 Block Diagram . . . 918
44.4 Product Dependencies . . . 918
44.5 Functional Description . . . 919
44.6 True Random Number Generator (TRNG) User Interface . . . 919

45. Electrical Characteristics . . . 926
45.1 Absolute Maximum Ratings . . . 926
45.2 Recommended Operating Conditions . . . 926
45.3 Electrical Parameters Usage . . . 927
45.4 I/O Characteristics . . . 927
45.5 Embedded Analog Peripherals Characteristics . . . 935
45.6 Embedded Flash Characteristics . . . 956
45.7 Power Supply Current Consumption . . . 957
45.8 Power On Considerations . . . 970

46. Mechanical Characteristics . . . 971
46.1 176-lead LQFP Package . . . 971
46.2 Soldering Profile . . . 972
46.3 Packaging Resources . . . 972

47. Marking . . . 973

48. Ordering Information . . . 974

49. Errata 975
49.1 Supply Controller (SUPC) 975
49.2 Parallel Input Output (PIO) Controller 975
49.3 Watchdog (WDT) / Reinforced Safety Watchdog (RSWDT) 976
49.4 Enhanced Embedded Flash Controller (EEFC) 976
49.5 Wait For Interrupt (WFI) 976
49.6 Power Supply and Power Control / Clock System 977
49.7 Power Management Controller (PMC) 977

50. Revision History 978

Table of Contents 989



Atmel | Enabling Unlimited Possibilities®



Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**

© 2016 Atmel Corporation. / Rev.: Atmel-43080I-ATPL-SAM4CP16C-Datasheet_22-Sep-16.

Atmel®, Atmel logo and combinations thereof, Enabling Unlimited Possibilities®, and others are registered trademarks or trademarks of Atmel Corporation in U.S. and other countries. ARM®, ARM Connected® logo, and others are the registered trademarks or trademarks of ARM Ltd. Other terms and product names may be trademarks of others.

DISCLAIMER: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

SAFETY-CRITICAL, MILITARY, AND AUTOMOTIVE APPLICATIONS DISCLAIMER: Atmel products are not designed for and will not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death (“Safety-Critical Applications”) without an Atmel officer's specific written consent. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Atmel products are not designed nor intended for use in military or aerospace applications or environments unless specifically designated by Atmel as military-grade. Atmel products are not designed nor intended for use in automotive applications unless specifically designated by Atmel as automotive-grade.