ARM-based Flash MCU

SAM G51 Series

DATASHEET

Description

The Atmel SAM G51 series is a member of a family of Flash microcontrollers based on the high-performance 32-bit ARM® Cortex®-M4 RISC processor with Floating Point Unit. It operates at a maximum speed of 48 MHz and features up to 256 Kbytes of Flash and up to 64 Kbytes of SRAM. The peripheral set includes one USART, two UARTs, two TWIs, one high-speed TWI, up to two SPIs, one three-channel general-purpose 16-bit timer, one RTT and one 8-channel 12-bit ADC.

The SAM G51 series is a general-purpose microcontroller with the best ratio in terms of reduced power consumption, processing power and optimized peripheral set. This enables the SAM G51 series to sustain a wide range of applications including consumer, industrial control, and PC peripherals.

The device operates from 1.62V to 2V and is available in a 49-ball WLCSP or a 100-lead LQFP package.

Features

- Core
  - ARM Cortex-M4 up to 48 MHz
  - Memory Protection Unit (MPU)
  - DSP Instructions
  - Floating Point Unit (FPU)
  - Thumb®-2 instruction set
- Memories
  - 256 Kbytes embedded Flash
  - 64 Kbytes embedded SRAM
- System
  - Embedded voltage regulator for single-supply operation
  - Power-on reset (POR) and Watchdog for safe operation
  - Quartz or ceramic resonator oscillators: 3 to 20MHz power with failure detection and 32.768kHz for RTT or device clock
  - High-precision 8/16/24MHz factory-trimmed internal RC oscillator. In-application trimming access for frequency adjustment
  - Slow clock internal RC oscillator as permanent low-power mode device clock
  - PLL range from 24 MHz to 48MHz for device clock
  - Up to 18 peripheral DMA (PDC) channels
  - 8 x 32-bit General-Purpose Backup Registers (GPBR)
  - 16 external interrupt lines
- Power consumption in active mode
  - 103 µA/MHz running Fibonacci on SRAM
- Low-power modes (typical value)
  - Wait mode 6.8 µA
  - Wake-up time 3.2 µs
- Peripherals
  - One USART with SPI Mode
  - Two 2-wire UARTs
  - Three Two-Wire Interface (TWI) modules featuring two fast mode TWI masters and one high-speed TWI slave
  - One fast SPI at up to 24Mbit/s
  - One three-channel 16-bit Timer/Counter (TC) with capture, waveform, compare and PWM modes
  - One 32-bit Real-Time Timer and Real-Time Clock (RTC)
- I/Os
  - Up to 38 I/O lines with external interrupt capability (edge or level sensitivity), debouncing, glitch filtering and on-die Series Resistor Termination. Individually Programmable Open-drain, Pull-up and pull-down resistor and Synchronous Output
  - Two up to 25-bit PIO Controllers
- Analog
  - One 8-channel 12-bit ADC, up to 800 KSps
- Package
  - 49-ball WLCSP
  - 100-lead LQFP
- Industrial temperature operating range(-40° C/+85° C)

1. Configuration Summary

Table 1-1 summarizes the configuration of the SAM G51 devices.

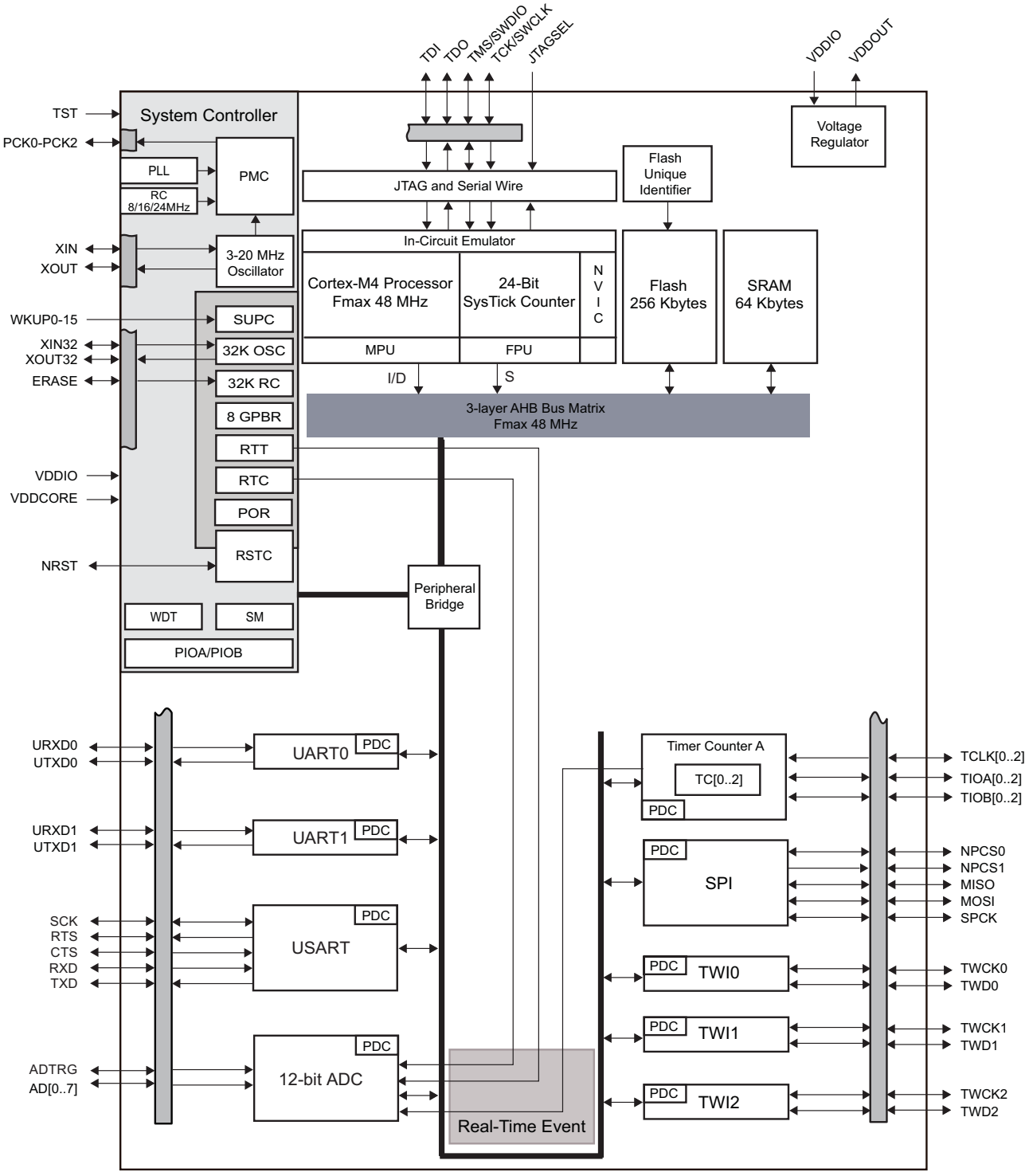
Table 1-1. Configuration Summary

Feature	SAM G51G18	SAM G51N18
Flash	256 Kbytes	256 Kbytes
SRAM	64 Kbytes	64 Kbytes
Package	WLCSP49	LQFP100
Number of PIOs	38	38
Event System	Yes	Yes
12-bit ADC	8 channels Performance: 800 KSps at 10-bit resolution 200 KSps at 11-bit resolution 50 KSps at 12-bit resolution	8 channels Performance: 600 KSps at 10-bit resolution 150 KSps at 11-bit resolution 37 KSps at 12-bit resolution
16-bit Timer	3 channels	3 channels
PDC Channels	18	18
USART/UART	1/2	1/2
SPI	2	2
TWI	2 masters 400 Kbit/s and 1 slave 3.4 Mbit/s	2 masters 400Kbits/s and 1 slave 3.4Mbit/s

Note: 1. One with SPI module + one USART configured in SPI mode.

2. SAM G51 Block Diagram

Figure 2-1. SAM G51 Block Diagram



3. Signal Description

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

Table 3-1. Signal Description List

Signal Name	Function	Type	Active Level	Voltage Reference	Comments
Power Supplies					
VDDIO	Peripherals I/O Lines, Voltage Regulator, ADC Power Supply	Power			1.62V to 2V
VDDOUT	Voltage Regulator Output	Power			1V output
VDDCORE	Core Chip Power Supply	Power			Connected externally to VDDOUT
GND	Ground	Ground			
Clocks, Oscillators and PLLs					
XIN	Main Oscillator Input	Input		VDDIO	Reset state: - PIO input - Internal pull-up disabled - Schmitt Trigger enabled
XOUT	Main Oscillator Output	Output			
XIN32	Slow Clock Oscillator Input	Input		VDDIO	
XOUT32	Slow Clock Oscillator Output	Output			
PCK0 - PCK2	Programmable Clock Output	Output			Reset state: - PIO input - Internal pull-up enabled - Schmitt Trigger enabled
ICE and JTAG					
TCK	Test Clock	Input		VDDIO	No pull-up resistor
TDI	Test Data In	Input		VDDIO	No pull-up resistor
TDO	Test Data Out	Output		VDDIO	
TRACESWO	Trace Asynchronous Data Out	Output		VDDIO	
SWDIO	Serial Wire Input/Output	I/O		VDDIO	
SWCLK	Serial Wire Clock	Input		VDDIO	
TMS	Test Mode Select	Input		VDDIO	No pull-up resistor
JTAGSEL	JTAG Selection	Input	High	VDDIO	Pull-down resistor
Flash Memory					
ERASE	Flash and NVM Configuration Bits Erase Command	Input	High	VDDIO	Pull-down (15 kΩ) resistor
Reset/Test					
NRST	Microcontroller Reset	I/O	Low	VDDIO	Pull-up resistor
TST	Test Mode Select	Input		VDDIO	Pull-down resistor
Universal Ansynchronous Receiver Transceiver - UARTx					
URXDx	UART Receive Data	Input			

Table 3-1. Signal Description List

Signal Name	Function	Type	Active Level	Voltage Reference	Comments
UTXDx	UART Transmit Data	Output			
PIO Controller - PIOA - PIOB - PIOC					
PA0 - PA24	Parallel I/O Controller A	I/O		VDDIO	Pulled-up input at reset
PB0 - PB12	Parallel I/O Controller B	I/O		VDDIO	Pulled-up input at reset
Wake-up Pins					
WKUP 0-15	Wake-up Pin / External Interrupt	I/O		VDDIO	Wake-up pins are used also as External Interrupt
Universal Synchronous Asynchronous Receiver Transmitter USART					
SCK	USART Serial Clock	I/O			
TXD	USART Transmit Data	I/O			
RXD	USART Receive Data	Input			
RTS	USART Request To Send	Output			
CTS	USART Clear To Send	Input			
Timer/Counter - TC					
TCLKx	TC Channel x External Clock Input	Input			
TIOAx	TC Channel x I/O Line A	I/O			
TIOBx	TC Channel x I/O Line B	I/O			
Serial Peripheral Interface - SPI					
MISO	Master In Slave Out	I/O			
MOSI	Master Out Slave In	I/O			
SPCK	SPI Serial Clock	I/O			High Speed Pad
NPCS0	SPI Peripheral Chip Select 0	I/O	Low		
NPCS1	SPI Peripheral Chip Select 1	Output	Low		
Two-Wire Interface- TWIx					
TWDx	TWIx Two-wire Serial Data	I/O			High Speed Pad for TWD0
TWCKx	TWIx Two-wire Serial Clock	I/O			High Speed Pad for TWDCK0
10-bit Analog-to-Digital Converter - ADCC					
AD0 - AD7	Analog Inputs	Analog			
ADTRG	ADC Trigger	Input			

4. Package and Pinout

Table 4-1. SAM G51 Packages

Device	Package
SAM G51G18	WLCSP49
SAM G51N18	LQFP100

4.1 49-ball WLCSP Pinout

Table 4-2. SAM G51G18 49-ball WLCSP Pinout

A1	PA9	B6	NRST	D4	PB10	F2	PA19/AD2
A2	GND	B7	PB12	D5	PA1	F3	PA17/AD0
A3	PA24	C1	VDDCORE	D6	PA5	F4	PA21
A4	PB8/XOUT	C2	PA11	D7	VDDCORE	F5	PA23
A5	PB9/XIN	C3	PA12	E1	PB2/AD6	F6	PA16
A6	PB4	C4	PB6	E2	PB0/AD4	F7	PA8/XOUT32
A7	VDDIO	C5	PA4	E3	PA18/AD1	G1	VDDIO
B1	PB11	C6	PA3	E4	PA14	G2	VDDOUT
B2	PB5	C7	PA0	E5	PA10	G3	GND
B3	PB7	D1	PA13	E6	TST	G4	VDDIO
B4	PA2	D2	PB3/AD7	E7	PA7/XIN32	G5	PA22
B5	JTAGSEL	D3	PB1/AD5	F1	PA20/AD3	G6	PA15
						G7	PA6

4.2 100-lead LQFP Pinout

Table 4-3. SAM G51N18 100-lead Pinout

1	NC	26	NC	51	NC	76	NC
2	NC	27	NC	52	NC	77	NC
3	NC	28	PA6	53	PA17	78	NC
4	NC	29	VDDIO	54	PA18	79	PA9
5	VDDIO	30	PA16	55	PA19	80	PB5
6	VDDIO	31	PA15	56	PA20	81	GND
7	NRST	32	PA23	57	PB0	82	GND
8	PB12	33	UNCONNECTED	58	PB1	83	GND
9	PA4	34	UNCONNECTED	59	PB2	84	PB6
10	PA3	35	PA22	60	PB3	85	PB7
11	PA0	36	PA21	61	VDDIO	86	PA24
12	PA1	37	VDDIO	62	PA14	87	PB8
13	PA5	38	VDDIO	63	PA13	88	PB9
14	VDDIO	39	GND	64	PA12	89	VDDIO
15	VDDCORE	40	GND	65	PA11	90	PA2
16	VDDCORE	41	GND	66	VDDCORE	91	PB4
17	TEST	42	GND	67	VDDCORE	92	PB4
18	PA7	43	GND	68	PB10	93	JTAGSEL
19	PA8	44	VDDOUT	69	PB11	94	VDDIO
20	GND	45	VDDOUT	70	GND	95	VDDIO
21	NC	46	VDDIO	71	GND	96	NC
22	NC	47	VDDIO	72	PA10	97	NC
23	NC	48	VDDIO	73	NC	98	NC
24	NC	49	NC	74	NC	99	NC
25	NC	50	NC	75	NC	100	NC

5. Power Considerations

5.1 Power Supplies

The SAM G51 devices feature the following power supply pins:

- VDDCORE pins: Power the core, including the processor, the embedded memories and the peripherals. VDDCORE must be connected to VDDOUT.
- VDDIO pins: Power the peripheral I/O lines, voltage regulator, ADC power supply; voltage ranges from 1.62V to 2V for voltage regulator, ADC.

The ground pins GND are common to VDDCORE and VDDIO.

5.2 Voltage Regulator

The SAM G51 devices embed a core voltage regulator that is managed by the Supply Controller and that supplies the Cortex-M4 core, internal memories (SRAM, ROM and Flash logic) and the peripherals. The voltage regulator supplies up to 25 mA and features a quiescent current less than 1.3 µA. An internal adaptative biasing adjusts the regulator quiescent current depending on the required load current.

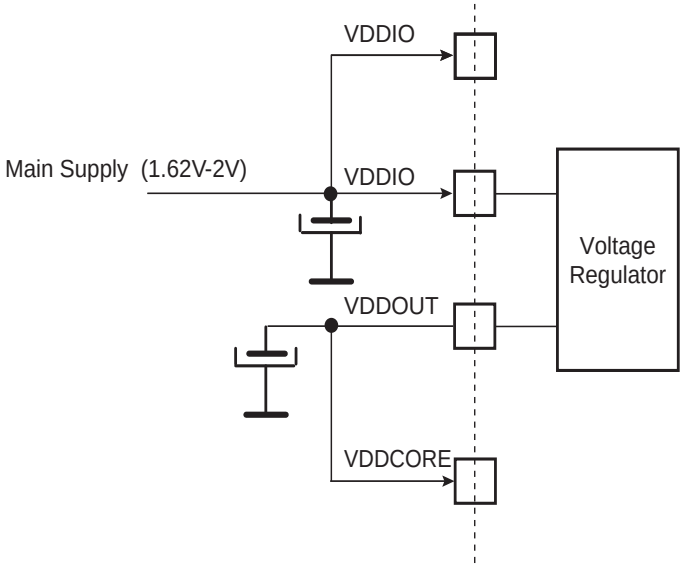
For adequate input and output power supply decoupling/bypassing, refer to information provided on the VDDCORE voltage regulator in the Electrical Characteristics section of the datasheet.

5.3 Typical Powering Schematics

The SAM G51 devices support a 1.62V-2V single supply mode. The internal regulator input is connected to the source and its output feeds VDDCORE. Figure 5-1 shows the power schematics.

To achieve system performance, the internal regulator must be used.

Figure 5-1. Single Supply



5.4 Functional Modes

5.4.1 Active Mode

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

5.4.2 Wait Mode

The purpose of wait mode is to achieve very low power consumption while maintaining the entire device in a powered state for a wake-up time of less than 5 µs. The wake-up time is achieved while the system is running from the internal SRAM. The function allowing to enter and exit the wait mode linked and executed in the internal SRAM. If the wake-up function is executed in internal Flash, the wake-up time is 70 µs (for C code running in Flash, the number of wait states must be 0).

The current consumption in wait mode is typically less than 10 µA (total current consumption). The clocks of the core, the peripherals and memories are stopped. However, power supplies are still maintained, thus allowing memory retention, CPU context saving and fast start-up. Wait mode is entered by setting the WAITMODE bit to 1 in the CKGR_MOR register in conjunction with FLPM = 0 or FLPM = 1 bits of the PMC_FSMR register. or by the Wait for Event (WFE) instruction.

Note: The WFE instruction can add complexity in application state machines. This is because the WFE instruction goes along with an event flag of the Cortex processor (cannot be managed by the software application). The event flag can be set by interrupts, a debug event or an event signal from another processor. Since an interrupt may take place just before the execution of WFE, WFE takes into account events that happened in the past. As a result, WFE prevents the device from entering wait mode if an interrupt event has occurred. To work around this complexity, the WAITMODE bit in the PMC Clock Generator Main Oscillator Register of the Power Management Controller (PMC) can be used.

The Cortex-M4 processor is able to handle external or internal events in order to wake up the core. This is done by configuring the external lines WKUP0-15 as fast start-up wake-up pins (refer to Section 5.5 “Fast Start-up”) or the RTC, RTT alarms for internal events.

To enter wait mode with WAITMODE bit:

- Select the 8/16/24 MHz fast RC oscillator as Main Clock. If 24 MHz is selected and the C code runs on the SRAM, wake-up time is less than 5 µs.
- Set the FLPM field in the PMC Fast Start-up Mode Register (PMC_FSMR).
- Set Flash Wait State to 0.
- Set the WAITMODE bit = 1 in PMC Main Oscillator Register (CKGR_MOR).
- Wait for Master Clock Ready MCKRDY = 1 in the PMC Status Register (PMC_SR).

To enter wait mode with WFE:

- Select the 8/16/24 MHz fast RC oscillator as Main Clock. If 24 MHz is selected and the C code runs on the SRAM, wake-up time is less than 5 µs.
- Set the FLPM field in the PMC Fast Start-up Mode Register (PMC_FSMR).
- Set Flash Wait State to 0.
- Set the LPM bit in the PMC Fast Startup Mode Register (PMC_FSMR).
- Execute the Wait-For-Event (WFE) instruction of the processor.

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

- FLPM = 0 in stand-by mode (Low power consumption)
- FLPM = 1 in deep power-down mode (Extra-low power consumption)
- FLPM = 2 in idle mode. Memory ready for read access.

5.4.3 Sleep Mode

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

Sleep mode is entered via Wait for Interrupt (WFI).

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

Table 5-1 summarizes the power consumption, wake-up time and system state in wait and in sleep modes.

Table 5-1. Low Power Mode Configuration Summary

Mode	SUPC, 32 kHz Oscillator RTC RTT POR Regulator	POR Supply Monitor on VDDIO	Core Memory Peripherals	Mode Entry	Potential Wake-up Sources	Core at Wake-up	PIO State While in Low Power Mode	PIO State at Wake-up	Consumption (2) (3)	Wake-up Time(1)
Wait Mode w/Flash in Deep Power Down Mode	ON	OFF	Powered (Not clocked)	WAITMODE = 1 + FLPM = 1 or WFE + SLEEPDEEP = 0 + LPM = 1 + FLPM = 1	Any event from: Fast start-up through WUP0-15 pins RTC alarm RTT alarm	Clocked back	Previous state saved	Unchanged	<10 µA (5)	< 5 µs(4)
Sleep Mode	ON	ON	Powered(6) (Not clocked)	WFI +SLEEPDEEP bit = 0 +LPM bit = 0	Entry mode =WFI Interrupt Only; Any Enabled Interrupt	Clocked back	Previous state saved	Unchanged	(7)	(7)

Notes:
1. When considering wake-up time, the time required to start the PLL is not taken into account. Once started, the device works with the 8/16/24 MHz Fast RC oscillator. The user has to add the PLL wake-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.
2. The external loads on PIOs are not taken into account in the calculation.
3. BOD current consumption is not included.
4. Wake-up from RAM.
5. 10 µA is for typical conditions.
6. Depends on MCK frequency.
7. In this mode, the core is supplied and not clocked. Some peripherals can be clocked.

5.5 Fast Start-up

The SAM G51 devices allow the processor to restart in a few microseconds while the processor is in wait mode. A fast start-up can occur upon detection of a low level on one of the 17 wake-up inputs.

The fast restart circuitry, as shown in Figure 5-2, 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 restarts from the last fast RC selected (the embedded 24 MHz fast RC oscillator), switches the master clock on the last clock of RC oscillator and reenables the processor clock. At wake-up of the wait mode, the code is executed in the SRAM.

Figure 5-2. Fast Start-up Source

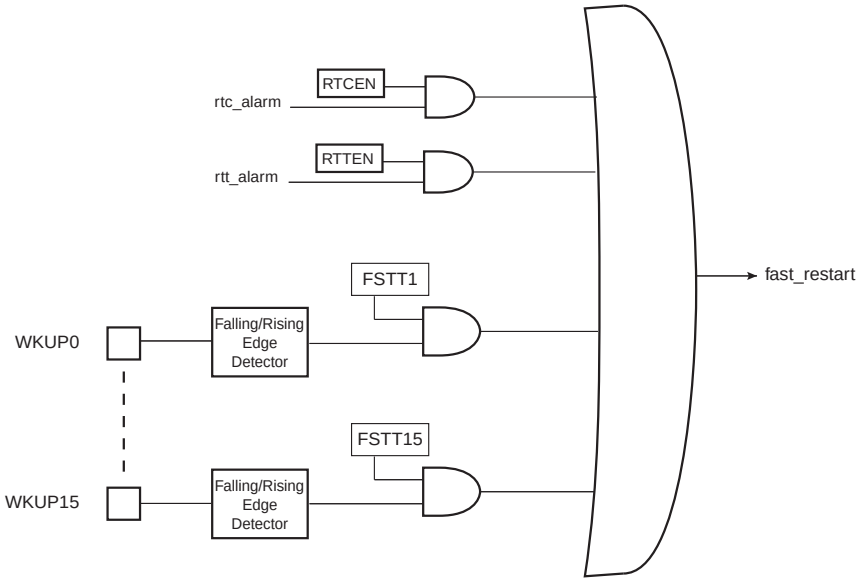
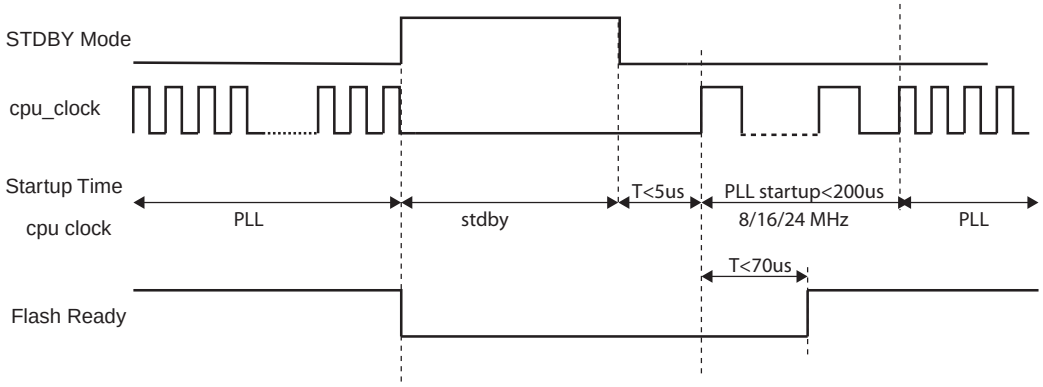


Figure 5-3. Start-up Sequence



Reset signals resynchronised on specific clocks

6. Processor and Architecture

6.1 ARM Cortex-M4 Processor

- Thumb-2 (ISA) subset consisting of all base Thumb-2 instructions, 16-bit and 32-bit
- Harvard processor architecture enabling simultaneous instruction fetch with data load/store
- Three-stage pipeline
- Single-cycle 32-bit multiply
- Hardware divide
- Thumb and debug states
- Handler and thread modes
- Low-latency ISR entry and exit
- Memory Protection Unit (MPU)
- Floating Point Unit(FPU)

6.2 APB/AHB Bridge

The SAM G51 devices embed one peripheral bridge. The peripherals of the bridge are clocked by MCK.

6.3 Peripheral DMA Controller

The Peripheral DMA Controller handles transfer requests from the channel according to the following priorities (CH0 is high priority):

Table 6-1. Peripheral DMA Controller

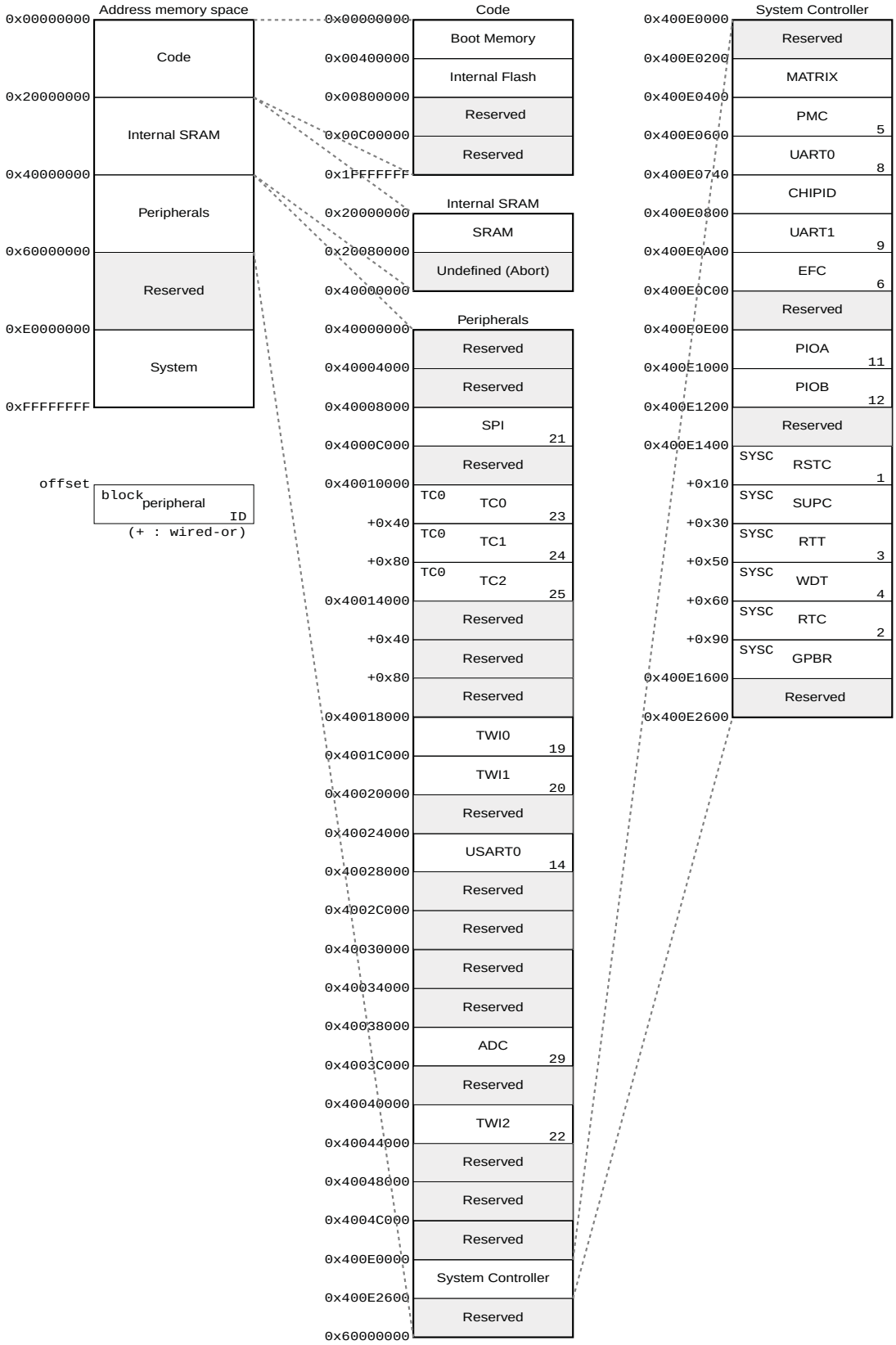
Instance Name	Channel T/R	Channel NR
MEM2MEM	Transmit	17
SPI	Transmit	16
TWI1	Transmit	15
TWI2	Transmit	14
UART0	Transmit	13
UART1	Transmit	12
USART	Transmit	11
TWI0	Transmit	10
MEM2MEM	Receive	9
TC0:TC2	Receive	8
SPI	Receive	7
TWI1	Receive	6
TWI2	Receive	5
UART0	Receive	4
UART1	Receive	3
USART	Receive	2
ADC	Receive	1
TWI0	Receive	0

6.4 Debug and Test Features

- Debug access to all memories and registers in the system, including Cortex-M4 register bank when the core is running, halted, slept or held in reset
- Serial Wire Debug Port (SW-DP) and Serial Wire JTAG Debug Port (SWJ-DP) debug access
- Flash Patch and Breakpoint (FPB) unit for implementing breakpoints and code patches
- Data Watchpoint and Trace (DWT) unit for implementing watchpoints, data tracing, and system profiling
- Instrumentation Trace Macrocell (ITM) for support of printf style debugging
- IEEE1149.1 JTAG boundary scan on all digital pins

6.5 Product Mapping

Figure 6-1. SAM G51 Series Product Mapping



7. Memories

7.1 Embedded Memories

7.1.1 Internal SRAM

The SAM G51 devices embed a total of 64 Kbytes of high-speed SRAM.

The SRAM is accessible over the Cortex-M4 bus at address 0x2000 0000.

The SRAM is in the bit band region. The bit band alias region is from 0x2200 0000 and 0x23FF FFFF.

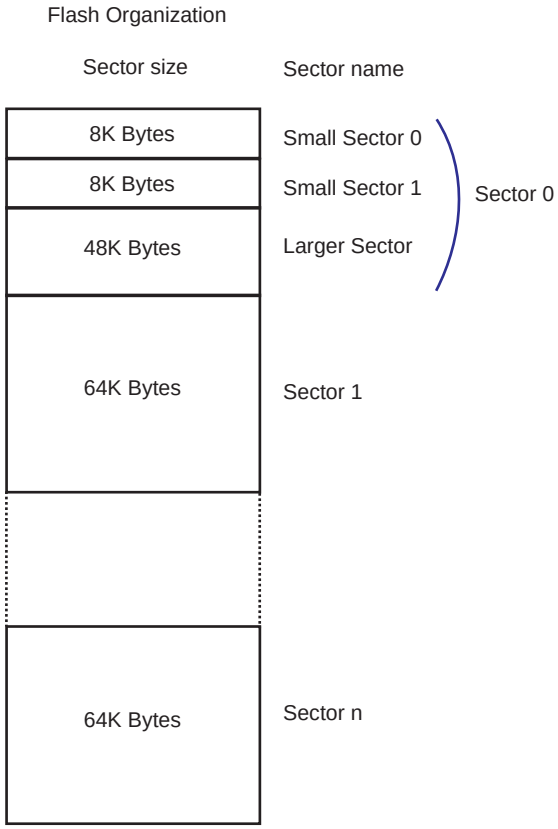
7.1.2 Embedded Flash

7.1.2.1 Flash Overview

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

The three smaller sectors are comprised of 2 sectors of 8 Kbytes and 1 sector of 48 Kbytes. Refer to Figure 7-1, "Global Flash Organization".

Figure 7-1. Global Flash Organization



Each sector is organized in pages of 512 bytes.

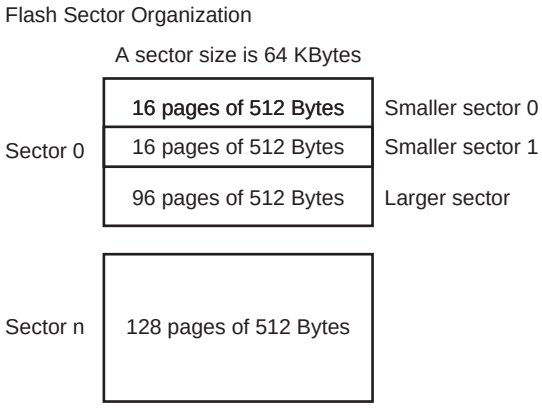
For sector 0:

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

From sector 1 to n:

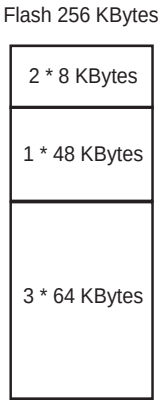
The rest of the array is composed of 64 Kbytes sectors of 128 pages of 512 bytes each. Refer to Figure 7-2, "Flash Sector Organization".

Figure 7-2. Flash Sector Organization



The SAM G51 devices Flash size is 256 Kbytes. Refer to Figure 7-3, "Flash Size" for the organization of the Flash.

Figure 7-3. Flash Size



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

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

The 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.

7.1.2.2 Enhanced Embedded Flash Controller

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

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

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

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

7.1.2.3 Flash Speed

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

For more details, refer to the AC characteristics in the Electrical Characteristics section.

7.1.2.4 Lock Regions

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

Table 7-1. Lock Bit Number

Product	Number of Lock Bits	Lock Region Size
SAM G51	32	8 Kbytes

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

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

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

7.1.2.5 Security Bit

The SAM G51 devices feature a security bit, based on a specific General Purpose NVM bit (GPNVM bit 0). When the security is enabled, any access to the Flash, SRAM, core registers and internal peripherals either through the ICE interface 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 and internal peripherals are permitted.

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

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

7.1.2.6 Calibration Bits

The GPNVM bits are used to calibrate the POR, the voltage regulator and RC 8/16/24. These bits are factory configured and cannot be changed by the user. The ERASE pin has no effect on the calibration bits.

7.1.2.7 Unique Identifier

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

7.1.2.8 User Signature

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

Read, write and erase of this area is allowed.

8. System Controller

The System Controller is a set of peripherals that allow handling of key elements of the system, such as power, resets, clocks, time, interrupts, watchdog, etc. Refer to the section on the Supply Controller (SUPC).

8.1 System Controller and Peripherals Mapping

Refer to Section 6-1 “SAM G51 Series Product Mapping”.

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

8.2 Power-on Reset, Supply Monitor

The SAM G51 devices embed three features to monitor, warn and/or reset the chip:

- Power-on reset on VDDIO
- Power-on reset on VDDCORE
- Supply monitor on VDDIO

8.3 Reset Controller

The Reset Controller is based on a power-on reset cell. The Reset Controller returns the source of the last reset to software: general reset, wake-up reset, software reset, user reset or watchdog reset.

The Reset Controller controls the internal resets of the system and the input/output of the NRST pin. It shapes a reset signal for the external devices, simplifying the connection of a push-button on the NRST pin to implemement a manual reset. By default, the NRST pin is configured as an input.

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

8.4 Supply Controller

The Supply Controller controls the power supplies of each section of the processor and the peripherals (via voltage regulator control).

The Supply Controller has its own reset circuitry and is clocked by the 32 kHz slow clock generator.

The reset circuitry is based on a zero-power power-on reset cell and a POR (power-on reset) cell. The zero-power power-on reset allows the Supply Controller to start properly, while the software-programmable POR allows detection of either a battery discharge or main voltage loss.

The slow clock generator is based on a 32 kHz crystal oscillator and an embedded 32 kHz RC oscillator. The user can also set the crystal oscillator in bypass mode instead of connecting a crystal. In this case, the user has to provide the external clock signal on XIN32. The slow clock defaults to the RC oscillator, but the software can enable the crystal oscillator and select it as the slow clock source.

The Supply Controller 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 enables to set the system in different low power modes and to wake it up from a wide range of events.

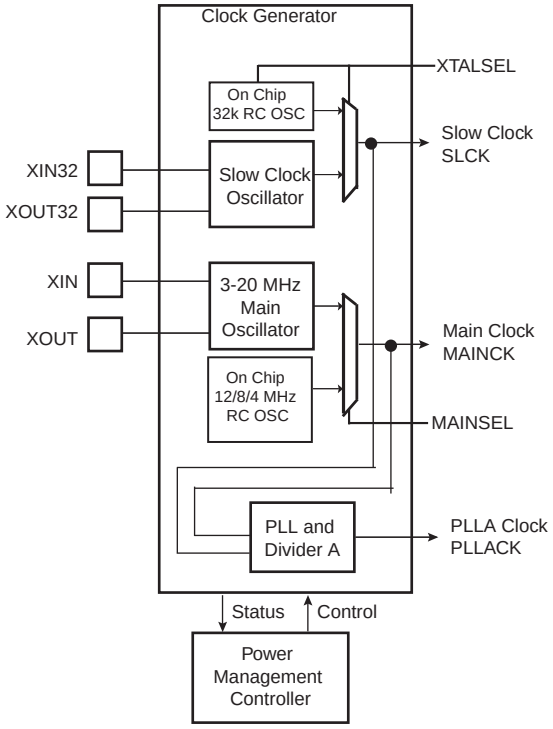
The threshold value of the voltage regulator can be adjusted by the VRVDD bitfield in SUPC_MR register. Refer to Supply Controller Mode Register details in the section on the Supply Controller (SUPC).

8.5 Clock Generator

The Clock Generator is made up of:

- One low-power 32768Hz slow clock oscillator with bypass mode
- One low-power RC oscillator
- One factory-programmed fast RC oscillator with three selectable output frequencies: 8, 16 or 24 MHz. At startup, 8 MHz is selected
- One 24 to 48 MHz programmable PLL that provides the clock MCK to the processor and to the peripherals. The PLL has an input divider to offer a wider range of output frequencies from the main clock input

Figure 8-1. Clock Generator Block Diagram



The RC32K is measured at ambient temperature during chip test and its value is stored in the Flash signature page. The frequency accuracy of the 32 kHz RC oscillator in provided in the Electrical Characteristics section.

When the application uses the RC32K, the value of the frequency accuracy of RC32K must be read and included in the API to use the RC32K at 32 kHz.

8.6 Power Management Controller

The Power Management Controller provides all the clock signals to the system:

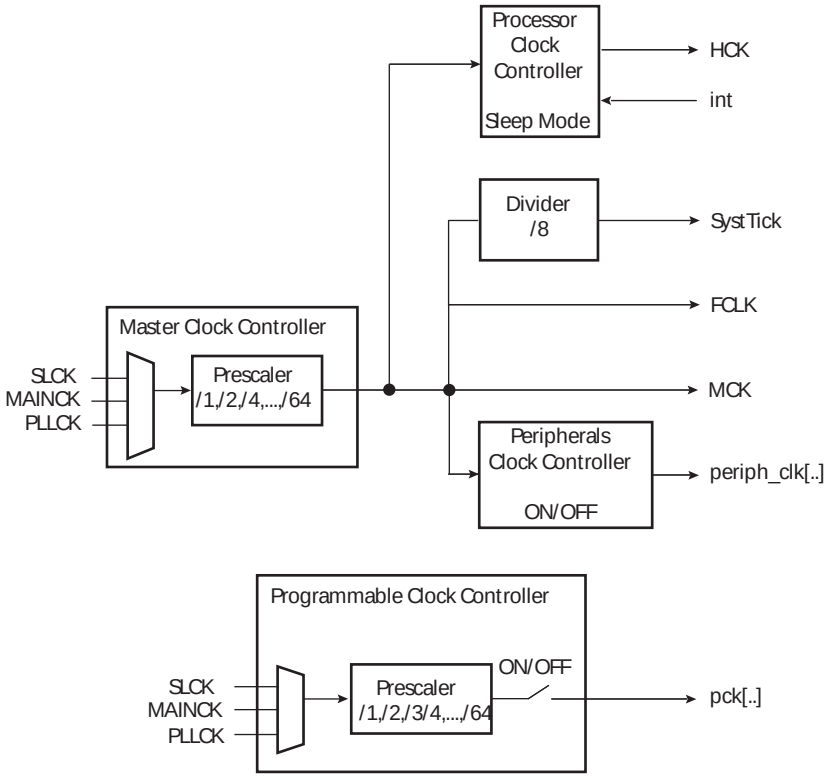
- Processor clock HCLK
- Free-running processor clock FCLK
- The Cortex SysTick external clock
- The master clock MCK, in particular to the matrix and the memory interfaces
- Independent peripheral clocks, typically at the frequency of MCK
- Three programmable clock outputs: PCK0, PCK1 and PCK2

The Supply Controller chooses between the 32 kHz RC oscillator or the crystal oscillator. The unused oscillator is disabled automatically so that power consumption is optimized.

By default, at start-up the chip runs out of the master clock using the fast RC oscillator running at 8 MHz.

The user can trim the 8, 16 and 24 MHz RC oscillator frequency by software.

Figure 8-2. SAM G51 Power Management Controller Block Diagram



The SysTick calibration value is fixed to 6000 which allows the generation of a time base of 1 ms with SystTick clock to MHz (max HCLK 48 MHz/8 = 6000, so STCALIB = 0x1770)

8.7 Watchdog Timer

- 16-bit key-protected only-once-programmable counter
- Windowed, prevents the processor to be in a dead-lock on the watchdog access

8.8 SysTick Timer

- 24-bit down counter
- Self-reload capability
- Flexible system timer

8.9 Real-Time Timer

- Real-Time Timer, allowing backup of time with different accuracies
  - 32-bit free-running backup counter
  - Integrates a 16-bit programmable prescaler running on slow clock
  - Alarm register generates a wake-up of the system through the Shutdown Controller
  - Wake-up from wait mode through the Power Management Controller

8.10 Real Time Clock

- Low power consumption
- Full asynchronous design
- Two-hundred-year calendar
- Programmable periodic interrupt
- Alarm and update parallel load
- Control of alarm and update Time/Calendar Data In

8.11 General-Purpose Backup Registers

- Eight 32-bit backup general-purpose registers

8.12 Nested Vectored Interrupt Controller

- Forty-seven maskable interrupts, external to NVIC
- Sixteen priority levels
- Dynamic reprioritization of interrupts
- Priority grouping
  - Selection of preempting interrupt levels and non-preempting interrupt levels
- Support for tail-chaining and late arrival of interrupts
  - Back-to-back interrupt processing without the overhead of state saving and restoration between interrupts
- Processor state automatically saved on interrupt entry, and restored on interrupt exit, with no instruction overhead

8.13 Chip Identification

- Chip Identifier (CHIPID) registers permit recognition of the device and its revision

Table 8-1. SAM G51 Chip IDs Register

Chip Name	CHIPID_CIDR	CHIPID_EXID
SAM G51G18	0x243B_09E0	0x0
SAM G51N18	0x243B_09E8	0x0

- JTAG ID: 0x05B3_A03F

8.14 PIO Controllers

- Two PIO Controllers, PIOA and PIOB that control a maximum of 25 I/O lines
- Fully programmable through Set/Clear registers

Table 8-2. PIO Lines Available Depending on Pin Count

Version	49 Pins	100 Pins
PIOA	25	25
PIOB	13	13

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

8.15 Peripheral Identifiers

Table 8-3 defines the peripheral identifiers of the SAM G51 devices. A peripheral identifier is required for the control of the peripheral interrupts with the Nested Vectored Interrupt Controller and for the control of the peripheral clock with the Power Management Controller. The external interrupts are connected to WKUP pins.

Table 8-3. 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 Flash Controller
7	—	—	—	Reserved
8	UART0	X	X	UART 0
9	UART1	X	X	UART 1
10	—	—	—	Reserved
11	PIOA	X	X	Parallel I/O Controller A
12	PIOB	X	X	Parallel I/O Controller B
13	—	—	—	Reserved
14	USART	X	X	USART
15	MEM2MEM	X	X	MEM2MEM
16	—	—	—	Reserved
17	—	—	—	Reserved
18	—	—	—	Reserved
19	TWI0	X	X	Two-Wire Interface 0 HS
20	TWI1	X	X	Two-Wire Interface 1
21	SPI	X	X	Serial Peripheral Interface
22	TWI2	X	X	Two Wire Interface 2
23	TC0	X	X	Timer/Counter 0
24	TC1	X	X	Timer/Counter 1
25	TC2	X	X	Timer/Counter 2
26	—	—	—	Reserved
27	—	—	—	Reserved
28	—	—	—	Reserved
29	ADC	X	X	Analog-to-Digital Converter
30	ARM	X	—	FPU
31	WKUP0	X	—	External interrupt 0

Table 8-3. Peripheral Identifiers (Continued)

Instance ID	Instance Name	NVIC Interrupt	PMC Clock Control	Instance Description
32	WKUP1	X	—	External interrupt 1
33	WKUP2	X	—	External interrupt 2
34	WKUP3	X	—	External interrupt 3
35	WKUP4	X	—	External interrupt 4
36	WKUP5	X	—	External interrupt 5
37	WKUP6	X	—	External interrupt 6
38	WKUP7	X	—	External interrupt 7
39	WKUP8	X	—	External interrupt 8
40	WKUP9	X	—	External interrupt 9
41	WKUP10	X	—	External interrupt 10
42	WKUP11	X	—	External interrupt 11
43	WKUP12	X	—	External interrupt 12
44	WKUP13	X	—	External interrupt 13
45	WKUP14	X	—	External interrupt 14
46	WKUP15	X	—	External interrupt 15

8.16 Peripherals Signals Multiplexing on I/O Lines

The SAM G51 devices feature two PIO (49-ball) controllers, PIOA and PIOB, which multiplex the I/O lines of the peripherals set.

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

Note that some peripheral functions, which are output only, may be duplicated within both tables.

8.16.1 PIO Controller A Multiplexing

Table 8-4. Multiplexing on PIO Controller A (PIOA)

I/O Line	Peripheral A	Peripheral B	Extra Function	System Function
PA0	—	TIOA0	WKUP0	—
PA1	—	TIOB0	WKUP1	—
PA2	TCLK0	—	WKUP2	—
PA3	TWD0	—	—	—
PA4	TWCK0	—	—	—
PA5	RXD	—	WKUP4	—
PA6	TXD	PCK0	—	—
PA7	—	—	—	XIN32
PA8	—	ADTRG	WKUP5	XOUT32
PA9	URXD0	NPCS1	WKUP6	—
PA10	UTXD0	—	—	—
PA11	NPCS0	—	WKUP7	—
PA12	MISO	—	—	—
PA13	MOSI	—	—	—
PA14	SPCK	—	WKUP8	—
PA15	RTS	SCK	—	—
PA16	CTS	TIOB1	—	—
PA17	—	PCK1	AD0	—
PA18	—	PCK2	AD1	—
PA19	TCLK1	—	AD2	—
PA20	TCLK2	—	AD3	—
PA21	TIOA2	PCK1	WKUP9	—
PA22	TIOB2	—	WKUP10	—
PA23	—	TIOA1	WKUP3	—
PA24	—	—	WKUP11	—

8.16.2 PIO Controller B Multiplexing

Table 8-5. Multiplexing on PIO Controller B (PIOB)

I/O Line	Peripheral A	Peripheral B	Extra Function	System Function
PB0	—	TWD2	AD4	—
PB1	—	TWCK2	AD5	—
PB2	URXD1	NPCS1	AD6/WKUP12	—
PB3	UTXD1	PCK2	AD7/WKUP13	—
PB4	—	—	—	TDI
PB5	—	—	—	TDO/ TRACESWO
PB6	—	—	—	TMS/SWDIO
PB7	—	—	—	TCK/SWCLK
PB8	TWD1	—	WKUP14	XOUT
PB9	TWCK1	—	WKUP15	XIN
PB10	TWD1(1)	TWD2(1)	—	
PB11	TWCK1(1)	TWCK2(1)	—	
PB12	—	—	—	ERASE

Note: 1. Each TWI (TWI1,TWI2) can be routed on two different pairs of IOs. TWI1 and TWI2 share one pair of IOs (PB10 and PB11). The configuration of the shared IOs determine which TWI is selected.

8.16.2.1 TWI Muxing on PB10 and PB11

The selection of the TWI used in PB10 and PB11 is determined by the configuration of PB10 and PB11. Three modes are possible: normal mode, alternative mode TWI1 and alternative mode TWI2.

Normal mode is:

- TWI1 only used: PB09 and PB08 must be configured as PIO Peripheral A
- TWI2 only used: PB00 and PB01 must be configured as PIO Peripheral B
- TWI1 and TWI2 used: PB09 and PB08 must be configured as PIO Peripheral A and PB00 and PB01 must be configured as PIO Peripheral B

Alternative mode TWI1 is:

- TWI1 is muxing on PB10 and PB11: PB10 and PB11 must be configured as PIO Peripheral A. PB8 and PB9 can be configured as GPIO, WKUP pin or XIN, XOUT. PB8 and PB9 cannot be used as peripherals

Alternative mode TWI2 is:

- TWI2 is muxing on PB10 and PB11: PB10 and PB11 must be configured as PIO Peripheral B. PB0 and PB1 can be configured as GPIO, analog input. PB0 and PB1 cannot be used as peripherals

Alternative Mode TWI1 example:

PB10 is driven by TWD1 signal if the PB10 is configured as peripheral (PIO_PSR[10]=1 and PIO_ABCDSR1[10]=PIO_ABCDSR2[10]=0).

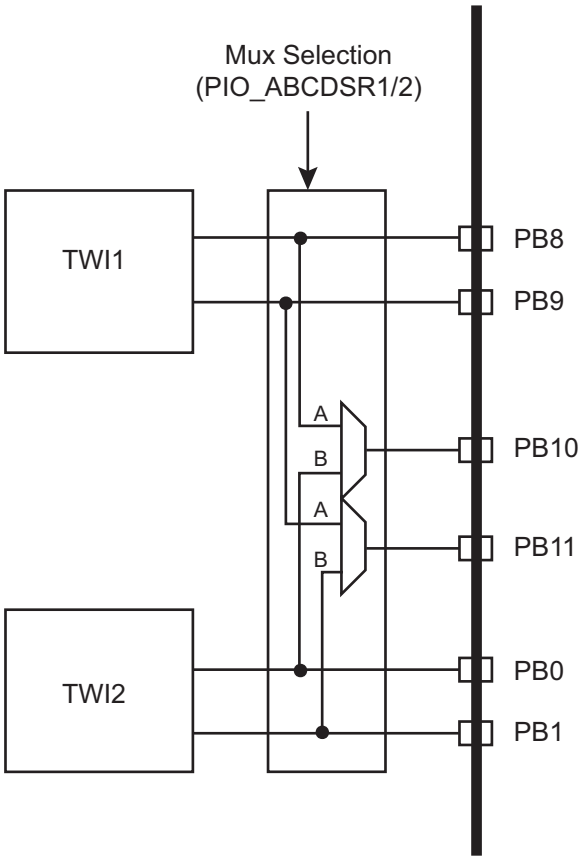
PB11 is driven by TWCK1 signal if the PB11 is configured as peripheral (PIO_PSR[11]=1 and PIO_ABCDSR1[11]=PIO_ABCDSR2[11]=0).

Table 8-6. TWI Multiplexing

Requirement Configuration	PIO Config		PB0	PB1	PB8	PB9	PB10	PB11
	Configuration of PB10	Configuration of PB11						
Normal Mode TWI1 and/or TWI2 Used	Enable PIO_PER[10]	Enable PIO_PER[11]	TWD2	TWCK2	TWD1	TWCK1	GPIO	GPIO
Alternative Mode TWI1	Config PB10 as Peripheral A	Config PB11 as Peripheral A	TWD2	TWCK2	GPIO or WKUP pin or XOUT(1)	GPIO or WKUP pin or XIN(1)	TWD1	TWCK1
Alternative Mode TWI2	Config PB10 as Peripheral B	Config PB10 as Peripheral B	GPIO or AD Input(1)	GPIO or AD Input(1)	TWD1	TWCK1	TWD2	TWCK2

Note: 1. Configuration of PBx can be done after the configuration of PB10 and PB11.

Figure 8-3. TWI Master PIO Muxing Selection



9. Real-Time Event Management

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

9.1 Embedded Characteristics

- Timers, IOs and RTC peripherals generate event triggers which are directly routed to event managers, such as the ADC, to start measurement/conversion without processor intervention.
- UART, USART, SPI, TWI and ADC also generate event triggers directly connected to the Peripheral DMA Controller (PDC) for data transfer without processor intervention.
- PMC security events (clock failure detection) can be programmed to switch the MCK on a reliable main RC internal clock without processor intervention.

9.2 Real-Time Event Mapping

Table 9-1. Real-time Event Mapping List

Event Generator	Event Manager	Function
IO (WKUP0/1)	General Purpose Backup Register (GPBR)	Security / Immediate GPBR clear (asynchronous) on tamper detection through WKUP0/1 IO pins
Power Management Controller (PMC)	PMC	Safety / Automatic switch to reliable main RC oscillator in case of main crystal clock failure
IO (ADTRG)	Analog-to-Digital Converter (ADC)	Trigger for measurement. Selection in ADC module.
TC Output 0	ADC	Trigger for measurement. Selection in ADC module.
TC Output 1	ADC	Trigger for measurement. Selection in ADC module.
TC Output 2	ADC	Trigger for measurement. Selection in ADC module.
RTCOUT0	ADC	Trigger for measurement. Selection in ADC module.
RTCOUT1	ADC	Trigger for measurement. Selection in ADC module.
RTTINC	ADC	Trigger for measurement. Selection in ADC module.
UART	PDC	Triggers 1 word transfer
USART	PDC	Triggers 1 word transfer
TWI0/1/2	PDC	Triggers 1 word transfer
ADC	PDC	Triggers 1 word transfer
Timer Counter	PDC	Triggers 1 word transfer
SPI	PDC	Triggers 1 word transfer

10. ARM Cortex-M4 Processor

10.1 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.

10.1.1 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.

10.1.2 Integrated Configurable Debug

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

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

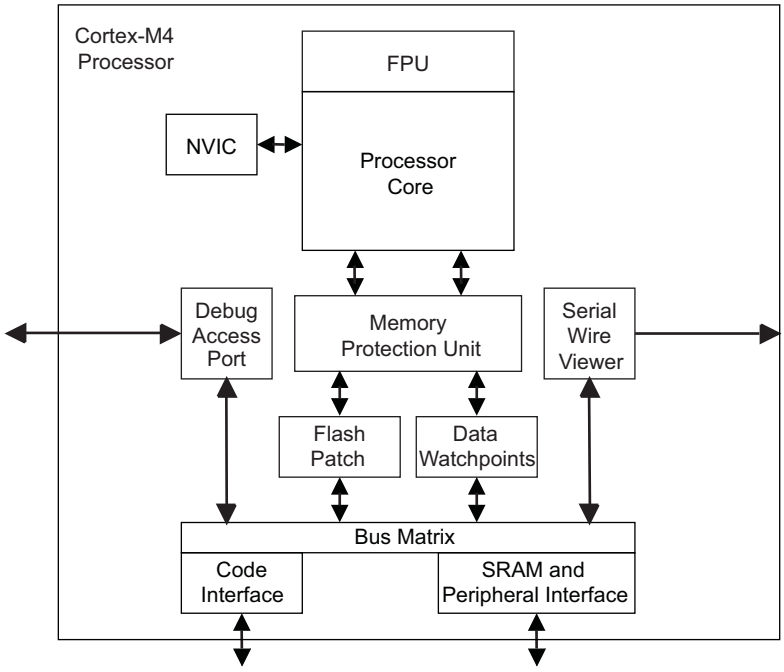
The Flash Patch and Breakpoint Unit (FPB) provides up to eight hardware breakpoint comparators that debuggers can use. The comparators in the FPB also provide remap functions of up to eight words in the program code in the CODE memory region. This enables applications stored on a non-erasable, ROM-based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

10.2 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.

10.3 Block Diagram

Figure 10-1. Typical Cortex-M4 Implementation



10.4 Cortex-M4 Models

10.4.1 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.

10.4.1.1 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.

10.4.1.2 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 10-1. Summary of processor mode, execution privilege level, and stack use options

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

Note: 1. See “Control Register” .

10.4.1.3 Core Registers

Figure 10-2. Processor Core Registers

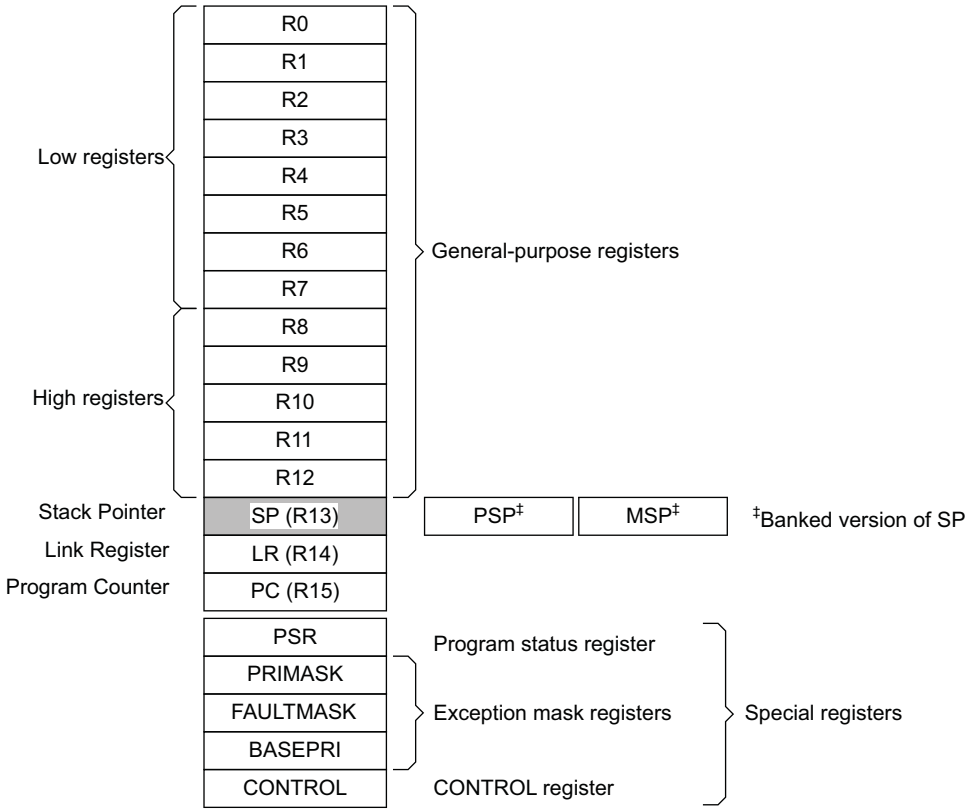


Table 10-2. Core Processor Registers

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.

10.4.1.4 General-purpose Registers

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

10.4.1.5 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.

10.4.1.6 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.

10.4.1.7 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.

10.4.1.8 Program Status Register

Name: PSR

Access: Read/Write

Reset: 0x000000000

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

The *Program Status Register* (PSR) combines:

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

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

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

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

The PSR combinations and attributes are:

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

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

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

10.4.1.9 Application Program Status Register

Name: APSR

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
N	Z	C	V	Q	–		

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

15	14	13	12	11	10	9	8
–							

7	6	5	4	3	2	1	0
–							

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

- **N: Negative Flag**

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

1: Operation result was negative or less than.

- **Z: Zero Flag**

0: Operation result was not zero

1: Operation result was zero.

- **C: Carry or Borrow Flag**

Carry or borrow flag:

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

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

- **V: Overflow Flag**

0: Operation did not result in an overflow

1: Operation resulted in an overflow.

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

Sticky saturation flag:

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

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

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

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

See “SEL” for more information.

10.4.1.10 Interrupt Program Status Register

Name: IPSR

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–							ISR_NUMBER

7	6	5	4	3	2	1	0
ISR_NUMBER							

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

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

0 = Thread mode

1 = Reserved

2 = NMI

3 = Hard fault

4 = Memory management fault

5 = Bus fault

6 = Usage fault

7–10 = Reserved

11 = SVCall

12 = Reserved for Debug

13 = Reserved

14 = PendSV

15 = SysTick

16 = IRQ0

61 = IRQ46

See “Exception Types” for more information.

10.4.1.11 Execution Program Status Register

Name: EPSR

Access: Read/Write

Reset: 0x000000000

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

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
ICI/IT						–	

7	6	5	4	3	2	1	0
–							

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

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

- **ICI: Interruptible-continuable Instruction**

When an interrupt occurs during the execution of an LDM, STM, PUSH, POP, VLDM, VSTM, VPUSH,

or VPOP instruction, the processor:

– Stops the load multiple or store multiple instruction operation temporarily

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

After servicing the interrupt, the processor:

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

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

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

- **IT: If-Then Instruction**

Indicates the execution state bits of the IT instruction.

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

- **T: Thumb State**

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

– Instructions BLX, BX and POP{PC}

– Restoration from the stacked xPSR value on an exception return

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

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

10.4.1.12 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.

10.4.1.13 Priority Mask Register

Name: PRIMASK

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–							

7	6	5	4	3	2	1	0
–							PRIMASK

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

- **PRIMASK**

0: No effect

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

10.4.1.14 Fault Mask Register

Name: FAULTMASK

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–							

7	6	5	4	3	2	1	0
–							FAULTMASK

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

- **FAULTMASK**

0: No effect.

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

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

10.4.1.15 Base Priority Mask Register

Name: BASEPRI

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–							

7	6	5	4	3	2	1	0
BASEPRI							

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

- **BASEPRI**

Priority mask bits:

0x0000: No effect

Nonzero: Defines the base priority for exception processing

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

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

10.4.1.16 Control Register

Name: CONTROL

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–							

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

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

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

Indicates whether the floating-point context is currently active:

0: No floating-point context active.

1: Floating-point context active.

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

- **SPSEL: Active Stack Pointer**

Defines the current stack:

0: MSP is the current stack pointer.

1: PSP is the current stack pointer.

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

- **nPRIV: Thread Mode Privilege Level**

Defines the Thread mode privilege level:

0: Privileged.

1: Unprivileged.

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

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

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

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

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" .

10.4.1.17 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.

10.4.1.18 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.

10.4.1.19 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.

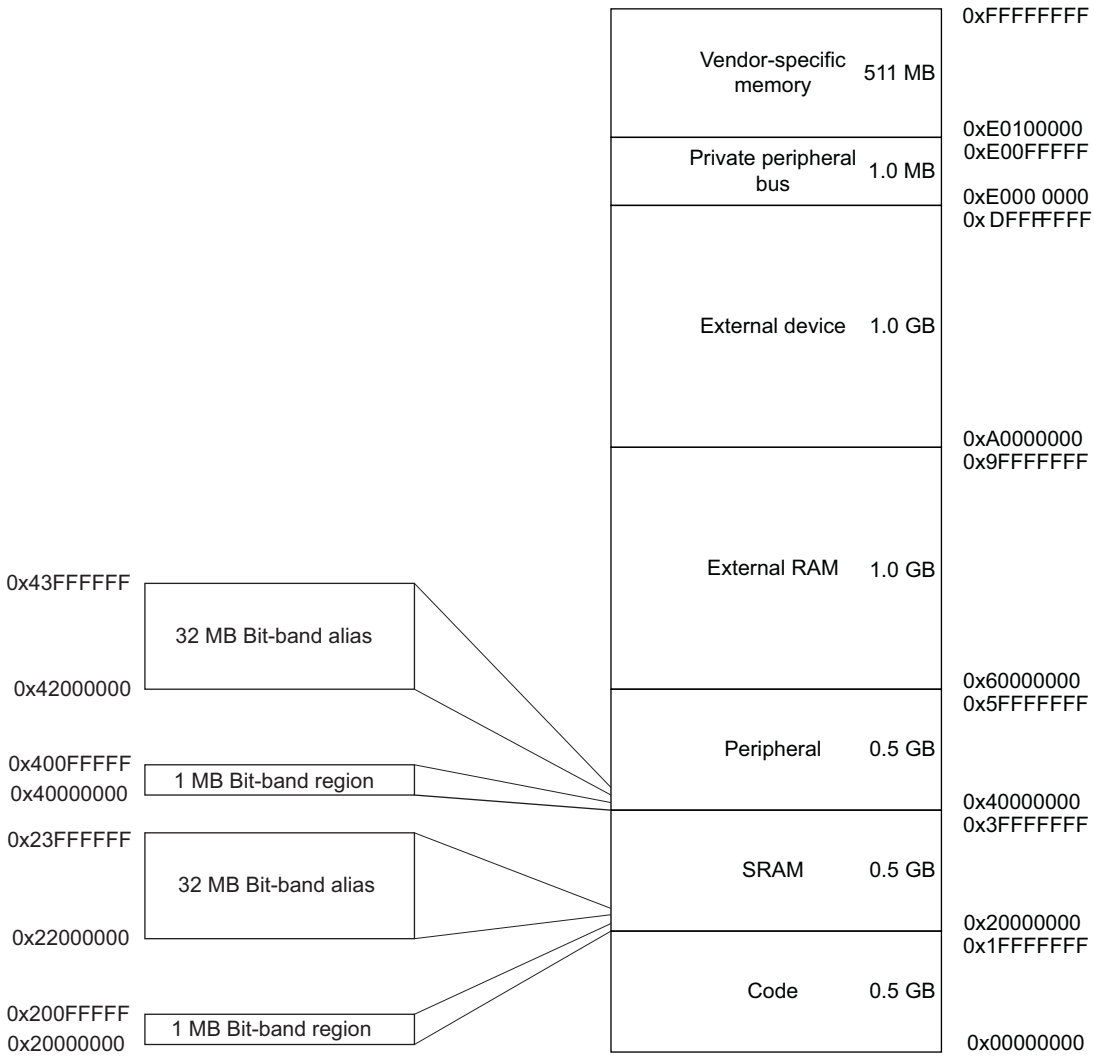
The following sections give more information about the CMSIS:

- Section 10.5.3 "Power Management Programming Hints"
- Section 10.6.2 "CMSIS Functions"
- Section 10.8.2.1 "NVIC Programming Hints" .

10.4.2 Memory Model

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

Figure 10-3. Memory Map



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

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

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

10.4.2.1 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.

10.4.2.2 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 10-3. 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.

10.4.2.3 Behavior of Memory Accesses

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

Table 10-4. 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 10-6.
0x40000000–0x5FFFFFFF	Peripheral	Device[1]	XN	This region includes bit band and bit band alias areas, see Table 10-6.
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 10-5 shows.

Table 10-5. 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.

10.4.2.4 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.

10.4.2.5 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 10-6.
- Accesses to the 32 MB peripheral alias region map to the 1 MB peripheral bit-band region, as shown in Table 10-7.

Table 10-6. SRAM Memory Bit-banding Regions

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

Table 10-7. Peripheral Memory Bit-banding Regions

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

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
```

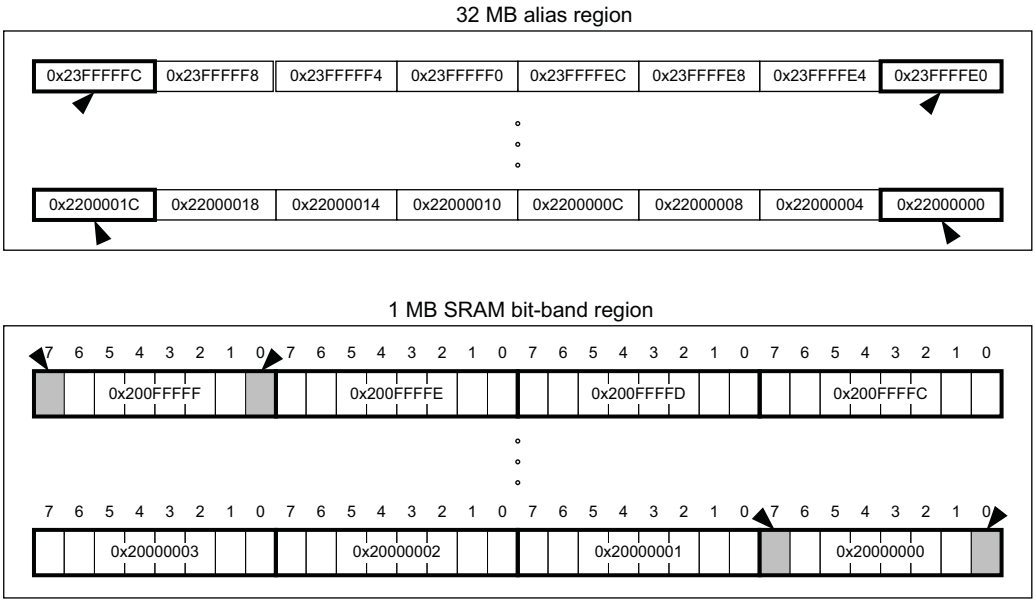
where:

- `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 10-4 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 10-4. 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.

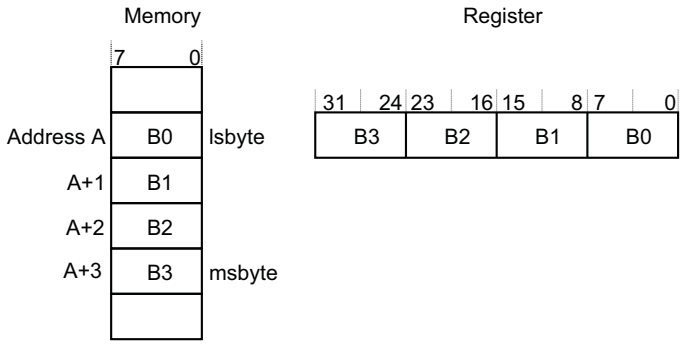
10.4.2.6 Memory Endianness

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 10-5. ***Little-endian Format***



10.4.2.7 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” .

10.4.2.8 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 10-8. 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);
```

10.4.3 Exception Model

This section describes the exception model.

10.4.3.1 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.

10.4.3.2 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 10-9. Properties of the Different Exception Types

Exception Number[1]	Irq Number[1]	Exception Type	Priority	Vector Address or Offset[2]	Activation
1	–	Reset	-3, the highest	0x00000004	Asynchronous
2	-14	NMI	-2	0x00000008	Asynchronous
3	-13	Hard fault	-1	0x0000000C	–
4	-12	Memory management fault	Configurable[3]	0x00000010	Synchronous
5	-11	Bus fault	Configurable[3]	0x00000014	Synchronous when precise, asynchronous when imprecise
6	-10	Usage fault	Configurable[3]	0x00000018	Synchronous
7–10	–	–	–	Reserved	–
11	-5	SVCall	Configurable[3]	0x0000002C	Synchronous
12–13	–	–	–	Reserved	–
14	-2	PendSV	Configurable[3]	0x00000038	Asynchronous
15	-1	SysTick	Configurable[3]	0x0000003C	Asynchronous
16 and above	0 and above	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 10-9 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" .

10.4.3.3 Exception Handlers

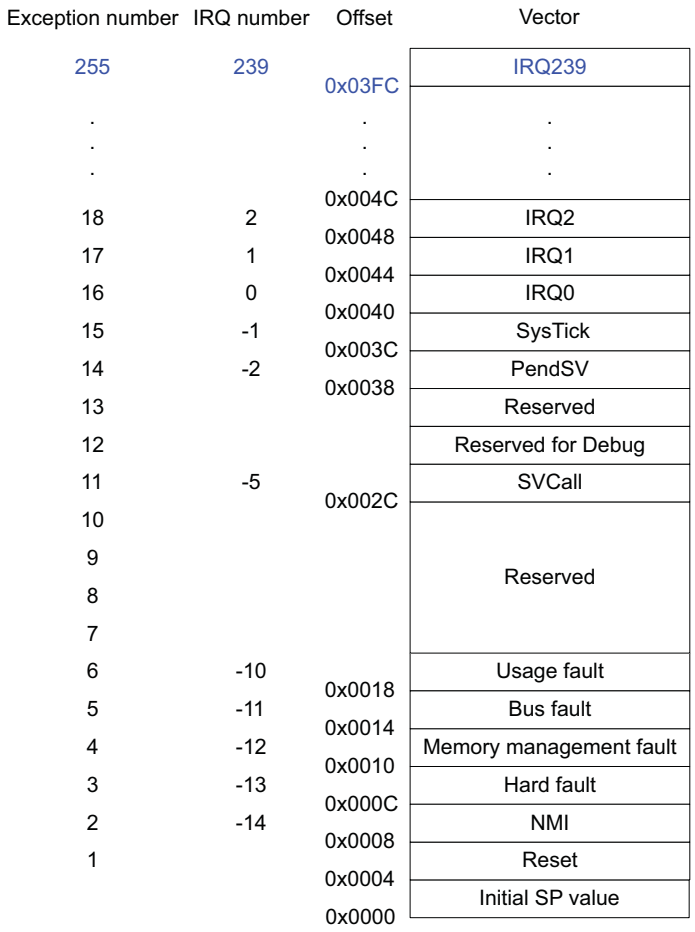
The processor handles exceptions using:

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

10.4.3.4 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 10-6 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 10-6. 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" .

10.4.3.5 Exception Priorities

As Table 10-9 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.

10.4.3.6 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” .

10.4.3.7 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.

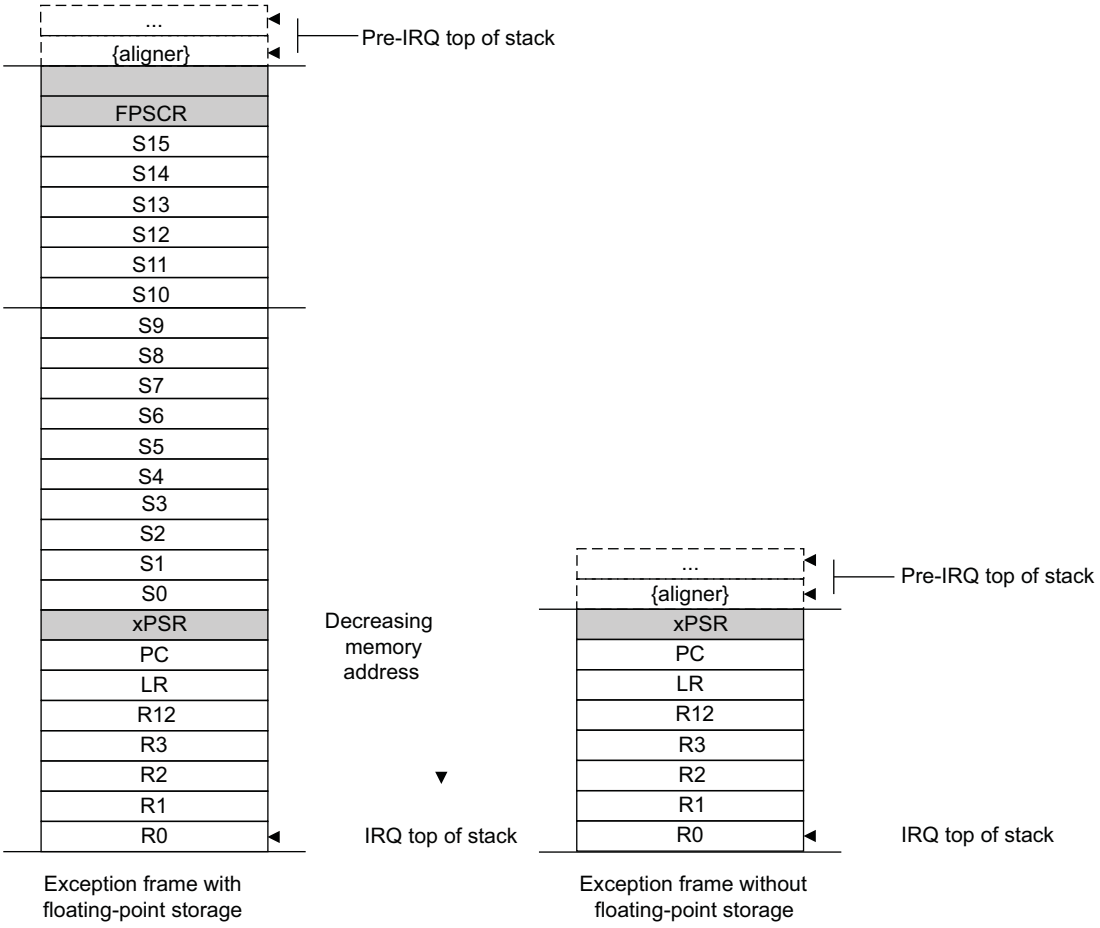
Sufficient priority means that the exception has more priority than any limits set by the mask registers, see “Exception Mask Registers” . An exception with less priority than this is pending but is not handled by the processor.

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 10-7 on page 56 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 10-7 on page 56 shows this stack frame also.

Figure 10-7. 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 10-10 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 10-10. 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.

10.4.3.8 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 10-11 shows the types of fault, the handler used for the fault, the corresponding fault status register, and the register bit that indicates that the fault has occurred. See "Configurable Fault Status Register" for more information about the fault status registers.

Table 10-11. Faults

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

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

Fault Escalation and Hard Faults

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

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

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

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

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

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

Fault Status Registers and Fault Address Registers

The fault status registers indicate the cause of a fault. For bus faults and memory management faults, the fault address register indicates the address accessed by the operation that caused the fault, as shown in Table 10-12.

Table 10-12. 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.

10.5 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.

10.5.1 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.

10.5.1.1 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.

10.5.1.2 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.

10.5.1.3 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.

10.5.2 Wakeup from Sleep Mode

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

10.5.2.1 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" .

10.5.2.2 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" .

10.5.2.3 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.

10.5.3 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
```

10.6 Cortex-M4 Instruction Set

10.6.1 Instruction Set Summary

The processor implements a version of the Thumb instruction set. Table 10-13 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 10-13. 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 10-13. 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 10-13. Cortex-M4 Instructions (Continued)

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

Table 10-13. Cortex-M4 Instructions (Continued)

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

Table 10-13. Cortex-M4 Instructions (Continued)

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

Table 10-13. 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	–

10.6.2 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 10-14. 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 10-15. CMSIS Intrinsic Functions to Access the Special Registers

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

10.6.3 Instruction Descriptions

10.6.3.1 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” .

10.6.3.2 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.

10.6.3.3 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 ≤ *n* ≤ 31.
LSR #*n*	logical shift right *n* bits, 1 ≤ *n* ≤ 32.
ROR #*n*	rotate right *n* bits, 1 ≤ *n* ≤ 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"

10.6.3.4 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

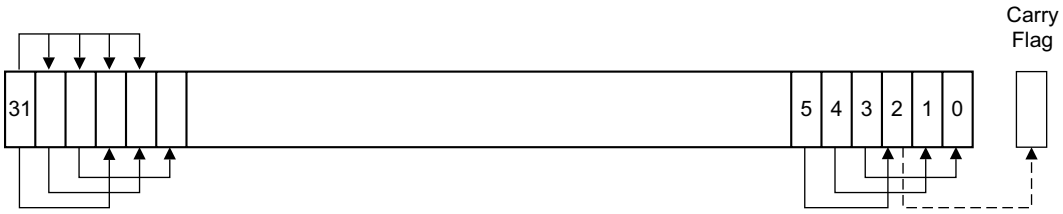
Arithmetic shift right by *n* bits moves the left-hand 32-n bits of the register, *Rm*, to the right by *n* places, into the right-hand 32-n bits of the result. And it copies the original bit[31] of the register into the left-hand *n* bits of the result. See Figure 10-8.

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 10-8. ASR #3



LSR

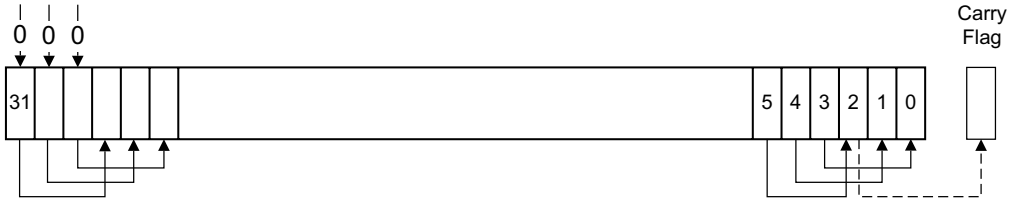
Logical shift right by n bits moves the left-hand 32-n bits of the register Rm, to the right by *n* places, into the right-hand 32-n bits of the result. And it sets the left-hand n bits of the result to 0. See Figure 10-9.

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 10-9. LSR #3



LSL

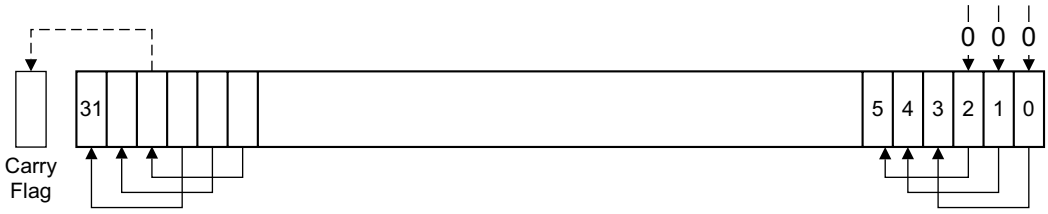
Logical shift left by *n* bits moves the right-hand 32-n bits of the register Rm, to the left by *n* places, into the left-hand 32-n bits of the result; and it sets the right-hand *n* bits of the result to 0. See Figure 10-10.

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 10-10. LSL #3



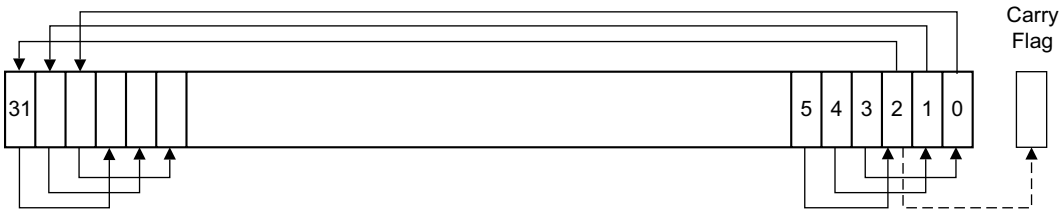
ROR

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 10-11.

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 10-11. ROR #3

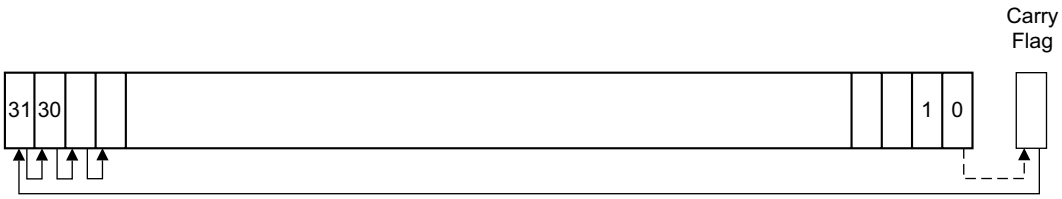


RRX

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 10-12.

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 10-12. RRX



10.6.3.5 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” .

10.6.3.6 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].

10.6.3.7 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 10-16 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 10-16 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 10-16 also shows the relationship between condition code suffixes and the N, Z, C, and V flags.

Table 10-16. Condition Code Suffixes

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

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
```

10.6.3.8 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
```

10.6.4 Memory Access Instructions

The table below shows the memory access instructions.

Table 10-17. 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

10.6.4.1 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
```

10.6.4.2 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 10-18. Offset Ranges

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

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.
```

10.6.4.3 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
```

10.6.4.4 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
```

10.6.4.5 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 10-19. 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
```

10.6.4.6 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
```

10.6.4.7 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}
```

10.6.4.8 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
```

10.6.4.9 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
```

10.6.5 General Data Processing Instructions

The table below shows the data processing instructions.

Table 10-20. Data Processing Instructions

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

Table 10-20. Data Processing Instructions (Continued)

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

10.6.5.1 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
```

10.6.5.2 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
```

10.6.5.3 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.
```

10.6.5.4 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
```

10.6.5.5 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
```

10.6.5.6 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.
```

10.6.5.7 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.
```

10.6.5.8 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.
```

10.6.5.9 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.
```

10.6.5.10 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.
```

10.6.5.11 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.
```

10.6.5.12 SHSUB16 and SHSUB8

Signed Halving Subtract 16 and Signed Halving Subtract 8

Syntax

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

where:

op is any of:

SHSUB16 Signed Halving Subtract 16.

SHSUB8 Signed Halving Subtract 8.

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

Rd is the destination register.

Rn is the first operand register.

Rm is the second operand register.

Operation

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

The SHSUB16 instruction:

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

The SHSUBB8 instruction:

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

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

10.6.5.13 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.
```

10.6.5.14 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.
```

10.6.5.15 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.
```

10.6.5.16 UADD16 and UADD8

Unsigned Add 16 and Unsigned Add 8

Syntax

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

where:

op is any of:

UADD16 Performs two 16-bit unsigned integer additions.

UADD8 Performs four 8-bit unsigned integer additions.

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

Rd is the destination register.

Rn is the first register holding the operand.

Rm is the second register holding the operand.

Operation

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

The UADD16 instruction:

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

The UADD16 instruction:

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not change the flags.

Examples

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

10.6.5.17 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.
```

10.6.5.18 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.
```

10.6.5.19 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.
```

10.6.5.20 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.
```

10.6.5.21 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.
```

10.6.5.22 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.
```

10.6.5.23 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.
```

10.6.5.24 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.
```

10.6.6 Multiply and Divide Instructions

The table below shows the multiply and divide instructions.

Table 10-21. Multiply and Divide Instructions

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

10.6.6.1 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)
```

10.6.6.2 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.
```

10.6.6.3 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.
```

10.6.6.4 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.
```

10.6.6.5 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.
```

10.6.6.6 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.
```

10.6.6.7 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.
```

10.6.6.8 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.
```

10.6.6.9 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.
```

10.6.6.10 SMUL and SMULW

Signed Multiply (halfwords) and Signed Multiply (word by halfword)

Syntax

```
op{XY}{cond} Rd,Rn, Rm
op{Y}{cond} Rd. Rn, Rm
```

For *SMULXY* only:

op is one of:

SMUL{*XY*} Signed Multiply (halfwords).

X and *Y* specify which halfword of the source registers *Rn* and *Rm* is used as the first and second multiply operand.
If *X* is B, then the bottom halfword, bits [15:0] of *Rn* is used.
If *X* is T, then the top halfword, bits [31:16] of *Rn* is used.If *Y* is B, then the bot tom halfword, bits [15:0], of *Rm* is used.
If *Y* is T, then the top halfword, bits [31:16], of *Rm* is used.

SMULW{Y} Signed Multiply (word by halfword).

Y specifies which halfword of the source register *Rm* is used as the second mul tiply operand.
If *Y* is B, then the bottom halfword (bits [15:0]) of *Rm* is used.
If *Y* is T, then the top halfword (bits [31:16]) of *Rm* is used.

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

Rd is the destination register.

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

Operation

The SMULBB, SMULTB, SMULBT and SMULTT instructions interprets the values from *Rn* and *Rm* as four signed 16-bit integers. These instructions:

- Multiplies the specified signed halfword, Top or Bottom, values from *Rn* and *Rm*.
- Writes the 32-bit result of the multiplication in *Rd.*

The SMULWT and SMULWB instructions interprets the values from *Rn* as a 32-bit signed integer and *Rm* as two halfword 16-bit signed integers. These instructions:

- Multiplies the first operand and the top, T suffix, or the bottom, B suffix, halfword of the second operand.
- Writes the signed most significant 32 bits of the 48-bit result in the destination register.

Restrictions

In these instructions:

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

Examples

```
SMULBT       R0, R4, R5  ; Multiplies the bottom halfword of R4 with the
                         ; top halfword of R5, multiplies results and
                         ; writes to R0
SMULBB       R0, R4, R5  ; Multiplies the bottom halfword of R4 with the
                         ; bottom halfword of R5, multiplies results and
                         ; writes to R0
SMULTT       R0, R4, R5  ; Multiplies the top halfword of R4 with the top
                         ; halfword of R5, multiplies results and writes
                         ; to R0
SMULTB       R0, R4, R5  ; Multiplies the top halfword of R4 with the
                         ; bottom halfword of R5, multiplies results and
```

```
                        ; and writes to R0
SMULWT      R4, R5, R3  ; Multiplies R5 with the top halfword of R3,
                        ; extracts top 32 bits and writes to R4
SMULWB      R4, R5, R3  ; Multiplies R5 with the bottom halfword of R3,
                        ; extracts top 32 bits and writes to R4.
```

10.6.6.11 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
```

10.6.6.12 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
```

10.6.7 Saturating Instructions

The table below shows the saturating instructions.

Table 10-22. 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” .

10.6.7.1 SSAT and USAT

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

Syntax

```
op{cond} Rd, #n, Rm {, shift #s}
```

where:

op	is one of:
	SSAT Saturates a signed value to a signed range.
	USAT Saturates a signed value to an unsigned range.
cond	is an optional condition code, see “Conditional Execution” .
Rd	is the destination register.
n	specifies the bit position to saturate to:
n ranges from 1	n ranges from 0 to 31 for USAT.
to 32 for SSAT	
Rm	is the register containing the value to saturate.
shift #s	is an optional shift applied to Rm before saturating. It must be one of the following:
ASR #s	where s is in the range 1 to 31.
LSL #s	where *s* is in the range 0 to 31.

Operation

These instructions saturate to a signed or unsigned *n*-bit value.

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

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

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

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

Examples

```
        SSAT    R7, #16, R7, LSL #4  ; Logical shift left value in R7 by 4, then
                                     ; saturate it as a signed 16-bit value and
                                     ; write it back to R7
        USATNE  R0, #7, R5           ; Conditionally saturate value in R5 as an
                                     ; unsigned 7 bit value and write it to R0.
```

10.6.7.2 SSAT16 and USAT16

Signed Saturate and Unsigned Saturate to any bit position for two halfwords.

Syntax

```
op{cond} Rd, #n, Rm
```

where:

op	is one of:
	SSAT16 Saturates a signed halfword value to a signed range.
	USAT16 Saturates a signed halfword value to an unsigned range.
cond	is an optional condition code, see “Conditional Execution” .
Rd	is the destination register.
n	specifies the bit position to saturate to:
n ranges from 1	n ranges from 0 to 15 for USAT.
to 16 for SSAT	
Rm	is the register containing the value to saturate.

Operation

The SSAT16 instruction:

Saturates two signed 16-bit halfword values of the register with the value to saturate from selected by the bit position in *n*.

Writes the results as two signed 16-bit halfwords to the destination register.

The USAT16 instruction:

Saturates two unsigned 16-bit halfword values of the register with the value to saturate from selected by the bit position in *n*.

Writes the results as two unsigned halfwords in the destination register.

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

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

Examples

```
SSAT16    R7, #9, R2   ; Saturates the top and bottom highwords of R2
                       ; as 9-bit values, writes to corresponding halfword
                       ; of R7
USAT16NE  R0, #13, R5  ; Conditionally saturates the top and bottom
                       ; halfwords of R5 as 13-bit values, writes to
                       ; corresponding halfword of R0.
```

10.6.7.3 QADD and QSUB

Saturating Add and Saturating Subtract, signed.

Syntax

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

where:

op is one of:

QADD Saturating 32-bit add.

QADD8 Saturating four 8-bit integer additions.

QADD16 Saturating two 16-bit integer additions.

QSUB Saturating 32-bit subtraction.

QSUB8 Saturating four 8-bit integer subtraction.

QSUB16 Saturating two 16-bit integer subtraction.

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

Rd is the destination register.

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

Operation

These instructions add or subtract two, four or eight values from the first and second operands and then writes a signed saturated value in the destination register.

The QADD and QSUB instructions apply the specified add or subtract, and then saturate the result to the signed range $-2^{n-1} \le x \le 2^{n-1}-1$, where x is given by the number of bits applied in the instruction, 32, 16 or 8.

If the returned result is different from the value to be saturated, it is called *saturation*. If saturation occurs, the QADD and QSUB instructions set the Q flag to 1 in the APSR. Otherwise, it leaves the Q flag unchanged. The 8-bit and 16-bit QADD and QSUB instructions always leave the Q flag unchanged.

To clear the Q flag to 0, the MSR instruction must be used; see "MSR" .

To read the state of the Q flag, the MRS instruction must be used; see "MRS" .

Restrictions

Do not use SP and do not use PC.

Condition Flags

These instructions do not affect the condition code flags.

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

Examples

```
QADD16   R7, R4, R2  ; Adds halfwords of R4 with corresponding halfword of
                     ; R2, saturates to 16 bits and writes to
                     ; corresponding halfword of R7
QADD8    R3, R1, R6  ; Adds bytes of R1 to the corresponding bytes of R6,
                     ; saturates to 8 bits and writes to corresponding
                     ; byte of R3
QSUB16   R4, R2, R3  ; Subtracts halfwords of R3 from corresponding
                     ; halfword of R2, saturates to 16 bits, writes to
                     ; corresponding halfword of R4
QSUB8    R4, R2, R5  ; Subtracts bytes of R5 from the corresponding byte
                     ; in R2, saturates to 8 bits, writes to corresponding
                     ; byte of R4.
```

10.6.7.4 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.
```

10.6.7.5 QDADD and QDSUB

Saturating Double and Add and Saturating Double and Subtract, signed.

Syntax

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

where:

op is one of:

QDADD Saturating Double and Add.

QDSUB Saturating Double and Subtract.

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

Rd is the destination register.

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

Operation

The QDADD instruction:

- Doubles the second operand value.
- Adds the result of the doubling to the signed saturated value in the first operand.
- Writes the result to the destination register.

The QDSUB instruction:

- Doubles the second operand value.
- Subtracts the doubled value from the signed saturated value in the first operand.
- Writes the result to the destination register.

Both the doubling and the addition or subtraction have their results saturated to the 32-bit signed integer range $-2^{31} \leq x \leq 2^{31} - 1$. If saturation occurs in either operation, it sets the Q flag in the APSR.

Restrictions

Do not use SP and do not use PC.

Condition Flags

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

Examples

```
QDADD     R7, R4, R2   ; Doubles and saturates R4 to 32 bits, adds R2,
                       ; saturates to 32 bits, writes to R7
QDSUB     R0, R3, R5   ; Subtracts R3 doubled and saturated to 32 bits
                       ; from R5, saturates to 32 bits, writes to R0.
```

10.6.7.6 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.
```

10.6.7.7 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.
```

10.6.8 Packing and Unpacking Instructions

The table below shows the instructions that operate on packing and unpacking data.

Table 10-23. 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

10.6.8.1 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.
```

10.6.8.2 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.
```

10.6.8.3 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.
```

10.6.9 Bitfield Instructions

The table below shows the instructions that operate on adjacent sets of bits in registers or bitfields.

Table 10-24. 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

10.6.9.1 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.
```

10.6.9.2 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.
```

10.6.9.3 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.
```

10.6.10 Branch and Control Instructions

The table below shows the branch and control instructions.

Table 10-25. 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

10.6.10.1 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 10-26. Branch Ranges

Instruction	Branch Range
B label	–16 MB to +16 MB
B*cond* label (outside IT block)	–1 MB to +1 MB
B*cond* label (inside IT block)	–16 MB to +16 MB
BL{*cond*} label	–16 MB to +16 MB
BX{*cond*} Rm	Any value in register
BLX{*cond*} Rm	Any value in register

The .W suffix might be used to get the maximum branch range. See "Instruction Width Selection" .

Restrictions

The restrictions are:

- Do not use PC in the BLX instruction
- For BX and BLX, bit[0] of *Rm* must be 1 for correct execution but a branch occurs to the target address created by changing bit[0] to 0
- When any of these instructions is inside an IT block, it must be the last instruction of the IT block.

B*cond* is the only conditional instruction that is not required to be inside an IT block. However, it has a longer branch range when it is inside an IT block.

Condition Flags

These instructions do not change the flags.

Examples

```
B       loopA    ; Branch to loopA
BLE     ng       ; Conditionally branch to label ng
B.W     target   ; Branch to target within 16MB range
BEQ     target   ; Conditionally branch to target
BEQ.W   target   ; Conditionally branch to target within 1MB
BL      funC     ; Branch with link (Call) to function funC, return address
                 ; stored in LR
BX      LR       ; Return from function call
BXNE    R0       ; Conditionally branch to address stored in R0
BLX     R0       ; Branch with link and exchange (Call) to a address stored in R0.
```

10.6.10.2 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
```

10.6.10.3 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
```

10.6.10.4 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
```

10.6.11 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 10-27. Floating-point Instructions

Mnemonic	Description
VABS	Floating-point Absolute
VADD	Floating-point Add
VCMP	Compare two floating-point registers, or one floating-point register and zero
VCMPE	Compare two floating-point registers, or one floating-point register and zero with Invalid Operation check
VCVT	Convert between floating-point and integer
VCVT	Convert between floating-point and fixed point
VCVTR	Convert between floating-point and integer with rounding
VCVTB	Converts half-precision value to single-precision
VCVTT	Converts single-precision register to half-precision
VDIV	Floating-point Divide
VFMA	Floating-point Fused Multiply Accumulate
VFNMA	Floating-point Fused Negate Multiply Accumulate
VFMS	Floating-point Fused Multiply Subtract
VFNMS	Floating-point Fused Negate Multiply Subtract
VLDM	Load Multiple extension registers
VLDR	Loads an extension register from memory
VLMA	Floating-point Multiply Accumulate
VLMS	Floating-point Multiply Subtract
VMOV	Floating-point Move Immediate
VMOV	Floating-point Move Register
VMOV	Copy ARM core register to single precision
VMOV	Copy 2 ARM core registers to 2 single precision
VMOV	Copies between ARM core register to scalar
VMOV	Copies between Scalar to ARM core register
VMRS	Move to ARM core register from floating-point System Register
VMSR	Move to floating-point System Register from ARM Core register
VMUL	Multiply floating-point
VNEG	Floating-point negate
VNMLA	Floating-point multiply and add
VNMLS	Floating-point multiply and subtract
VNMUL	Floating-point multiply
VPOP	Pop extension registers

Table 10-27. Floating-point Instructions (Continued)

Mnemonic	Description
VPUSH	Push extension registers
VSQRT	Floating-point square root
VSTM	Store Multiple extension registers
VSTR	Stores an extension register to memory
VSUB	Floating-point Subtract

10.6.11.1 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
```

10.6.11.2 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
```

10.6.11.3 VCMP, VCMPE

Compares two floating-point registers, or one floating-point register and zero.

Syntax

```
VCMP{E}{cond}.F32 Sd, Sm
VCMP{E}{cond}.F32 Sd, #0.0
```

where:

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

E — If present, any NaN operand causes an Invalid Operation exception. Otherwise, only a signaling NaN causes the exception.

Sd — is the floating-point operand to compare.

Sm — is the floating-point operand that is compared with.

Operation

This instruction:

1. Compares:
   - Two floating-point registers.
   - One floating-point register and zero.
2. Writes the result to the FPSCR flags.

Restrictions

This instruction can optionally raise an Invalid Operation exception if either operand is any type of NaN. It always raises an Invalid Operation exception if either operand is a signaling NaN.

Condition Flags

When this instruction writes the result to the FPSCR flags, the values are normally transferred to the ARM flags by a subsequent VMRS instruction, see "" .

Examples

```
VCMP.F32    S4, #0.0
VCMP.F32    S4, S2
```

10.6.11.4 VCVT, VCVTR between Floating-point and Integer

Converts a value in a register from floating-point to a 32-bit integer.

Syntax

```
VCVT{R}{cond}.Tm.F32 Sd, Sm
VCVT{cond}.F32.Tm Sd, Sm
```

where:

R — If *R* is specified, the operation uses the rounding mode specified by the FPSCR. If *R* is omitted. the operation uses the Round towards Zero rounding mode.

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

Tm — is the data type for the operand. It must be one of:

S32 signed 32-bit value. — U32 unsigned 32-bit value.

Sd, Sm — are the destination register and the operand register.

Operation

These instructions:

1. Either
   - Converts a value in a register from floating-point value to a 32-bit integer.
   - Converts from a 32-bit integer to floating-point value.
2. Places the result in a second register.

The floating-point to integer operation normally uses the *Round towards Zero* rounding mode, but can optionally use the rounding mode specified by the FPSCR.

The integer to floating-point operation uses the rounding mode specified by the FPSCR.

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

10.6.11.5 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.

10.6.11.6 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.

10.6.11.7 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.

10.6.11.8 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.

10.6.11.9 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.

10.6.11.10 VLDM

Floating-point Load Multiple

Syntax

```
VLDM{mode}{cond}{.size} Rn{!}, list
```

where:

mode — is the addressing mode:
- *IA* Increment After. The consecutive addresses start at the address specified in *Rn*.
- *DB* Decrement Before. The consecutive addresses end just before the address specified in *Rn*.

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

size — is an optional data size specifier.

Rn — is the base register. The SP can be used

! — is the command to the instruction to write a modified value back to *Rn*. This is required if mode == DB, and is optional if mode == IA.

list — is the list of extension registers to be loaded, as a list of consecutively numbered doubleword or singleword registers, separated by commas and surrounded by brackets.

Operation

This instruction loads:

- Multiple extension registers from consecutive memory locations using an address from an ARM core register as the base address.

Restrictions

The restrictions are:

- If *size* is present, it must be equal to the size in bits, 32 or 64, of the registers in *list*.
- For the base address, the SP can be used.
  In the ARM instruction set, if *!* is not specified the PC can be used.
- list must contain at least one register. If it contains doubleword registers, it must not contain more than 16 registers.
- If using the *Decrement Before addressing* mode, the write back flag, *!*, must be appended to the base register specification.

Condition Flags

These instructions do not change the flags.

10.6.11.11 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.

10.6.11.12 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.

10.6.11.13 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.

10.6.11.14 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.

10.6.11.15 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.

10.6.11.16 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.

10.6.11.17 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.

10.6.11.18 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.

10.6.11.19 VMRS

Move to ARM Core register from floating-point System Register.

Syntax

```
VMRS{cond} Rt, FPSCR
VMRS{cond} APSR_nzcv, FPSCR
```

where:

cond	is an optional condition code, see “Conditional Execution” .
Rt	is the destination ARM core register. This register can be R0–R14.
APSR_nzcv	Transfer floating-point flags to the APSR flags.

Operation

This instruction performs one of the following actions:

- Copies the value of the FPSCR to a general-purpose register.
- Copies the value of the FPSCR flag bits to the APSR N, Z, C, and V flags.

Restrictions

Rt cannot be PC or SP.

Condition Flags

These instructions optionally change the flags: N, Z, C, V

10.6.11.20 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.

10.6.11.21 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.

10.6.11.22 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.

10.6.11.23 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.

10.6.11.24 VPOP

Floating-point extension register Pop.

Syntax

```
VPOP{cond}{.size} list
```

where:

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

size — is an optional data size specifier.
If present, it must be equal to the size in bits, 32 or 64, of the registers in *list*.

list — is the list of extension registers to be loaded, as a list of consecutively numbered doubleword or singleword registers, separated by commas and surrounded by brackets.

Operation

This instruction loads multiple consecutive extension registers from the stack.

Restrictions

The list must contain at least one register, and not more than sixteen registers.

Condition Flags

These instructions do not change the flags.

10.6.11.25 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.

10.6.11.26 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.

10.6.11.27 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.

10.6.11.28 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.

10.6.11.29 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.

10.6.12 Miscellaneous Instructions

The table below shows the remaining Cortex-M4 instructions.

Table 10-28. 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

10.6.12.1 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.

10.6.12.2 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)
```

10.6.12.3 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
```

10.6.12.4 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
```

10.6.12.5 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
```

10.6.12.6 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
```

10.6.12.7 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
```

10.6.12.8 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
```

10.6.12.9 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
```

10.6.12.10SVC

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)
```

10.6.12.11 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
```

10.6.12.12WFI

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
```

10.7 Cortex-M4 Core Peripherals

10.7.1 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 10.8 ”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 10.9 ”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 10.10 ”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 10.11 ”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 10.12 ”Floating Point Unit (FPU)”

10.7.2 Address Map

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

Table 10-29. 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.

10.8 Nested Vectored Interrupt Controller (NVIC)

This section describes the NVIC and the registers it uses. The NVIC supports:

- Up to 47 interrupts.
- A programmable priority level of 0–15 for each interrupt. A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority.
- Level detection of interrupt signals.
- Dynamic reprioritization of interrupts.
- Grouping of priority values into group priority and subpriority fields.
- Interrupt tail-chaining.
- An external *Non-maskable interrupt* (NMI)

The processor automatically stacks its state on exception entry and unstacks this state on exception exit, with no instruction overhead. This provides low latency exception handling.

10.8.1 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.

10.8.1.1 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.

10.8.2 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” .

10.8.2.1 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 10-30. 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–IPR11) 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 10-31 shows how the interrupts, or IRQ numbers, map onto the interrupt registers and corresponding CMSIS variables that have one bit per interrupt.

Table 10-31. Mapping of Interrupts to the Interrupt Variables

	CMSIS Array Elements[1]				
Interrupts	**Set-enable**	**Clear-enable**	**Set-pending**	**Clear-pending**	**Active Bit**
0–31	ISER[0]	ICER[0]	ISPR[0]	ICPR[0]	IABR[0]
32–47	ISER[1]	ICER[1]	ISPR[1]	ICPR[1]	IABR[1]

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

10.8.3 Nested Vectored Interrupt Controller (NVIC) User Interface

Table 10-32. Nested Vectored Interrupt Controller (NVIC) Register Mapping

Offset	Register	Name	Access	Reset
0xE000E100	Interrupt Set-enable Register 0	NVIC_ISER0	Read/Write	0x00000000
...	...	...	...	...
0xE000E11C	Interrupt Set-enable Register 7	NVIC_ISER7	Read/Write	0x00000000
0XE000E180	Interrupt Clear-enable Register 0	NVIC_ICER0	Read/Write	0x00000000
...	...	...	...	...
0xE000E19C	Interrupt Clear-enable Register 7	NVIC_ICER7	Read/Write	0x00000000
0XE000E200	Interrupt Set-pending Register 0	NVIC_ISPR0	Read/Write	0x00000000
...	...	...	...	...
0xE000E21C	Interrupt Set-pending Register 7	NVIC_ISPR7	Read/Write	0x00000000
0XE000E280	Interrupt Clear-pending Register 0	NVIC_ICPR0	Read/Write	0x00000000
...	...	...	...	...
0xE000E29C	Interrupt Clear-pending Register 7	NVIC_ICPR7	Read/Write	0x00000000
0xE000E300	Interrupt Active Bit Register 0	NVIC_IABR0	Read/Write	0x00000000
...	...	...	...	...
0xE000E31C	Interrupt Active Bit Register 7	NVIC_IABR7	Read/Write	0x00000000
0xE000E400	Interrupt Priority Register 0	NVIC_IPR0	Read/Write	0x00000000
...	...	...	...	...
0xE000E42C	Interrupt Priority Register 11	NVIC_IPR11	Read/Write	0x00000000
0xE000EF00	Software Trigger Interrupt Register	NVIC_STIR	Write-only	0x00000000

10.8.3.1 Interrupt Set-enable Registers

Name: NVIC_ISERx [x=0..7]

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
SETENA							

23	22	21	20	19	18	17	16
SETENA							

15	14	13	12	11	10	9	8
SETENA							

7	6	5	4	3	2	1	0
SETENA							

These registers enable interrupts and show which interrupts are enabled.

- **SETENA: Interrupt Set-enable**

Write:

0: No effect.

1: Enables the interrupt.

Read:

0: Interrupt disabled.

1: Interrupt enabled.

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

10.8.3.2 Interrupt Clear-enable Registers

Name: NVIC_ICERx [x=0..7]

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
CLRENA							

23	22	21	20	19	18	17	16
CLRENA							

15	14	13	12	11	10	9	8
CLRENA							

7	6	5	4	3	2	1	0
CLRENA							

These registers disable interrupts, and show which interrupts are enabled.

- **CLRENA: Interrupt Clear-enable**

Write:

0: No effect.

1: Disables the interrupt.

Read:

0: Interrupt disabled.

1: Interrupt enabled.

10.8.3.3 Interrupt Set-pending Registers

Name: NVIC_ISPRx [x=0..7]

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
SETPEND							

23	22	21	20	19	18	17	16
SETPEND							

15	14	13	12	11	10	9	8
SETPEND							

7	6	5	4	3	2	1	0
SETPEND							

These registers force interrupts into the pending state, and show which interrupts are pending.

- **SETPEND: Interrupt Set-pending**

Write:

0: No effect.

1: Changes the interrupt state to pending.

Read:

0: Interrupt is not pending.

1: Interrupt is pending.

Notes:
1. Writing a 1 to an ISPR bit corresponding to an interrupt that is pending has no effect.
2. Writing a 1 to an ISPR bit corresponding to a disabled interrupt sets the state of that interrupt to pending.

10.8.3.4 Interrupt Clear-pending Registers

Name: NVIC_ICPRx [x=0..7]

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
CLRPEND							

23	22	21	20	19	18	17	16
CLRPEND							

15	14	13	12	11	10	9	8
CLRPEND							

7	6	5	4	3	2	1	0
CLRPEND							

These registers remove the pending state from interrupts, and show which interrupts are pending.

- **CLRPEND: Interrupt Clear-pending**

Write:

0: No effect.

1: Removes the pending state from an interrupt.

Read:

0: Interrupt is not pending.

1: Interrupt is pending.

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

10.8.3.5 Interrupt Active Bit Registers

Name: NVIC_IABRx [x=0..7]

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
ACTIVE							

23	22	21	20	19	18	17	16
ACTIVE							

15	14	13	12	11	10	9	8
ACTIVE							

7	6	5	4	3	2	1	0
ACTIVE							

These registers indicate which interrupts are active.

- **ACTIVE: Interrupt Active Flags**

0: Interrupt is not active.

1: Interrupt is active.

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

10.8.3.6 Interrupt Priority Registers

Name: NVIC_IPRx [x=0..11]

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
PRI3							

23	22	21	20	19	18	17	16
PRI2							

15	14	13	12	11	10	9	8
PRI1							

7	6	5	4	3	2	1	0
PRI0							

The NVIC_IPR0–NVIC_IPR11 registers provide a 8-bit priority field for each interrupt. These registers are byte-accessible. Each register holds four priority fields that map up to four elements in the CMSIS interrupt priority array IP[0] to IP[46].

- **PRI3: Priority (4m+3)**

Priority, Byte Offset 3, refers to register bits [31:24].

- **PRI2: Priority (4m+2)**

Priority, Byte Offset 2, refers to register bits [23:16].

- **PRI1: Priority (4m+1)**

Priority, Byte Offset 1, refers to register bits [15:8].

- **PRI0: Priority (4m)**

Priority, Byte Offset 0, refers to register bits [7:0].

Notes:
1. Each priority field holds a priority value, 0–15. The lower the value, the greater the priority of the corresponding interrupt. The processor implements only bits[7:4] of each field; bits[3:0] read as zero and ignore writes.
2. For more information about the IP[0] to IP[46] interrupt priority array, that provides the software view of the interrupt priorities, see Table 10-30, “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.

10.8.3.7 Software Trigger Interrupt Register

Name: NVIC_STIR

Access: Write-only

Reset: 0x000000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	INTID

7	6	5	4	3	2	1	0
INTID							

Write to this register to generate an interrupt from the software.

- **INTID: Interrupt ID**

Interrupt ID of the interrupt to trigger, in the range 0–239. For example, a value of 0x03 specifies interrupt IRQ3.

10.9 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.

10.9.1 System Control Block (SCB) User Interface

Table 10-33. 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
0xE000ED3C	Auxiliary Fault Status Register	SCB_AFSR	Read/Write	0x00000000

Notes: 1. See the register description for more information.

2. This register contains the subregisters: "MMFSR: Memory Management Fault Status Subregister" (0xE000ED28 - 8 bits), "BFSR: Bus Fault Status Subregister" (0xE000ED29 - 8 bits), "UFSR: Usage Fault Status Subregister" (0xE000ED2A - 16 bits).

10.9.1.1 Auxiliary Control Register

Name: SCB_ACTLR

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–						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.

10.9.1.2 CPUID Base Register

Name: SCB_CPUID

Access: Read/Write

Reset: 0x000000000

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.

10.9.1.3 Interrupt Control and State Register

Name: SCB_ICSR

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
NMIPENDSET	–		PENDSVSET	PENDSVCLR	PENDSTSET	PENDSTCLR	–
23	22	21	20	19	18	17	16
–	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.

10.9.1.4 Vector Table Offset Register

Name: SCB_VTOR

Access: Read/Write

Reset: 0x000000000

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.

10.9.1.5 Application Interrupt and Reset Control Register

Name: SCB_AIRCR

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
VECTKEYSTAT/VECTKEY							
23	22	21	20	19	18	17	16
VECTKEYSTAT/VECTKEY							
15	14	13	12	11	10	9	8
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

PRIGROUP	Interrupt Priority Level Value, PRI_*N*[7:0]			Number of	
	Binary Point[1]	Group Priority Bits	Subpriority Bits	Group Priorities	Subpriorities
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.

10.9.1.6 System Control Register

Name: SCB_SCR

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–							

7	6	5	4	3	2	1	0
–			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.

10.9.1.7 Configuration and Control Register

Name: SCB_CCR

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–						STKALIGN	BFHFNMIGN

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

The SCB_CCR controls the entry to the Thread mode and enables the handlers for NMI, hard fault and faults escalated by FAULTMASK to ignore BusFaults. It also enables the division by zero and unaligned access trapping, and the access to the NVIC_STIR by unprivileged software (see “Software Trigger Interrupt Register”).

- **STKALIGN: Stack Alignment**

Indicates the stack alignment on exception entry:

0: 4-byte aligned.

1: 8-byte aligned.

On exception entry, the processor uses bit [9] of the stacked PSR to indicate the stack alignment. On return from the exception, it uses this stacked bit to restore the correct stack alignment.

- **BFHFNMIGN: Bus Faults Ignored**

Enables handlers with priority -1 or -2 to ignore data bus faults caused by load and store instructions. This applies to the hard fault and FAULTMASK escalated handlers:

0: Data bus faults caused by load and store instructions cause a lock-up.

1: Handlers running at priority -1 and -2 ignore data bus faults caused by load and store instructions.

Set this bit to 1 only when the handler and its data are in absolutely safe memory. The normal use of this bit is to probe system devices and bridges to detect control path problems and fix them.

- **DIV_0_TRP: Division by Zero Trap**

Enables faulting or halting when the processor executes an SDIV or UDIV instruction with a divisor of 0:

0: Do not trap divide by 0.

1: Trap divide by 0.

When this bit is set to 0, a divide by zero returns a quotient of 0.

- **UNALIGN_TRP: Unaligned Access Trap**

Enables unaligned access traps:

0: Do not trap unaligned halfword and word accesses.

1: Trap unaligned halfword and word accesses.

If this bit is set to 1, an unaligned access generates a usage fault.

Unaligned LDM, STM, LDRD, and STRD instructions always fault irrespective of whether UNALIGN_TRP is set to 1.

- **USERSETMPEND: Unprivileged Software Access**

Enables unprivileged software access to the NVIC_STIR, see “Software Trigger Interrupt Register” :

0: Disable.

1: Enable.

- **NONBASETHRDENA: Thread Mode Enable**

Indicates how the processor enters Thread mode:

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

1: The processor can enter the Thread mode from any level under the control of an EXC_RETURN value, see “Exception Return”
.

10.9.1.8 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 10-34. 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.

10.9.1.9 System Handler Priority Register 1

Name: SCB_SHPR1

Access: Read/Write

Reset: 0x000000000

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.

10.9.1.10 System Handler Priority Register 2

Name: SCB_SHPR2

Access: Read/Write

Reset: 0x000000000

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.

10.9.1.11 System Handler Priority Register 3

Name: SCB_SHPR3

Access: Read/Write

Reset: 0x000000000

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.

10.9.1.12 System Handler Control and State Register

Name: SCB_SHCSR

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
–							

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

15	14	13	12	11	10	9	8
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.

10.9.1.13 Configurable Fault Status Register

Name: SCB_CFSR

Access: Read/Write

Reset: 0x000000000

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	–		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.

- **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” .

10.9.1.14 Configurable Fault Status Register (Byte Access)

Name: SCB_CFSR (BYTE)

Access: Read/Write

Reset: 0x000000000

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 10.9.1.13.

- **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 10.9.1.13.

- **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 10.9.1.13.

Note: The UFSR bits are sticky. This means that as one or more fault occurs, the associated bits are set to 1. A bit that is set to 1 is cleared to 0 only by wrting a 1 to that bit, or by a reset.

The SCB_CFSR indicates the cause of a memory management fault, bus fault, or usage fault. It is byte accessible. The user can access the SCB_CFSR or its subregisters as follows:

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

10.9.1.15 Hard Fault Status Register

Name: SCB_HFSR

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
DEBUGEVT	FORCED	–					

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–							

7	6	5	4	3	2	1	0
–						VECTTBL	–

The SCB_HFSR gives information about events that activate the hard fault handler. This register is read, write to clear. This means that bits in the register read normally, but wrting a 1 to any bit clears that bit to 0.

- **DEBUGEVT: Reserved for Debug Use**

When writing to the register, write a 0 to this bit, otherwise the behavior is unpredictable.

- **FORCED: Forced Hard Fault**

It indicates a forced hard fault, generated by escalation of a fault with configurable priority that cannot be handles, either because of priority or because it is disabled:

0: No forced hard fault.

1: Forced hard fault.

When this bit is set to 1, the hard fault handler must read the other fault status registers to find the cause of the fault.

- **VECTTBL: Bus Fault on a Vector Table**

It indicates a bus fault on a vector table read during an exception processing:

0: No bus fault on vector table read.

1: Bus fault on vector table read.

This error is always handled by the hard fault handler.

When this bit is set to 1, the PC value stacked for the exception return points to the instruction that was preempted by the exception.

Note: The HFSR bits are sticky. This means that, as one or more fault occurs, the associated bits are set to 1. A bit that is set to 1 is cleared to 0 only by wrting a 1 to that bit, or by a reset.

10.9.1.16 MemManage Fault Address Register

Name: SCB_MMFAR

Access: Read/Write

Reset: 0x000000000

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” .

10.9.1.17 Bus Fault Address Register

Name: SCB_BFAR

Access: Read/Write

Reset: 0x000000000

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” .

10.10 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.

10.10.1 System Timer (SysTick) User Interface

Table 10-35. System Timer (SYST) Register Mapping

Offset	Register	Name	Access	Reset
0xE000E010	SysTick Control and Status Register	SYST_CSR	Read/Write	0x00000004
0xE000E014	SysTick Reload Value Register	SYST_RVR	Read/Write	Unknown
0xE000E018	SysTick Current Value Register	SYST_CVR	Read/Write	Unknown
0xE000E01C	SysTick Calibration Value Register	SYST_CALIB	Read-only	0x00001770

10.10.1.1 SysTick Control and Status

Name: SYST_CSR

Access: Read/Write

Reset: 0x000000000

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.

10.10.1.2 SysTick Reload Value Registers

Name: SYST_RVR

Access: Read/Write

Reset: 0x000000000

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.

10.10.1.3 SysTick Current Value Register

Name: SYST_CVR

Access: Read/Write

Reset: 0x000000000

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.

10.10.1.4 SysTick Calibration Value Register

Name: SYST_CALIB

Access: Read/Write

Reset: 0x00001770

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 0x00001770 (6000 decimal).

In order to achieve a 1 ms timebase on SystTick, the TENMS field must be programmed to a value corresponding to the processor clock frequency (in kHz) divided by 8.

For example, for devices running the processor clock at 48 MHz, the TENMS field value must be 0x0001770 (48000 kHz/8).

10.11 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 10-36 shows the possible MPU region attributes. These include Share ability and cache behavior attributes that are not relevant to most microcontroller implementations. See “MPU Configuration for a Microcontroller” for guidelines for programming such an implementation.

Table 10-36. 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.

10.11.1 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 10-37. TEX, C, B, and S Encoding

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

Note: 1. The MPU ignores the value of this bit.

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

Table 10-38. Cache Policy for Memory Attribute Encoding

Encoding, AA or BB	Corresponding Cache Policy
00	Non-cacheable
01	Write back, write and read allocate
10	Write through, no write allocate
11	Write back, no write allocate

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

Table 10-39. AP Encoding

AP[2:0]	Privileged Permissions	Unprivileged Permissions	Description
000	No access	No access	All accesses generate a permission fault
001	RW	No access	Access from privileged software only
010	RW	RO	Writes by unprivileged software generate a permission fault
011	RW	RW	Full access
100	Unpredictable	Unpredictable	Reserved
101	RO	No access	Reads by privileged software only
110	RO	RO	Read only, by privileged or unprivileged software
111	RO	RO	Read only, by privileged or unprivileged software

10.11.1.1 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.

10.11.1.2 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.

10.11.1.3 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.

10.11.1.4 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
```

10.11.1.5 Subregions

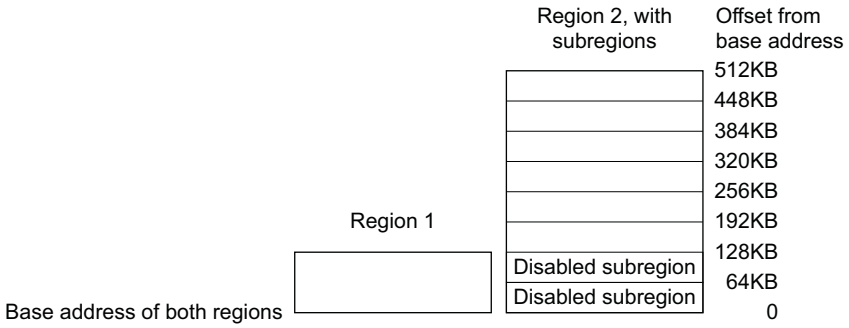
Regions of 256 bytes or more are divided into eight equal-sized subregions. Set the corresponding bit in the SRD field of the MPU_RASR field to disable a subregion. See "MPU Region Attribute and Size Register" . The least significant bit of SRD controls the first subregion, and the most significant bit 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.

10.11.1.6 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 10-13 below:

Figure 10-13. SRD Use



10.11.1.7 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 10-40. Memory Region Attributes for a Microcontroller

Memory Region	TEX	C	B	S	Memory Type and Attributes
Flash memory	b000	1	0	0	Normal memory, non-shareable, write-through
Internal SRAM	b000	1	0	1	Normal memory, shareable, write-through
External SRAM	b000	1	1	1	Normal memory, shareable, write-back, write-allocate
Peripherals	b000	0	1	1	Device memory, shareable

In most microcontroller implementations, the 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.

10.11.2 Memory Protection Unit (MPU) User Interface

Table 10-41. 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

10.11.2.1 MPU Type Register

Name: MPU_TYPE

Access: Read/Write

Reset: 0x00000800

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
IREGION							

15	14	13	12	11	10	9	8
DREGION							

7	6	5	4	3	2	1	0
–							SEPARATE

The MPU_TYPE register indicates whether the MPU is present, and if so, how many regions it supports.

- **IREGION: Instruction Region**

Indicates the number of supported MPU instruction regions.

Always contains 0x00. The MPU memory map is unified and is described by the DREGION field.

- **DREGION: Data Region**

Indicates the number of supported MPU data regions:

0x08 = Eight MPU regions.

- **SEPARATE: Separate Instruction**

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

0: Unified.

10.11.2.2 MPU Control Register

Name: MPU_CTRL

Access: Read/Write

Reset: 0x00000800

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–							

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

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

- **PRIVDEFENA: Privileged Default Memory Map 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.

10.11.2.3 MPU Region Number Register

Name: MPU_RNR

Access: Read/Write

Reset: 0x00000800

31	30	29	28	27	26	25	24
–							

23	22	21	20	19	18	17	16
–							

15	14	13	12	11	10	9	8
–							

7	6	5	4	3	2	1	0
REGION							

The MPU_RNR selects which memory region is referenced by the MPU_RBAR and MPU_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.

10.11.2.4 MPU Region Base Address Register

Name: MPU_RBAR

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	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.

10.11.2.5 MPU Region Attribute and Size Register

Name: MPU_RASR

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–			XN	–	AP		

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

15	14	13	12	11	10	9	8
SRD							

7	6	5	4	3	2	1	0
–			SIZE				ENABLE

The MPU_RASR defines the region size and memory attributes of the MPU region specified by the MPU_RNR, and enables that region and any subregions.

MPU_RASR is accessible using word or halfword accesses:

- The most significant halfword holds the region attributes.
- The least significant halfword holds the region size, and the region and subregion enable bits.

- **XN: Instruction Access Disable**

0: Instruction fetches enabled.

1: Instruction fetches disabled.

- **AP: Access Permission**

See Table 10-39.

- **TEX, C, B: Memory Access Attributes**

See Table 10-37.

- **S: Shareable**

See Table 10-37.

- **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” .

10.11.2.6 MPU Region Base Address Register Alias 1

Name: MPU_RBAR_A1

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	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.

10.11.2.7 MPU Region Attribute and Size Register Alias 1

Name: MPU_RASR_A1

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–			XN	–	AP		

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

15	14	13	12	11	10	9	8
SRD							

7	6	5	4	3	2	1	0
–			SIZE				ENABLE

The MPU_RASR defines the region size and memory attributes of the MPU region specified by the MPU_RNR, and enables that region and any subregions.

MPU_RASR is accessible using word or halfword accesses:

- The most significant halfword holds the region attributes.
- The least significant halfword holds the region size, and the region and subregion enable bits.

- **XN: Instruction Access Disable**

0: Instruction fetches enabled.

1: Instruction fetches disabled.

- **AP: Access Permission**

See Table 10-39.

- **TEX, C, B: Memory Access Attributes**

See Table 10-37.

- **S: Shareable**

See Table 10-37.

- **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” .

10.11.2.8 MPU Region Base Address Register Alias 2

Name: MPU_RBAR_A2

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	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.

10.11.2.9 MPU Region Attribute and Size Register Alias 2

Name: MPU_RASR_A2

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–			XN	–	AP		

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

15	14	13	12	11	10	9	8
SRD							

7	6	5	4	3	2	1	0
–			SIZE				ENABLE

The MPU_RASR defines the region size and memory attributes of the MPU region specified by the MPU_RNR, and enables that region and any subregions.

MPU_RASR is accessible using word or halfword accesses:

- The most significant halfword holds the region attributes.
- The least significant halfword holds the region size, and the region and subregion enable bits.

- **XN: Instruction Access Disable**

0: Instruction fetches enabled.

1: Instruction fetches disabled.

- **AP: Access Permission**

See Table 10-39.

- **TEX, C, B: Memory Access Attributes**

See Table 10-37.

- **S: Shareable**

See Table 10-37.

- **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” .

10.11.2.10 MPU Region Base Address Register Alias 3

Name: MPU_RBAR_A3

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	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.

10.11.2.11 MPU Region Attribute and Size Register Alias 3

Name: MPU_RASR_A3

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–			XN	–	AP		

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

15	14	13	12	11	10	9	8
SRD							

7	6	5	4	3	2	1	0
–			SIZE				ENABLE

The MPU_RASR defines the region size and memory attributes of the MPU region specified by the MPU_RNR, and enables that region and any subregions.

MPU_RASR is accessible using word or halfword accesses:

- The most significant halfword holds the region attributes.
- The least significant halfword holds the region size, and the region and subregion enable bits.

- **XN: Instruction Access Disable**

0: Instruction fetches enabled.

1: Instruction fetches disabled.

- **AP: Access Permission**

See Table 10-39.

- **TEX, C, B: Memory Access Attributes**

See Table 10-37.

- **S: Shareable**

See Table 10-37.

- **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” .

10.12 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.

10.12.1 Enabling the FPU

The FPU is disabled from reset. It must be enabled before any floating-point instructions can be used. Example 4-1 shows an example code sequence for enabling the FPU in both privileged and user modes. The processor must be in privileged mode to read from and write to the CPACR.

Example of Enabling the FPU:

```
; CPACR is located at address 0xE000ED88
LDR.W R0, =0xE000ED88
; Read CPACR
LDR R1, [R0]
; Set bits 20-23 to enable CP10 and CP11 coprocessors
ORR R1, R1, #(0xF << 20)
; Write back the modified value to the CPACR
STR R1, [R0]; wait for store to complete
DSB
;reset pipeline now the FPU is enabled
ISB
```

10.12.2 Floating Point Unit (FPU) User Interface

Table 10-42. 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

10.12.2.1 Coprocessor Access Control Register

Name: CPACR

Access: Read/Write

Reset: 0x000000000

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.

10.12.2.2 Floating-point Context Control Register

Name: FPCCR

Access: Read/Write

Reset: 0x000000000

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.

10.12.2.3 Floating-point Context Address Register

Name: FPCAR

Access: Read/Write

Reset: 0x000000000

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.

10.12.2.4 Floating-point Status Control Register

Name: FPSCR

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
N	Z	C	V	–	AHP	DN	FZ

23	22	21	20	19	18	17	16
RMode		–					

15	14	13	12	11	10	9	8
–							

7	6	5	4	3	2	1	0
IDC	–		IXC	UFC	OFC	DZC	IOC

The FPSCR 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.

10.12.2.5 Floating-point Default Status Control Register

Name: FPDSCR

Access: Read/Write

Reset: 0x000000000

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.

10.13 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.

11. Debug and Test Features

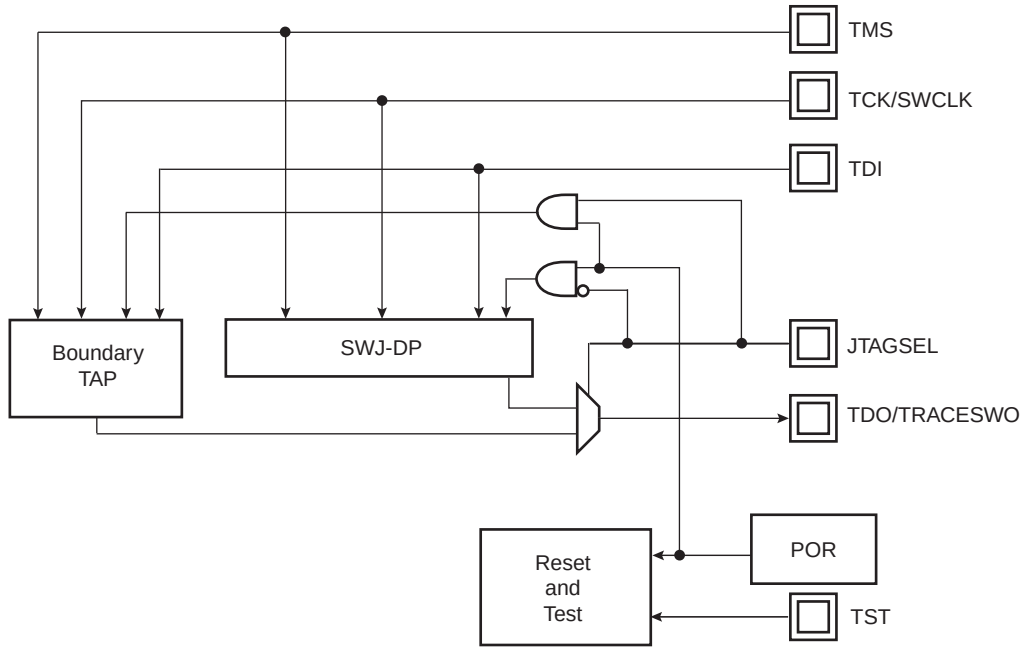
11.1 Description

The SAM G51 features a number of complementary debug and test capabilities. The Serial Wire/JTAG Debug Port (SWJ-DP) combining a Serial Wire Debug Port (SW-DP) and JTAG Debug 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.

11.2 Embedded Characteristics

- Debug access to all memories and registers in the system, including Cortex-M4 register bank when the core is running, halted, or held in reset.
- Serial Wire Debug Port (SW-DP) and Serial Wire JTAG Debug Port (SWJ-DP) debug access.
- Flash Patch and Breakpoint (FPB) unit for implementing breakpoints and code patches.
- Data Watchpoint and Trace (DWT) unit for implementing watchpoints, data tracing, and system profiling.
- Instrumentation Trace Macrocell (ITM) for support of printf style debugging.
- IEEE1149.1 JTAG Boundary-scan on all digital pins.

Figure 11-1. Debug and Test Block Diagram

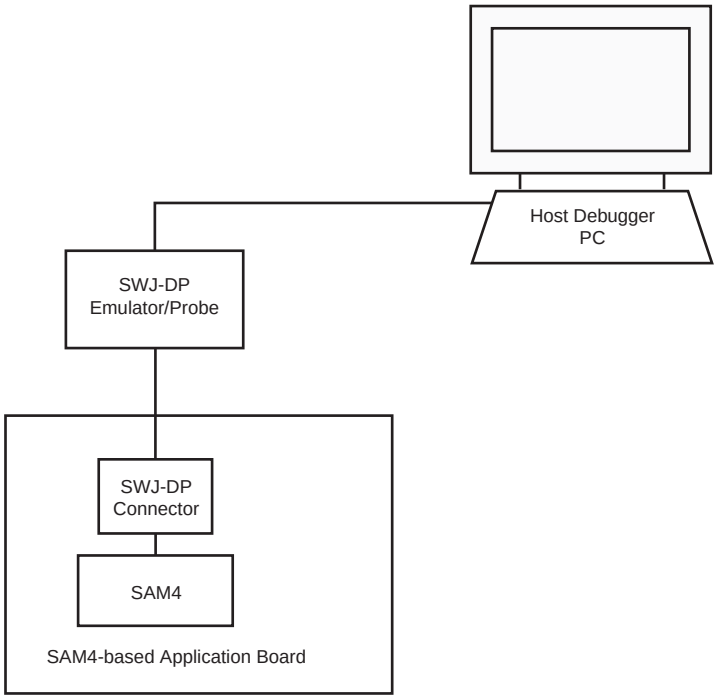


11.3 Application Examples

11.3.1 Debug Environment

Figure 11-2 shows a complete debug environment example. The SWJ-DP interface is used for standard debugging functions, such as downloading code and single-stepping through the program, and viewing core and peripheral registers.

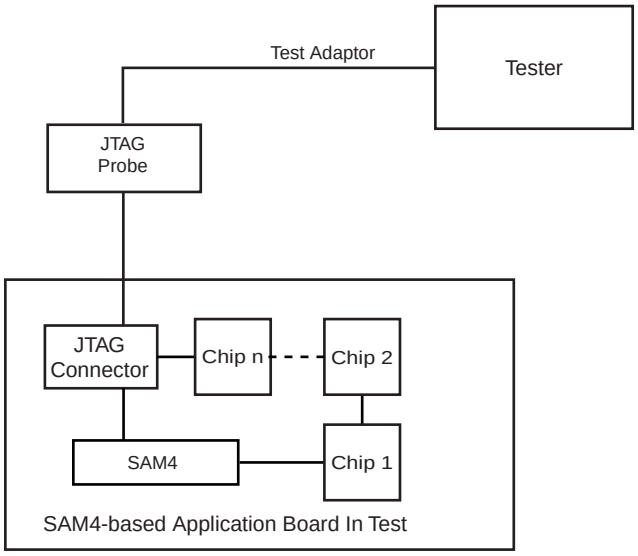
Figure 11-2. Application Debug Environment Example



11.3.2 Test Environment

Figure 11-3 shows a test environment example (JTAG boundary scan). Test vectors are sent and interpreted by the tester. In this example, the “board in test” is designed using a number of JTAG-compliant devices. These devices can be connected to form a single scan chain.

Figure 11-3. Application Test Environment Example



11.4 Debug and Test Pin Description

Table 11-1. Debug and Test Signal List

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

Note: 1. TDO pin is set in input mode when the Cortex-M4 Core is not in debug mode. Thus the internal pull-up corresponding to this PIO line must be enabled to avoid current consumption due to floating input.

11.5 Functional Description

11.5.1 Test Pin

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

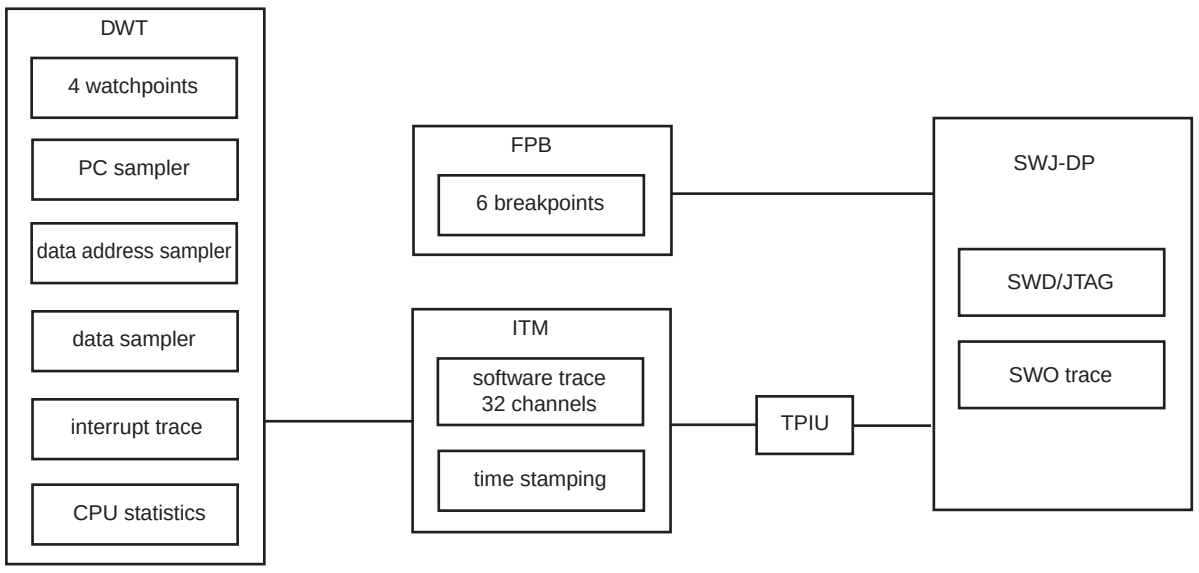
11.5.2 Debug Architecture

Figure 11-4 shows the Debug Architecture used in the SAM4. The Cortex-M4 embeds four functional units for debug:

- SWJ-DP (Serial Wire/JTAG Debug Port).
- FPB (Flash Patch Breakpoint.
- DWT (Data Watchpoint and Trace).
- ITM (Instrumentation Trace Macrocell).
- TPIU (Trace Port Interface Unit).

The debug architecture information that follows is mainly dedicated to developers of SWJ-DP emulators/probes and debugging tool vendors for Cortex M4-based microcontrollers. For further details on SWJ-DP see the Cortex M4 technical reference manual.

Figure 11-4. Debug Architecture



11.5.3 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 Serial Wire Debug Port (SW-DP), from 2 to 3 pins and JTAG Debug Port(JTAG-DP), 5 pins.

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

When the Serial Wire Debug Port is active, TDO/TRACESWO can be used for trace. The asynchronous TRACE output (TRACESWO) is multiplexed with TDO. So the asynchronous trace can only be used with SW-DP, not JTAG-DP.

Table 11-2. SWJ-DP Pin List

Pin Name	JTAG Port	Serial Wire Debug Port
TMS/SWDIO	TMS	SWDIO
TCK/SWCLK	TCK	SWCLK
TDI	TDI	-
TDO/TRACESWO	TDO	TRACESWO (optional: trace)

SW-DP or JTAG-DP mode is selected when JTAGSEL is low. It is not possible to switch directly between SWJ-DP and JTAG boundary scan operations. A chip reset must be performed after JTAGSEL is changed.

11.5.3.1 SW-DP and JTAG-DP Selection Mechanism

Debug port selection mechanism is done by sending specific SWDIOTMS sequence. The JTAG-DP is selected by default after reset.

- Switch from JTAG-DP to SW-DP. The sequence is:
  - Send more than 50 SWCLKTCK cycles with SWDIOTMS = 1
  - Send the 16-bit sequence on SWDIOTMS = 0111100111100111 (0x79E7 MSB first)
  - Send more than 50 SWCLKTCK cycles with SWDIOTMS = 1
- Switch from SWD to JTAG. The sequence is:
  - Send more than 50 SWCLKTCK cycles with SWDIOTMS = 1
  - Send the 16-bit sequence on SWDIOTMS = 0011110011100111 (0x3CE7 MSB first)
  - Send more than 50 SWCLKTCK cycles with SWDIOTMS = 1

11.5.4 FPB (Flash Patch Breakpoint)

The FPB:

- Implements hardware breakpoints.
- Patches code and data from code space to system space.

The FPB unit contains:

- Two literal comparators for matching against literal loads from code space, and remapping to a corresponding area in system space.
- Six instruction comparators for matching against instruction fetches from code space and remapping to a corresponding area in system space.
- Alternatively, comparators can also be configured to generate a breakpoint instruction to the processor core on a match.

11.5.5 DWT (Data Watchpoint and Trace)

The DWT contains four comparators which can be configured to generate the following:

- PC sampling packets at set intervals.
- PC or data watchpoint packets.
- Watchpoint event to halt core.

The DWT contains counters for the 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.

11.5.6 ITM (Instrumentation Trace Macrocell)

The ITM is an application driven trace source that supports printf style debugging to trace Operating System (OS) and application events, and emits diagnostic system information. The ITM emits trace information as packets which can be generated by three different sources with several priority levels:

- Software trace: Software can write directly to ITM stimulus registers. This can be done thanks to the “printf” function. For more information, refer to Section 11.5.6.1 “How to Configure the ITM”.
- Hardware trace: The ITM emits packets generated by the DWT.
- Time stamping: Timestamps are emitted relative to packets. The ITM contains a 21-bit counter to generate the timestamp.

11.5.6.1 How to Configure the ITM

The following example describes how to output trace data in asynchronous trace mode.

- Configure the TPIU for asynchronous trace mode (refer to Section 11.5.6.3 “How to Configure the TPIU”).
- Enable the write accesses into the ITM registers by writing “0xC5ACCE55” into the Lock Access Register (address: 0xE0000FB0).
- Write 0x00010015 into the Trace Control Register:
  - Enable ITM.
  - Enable synchronization packets.
  - Enable SWO behavior.
  - Fix the ATB ID to 1.
- Write 0x1 into the Trace Enable Register:
  - Enable the stimulus port 0.
- Write 0x1 into the Trace Privilege Register:
  - Stimulus port 0 only accessed in privileged mode (clearing a bit in this register will result in the corresponding stimulus port being accessible in user mode).
- Write into the Stimulus Port 0 Register: TPIU (Trace Port Interface Unit).
  - The TPIU acts as a bridge between the on-chip trace data and the Instruction Trace Macrocell (ITM).
  - The TPIU formats and transmits trace data off-chip at frequencies asynchronous to the core.

11.5.6.2 Asynchronous Mode

The TPIU is configured in asynchronous mode, trace data are output using the single TRACESWO pin. The TRACESWO signal is multiplexed with the TDO signal of the JTAG Debug Port. As a consequence, asynchronous trace mode is only available when the serial wire debug mode is selected since TDO signal is used in JTAG debug mode.

Two encoding formats are available for the single pin output:

- Manchester encoded stream. This is the reset value.
- NRZ-based UART byte structure.

11.5.6.3 How to Configure the TPIU

This example only concerns the asynchronous trace mode.

- Set the TRCENA bit to 1 into the Debug Exception and Monitor Register (0xE000EDFC) to enable the use of trace and debug blocks.
- Write 0x2 into the Selected Pin Protocol Register.
  - Select the Serial Wire Output – NRZ.
- Write 0x100 into the Formatter and Flush Control Register.

- Set the suitable clock prescaler value into the Async Clock Prescaler Register to scale the baud rate of the asynchronous output (this can be done automatically by the debugging tool).

11.5.7 IEEE 1149.1 JTAG Boundary Scan

IEEE 1149.1 JTAG Boundary Scan allows pin-level access independent of the device packaging technology.

IEEE 1149.1 JTAG Boundary Scan is enabled when TST is tied to low, while JTAG SEL is high during power-up and must be kept in this state during the whole boundary scan operation. The SAMPLE, EXTEST and BYPASS functions are implemented. In SWD/JTAG debug mode, the ARM processor responds with a non-JTAG chip ID that identifies the processor. This is not IEEE 1149.1 JTAG-compliant.

It is not possible to switch directly between JTAG Boundary Scan and SWJ Debug Port operations. A chip reset must be performed after JTAGSEL is changed. A Boundary-scan Descriptor Language (BSDL) file is provided on Atmel's web site to set up the test.

11.5.7.1 JTAG Boundary-scan Register

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

Each SAM4 input/output pin corresponds to a 3-bit register in the BSR. The OUTPUT bit contains data that can be forced on the pad. The INPUT bit facilitates the observability of data applied to the pad. The CONTROL bit selects the direction of the pad.

For more information, please refer to BSDL files available for the SAM4 Series.

11.5.8 ID Code Register

Access: Read-only

31	30	29	28	27	26	25	24
VERSION				PART NUMBER			

23	22	21	20	19	18	17	16
PART NUMBER							

15	14	13	12	11	10	9	8
PART NUMBER				MANUFACTURER IDENTITY			

7	6	5	4	3	2	1	0
MANUFACTURER IDENTITY							1

- **VERSION[31:28]: Product Version Number**

Set to 0x0.

- **PART NUMBER[27:12]: Product Part Number**

Chip Name	Chip ID
SAM G51G18	0x243B_09E0
SAM G51N18	0x243B_09E8

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

Set to 0x01F.

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

Set to 0x1.

Chip Name	JTAG ID Code
SAM G51	0x05B3_A03F

12. Reset Controller (RSTC)

12.1 Description

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

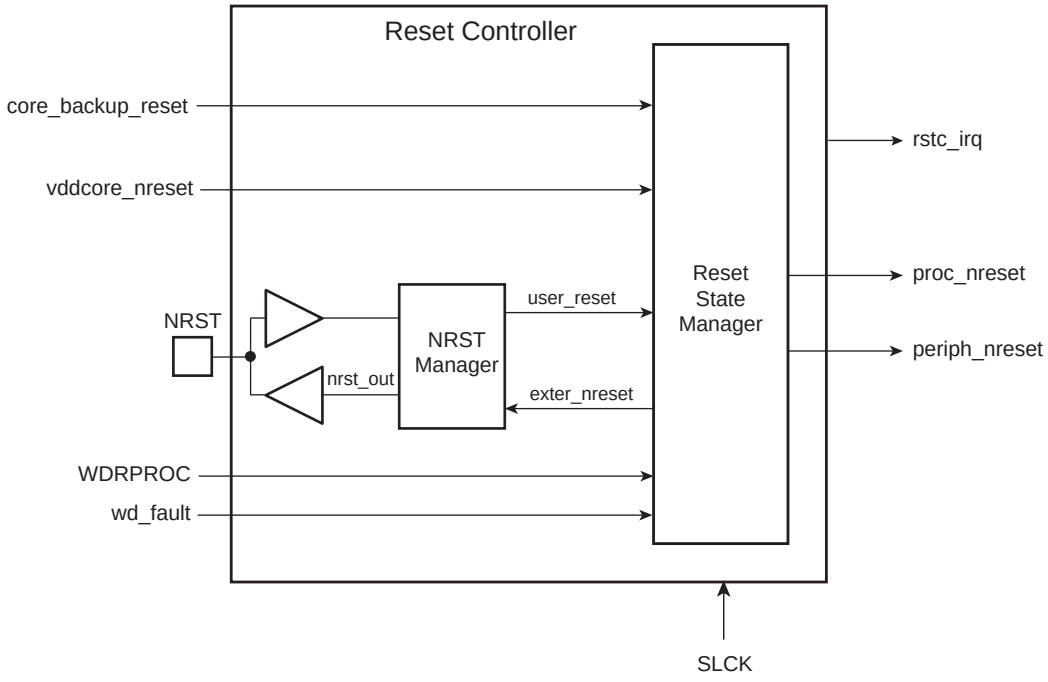
The Reset Controller also drives independently or simultaneously the external reset and the peripheral and processor resets.

12.2 Embedded Characteristics

- Management of All System Resets, Including
  - External Devices through the NRST Pin
  - Processor Reset
  - Processor Peripheral Set Reset
- Based on Embedded Power-on Cell
- Reset Source Status
  - Status of the Last Reset
  - Either Software Reset, User Reset, Watchdog Reset
- External Reset Signal Shaping

12.3 Block Diagram

Figure 12-1. Reset Controller Block Diagram



12.4 Functional Description

12.4.1 Reset Controller Overview

The Reset Controller is made up of an NRST Manager and a Reset State Manager. It runs at Slow Clock and generates the following reset signals:

- proc_nreset: processor reset line. It also resets the Watchdog Timer.
- periph_nreset: affects the whole set of embedded peripherals
- nrst_out: drives the NRST pin

These reset signals are asserted by the Reset Controller, either on external events or on software action. The Reset State Manager controls the generation of reset signals and provides a signal to the NRST Manager when an assertion of the NRST pin is required.

The NRST Manager shapes the NRST assertion during a programmable time, thus controlling external device resets.

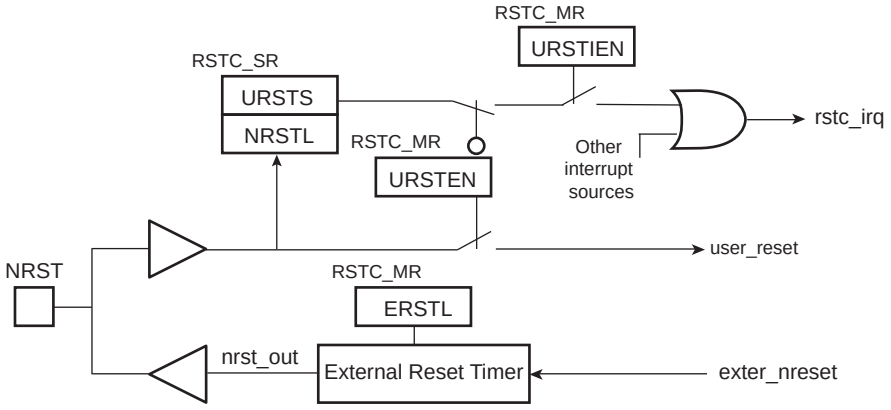
The Reset Controller Mode Register (RSTC_MR), allowing the configuration of the Reset Controller, is powered with VDDIO, so that its configuration is saved as long as VDDIO is on.

12.4.2 NRST Manager

After power-up, NRST is an output during the ERSTL time period defined in the RSTC_MR. When ERSTL has elapsed, the pin behaves as an input and all the system is held in reset if NRST is tied to GND by an external signal.

The NRST Manager samples the NRST input pin and drives this pin low when required by the Reset State Manager. Figure 12-2 shows the block diagram of the NRST Manager.

Figure 12-2. NRST Manager



12.4.2.1 NRST Signal or Interrupt

The NRST Manager samples the NRST pin at Slow Clock speed. When the line is detected low, a User Reset is reported to the Reset State Manager.

However, the NRST Manager can be programmed to not trigger a reset when an assertion of NRST occurs. Writing a 0 to bit URSTEN in the RSTC_MR disables the User Reset trigger.

The level of the pin NRST can be read at any time in bit NRSTL (NRST level) in the RSTC_SR. As soon as the pin NRST is asserted, bit URSTS in the RSTC_SR is set. This bit clears only when the RSTC_SR is read.

The Reset Controller can also be programmed to generate an interrupt instead of generating a reset. To do so, a 1 must be written to bit URSTIEN in the RSTC_MR.

12.4.2.2 NRST External Reset Control

The Reset State Manager asserts the signal exter_nreset to assert the NRST pin. When this occurs, the “nrst_out” signal is driven low by the NRST Manager for a time programmed by field ERSTL in the RSTC_MR. This assertion duration,

named EXTERNAL_RESET_LENGTH, lasts $2^{(ERSTL+1)}$ Slow Clock cycles. This gives the approximate duration of an assertion between 60 µs and 2 seconds. Note that ERSTL at 0 defines a two-cycle duration for the NRST pulse.

This feature allows the Reset Controller to shape the NRST pin level, and thus to guarantee that the NRST line is driven low for a time compliant with potential external devices connected on the system reset.

As the ERSTL field is in the RSTC_MR, which is backed-up, it can be used to shape the system power-up reset for devices requiring a longer startup time than the Slow Clock Oscillator.

12.4.3 Brownout Manager

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

12.4.4 Reset States

The Reset State Manager handles the different reset sources and generates the internal reset signals. It reports the reset status in field RSTTYP of the Status Register (RSTC_SR). The update of the field RSTTYP is performed when the processor reset is released.

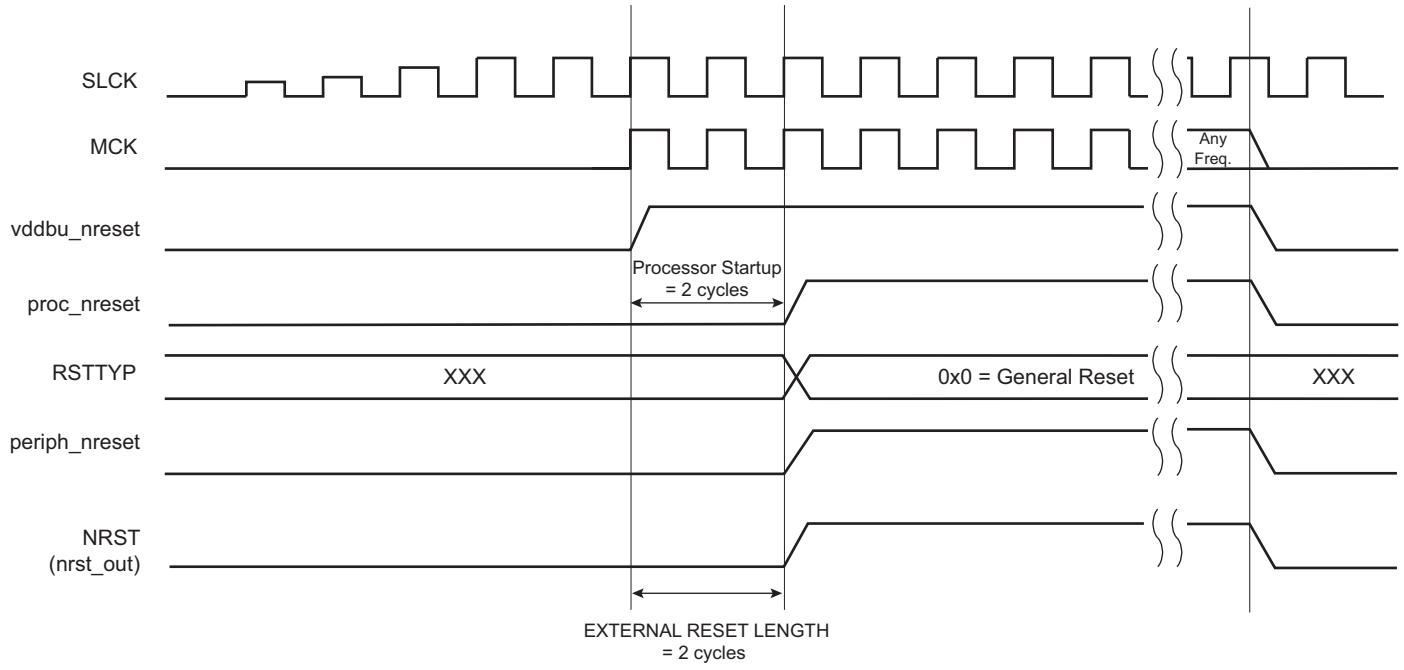
12.4.4.1 General Reset

A general reset occurs when a VDDIO 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 field RSTTYP in the RSTC_SR reports a General Reset. As the RSTC_MR is reset, the NRST line rises two cycles after the vddcore_nreset, as ERSTL defaults at value 0x0.

Figure 12-3 shows how the General Reset affects the reset signals.

Figure 12-3. General Reset State



12.4.4.2 Backup Reset

A Backup Reset occurs when the chip exits from Backup Mode. While exiting Backup Mode, the vddcore_nreset signal is asserted by the Supply Controller.

Field RSTTYP in the RSTC_SR is updated to report a Backup Reset.

12.4.4.3 User Reset

The User Reset is entered when a low level is detected on the NRST pin and bit URSTEN in the RSTC_MR is at 1. The NRST input signal is resynchronized with SLCK to insure proper behavior of the system.

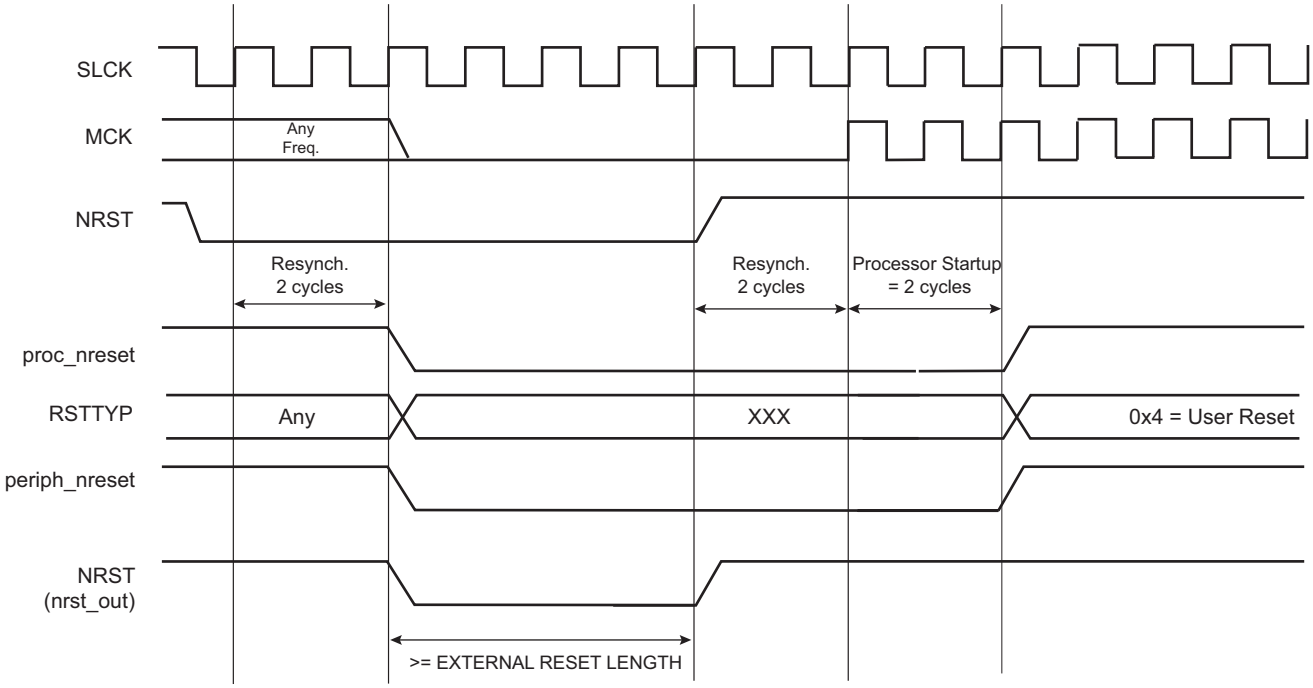
The User Reset is entered as soon as a low level is detected on NRST. The Processor Reset and the Peripheral Reset are asserted.

The User Reset is left when NRST rises, after a two-cycle resynchronization time and a 3-cycle processor startup. The processor clock is re-enabled as soon as NRST is confirmed high.

When the processor reset signal is released, field RSTTYP in the RSTC_SR is loaded with the value 0x4, indicating a User Reset.

The NRST Manager guarantees that the NRST line is asserted for EXTERNAL_RESET_LENGTH Slow Clock cycles, as programmed in field ERSTL. However, if NRST does not rise after EXTERNAL_RESET_LENGTH because it is driven low externally, the internal reset lines remain asserted until NRST actually rises.

Figure 12-4. User Reset State



12.4.4.4 Software Reset

The Reset Controller offers several commands used to assert the different reset signals. These commands are performed by writing the Control Register (RSTC_CR) with the following bits at 1:

- PROCRST: Writing PROCRST at 1 resets the processor and the watchdog timer.
- PERRST: Writing PERRST at 1 resets all the embedded peripherals including the memory system, and, in particular, the Remap Command. The Peripheral Reset is generally used for debug purposes.
  Except for debug purposes, PERRST must always be used in conjunction with PROCRST (PERRST and PROCRST set both at 1 simultaneously).
- EXTRST: Writing EXTRST at 1 asserts low the NRST pin during a time defined by the field ERSTL in the Mode Register (RSTC_MR).

The software reset is entered if at least one of these bits is set by the software. All these commands can be performed independently or simultaneously. The software reset lasts 3 Slow Clock cycles.

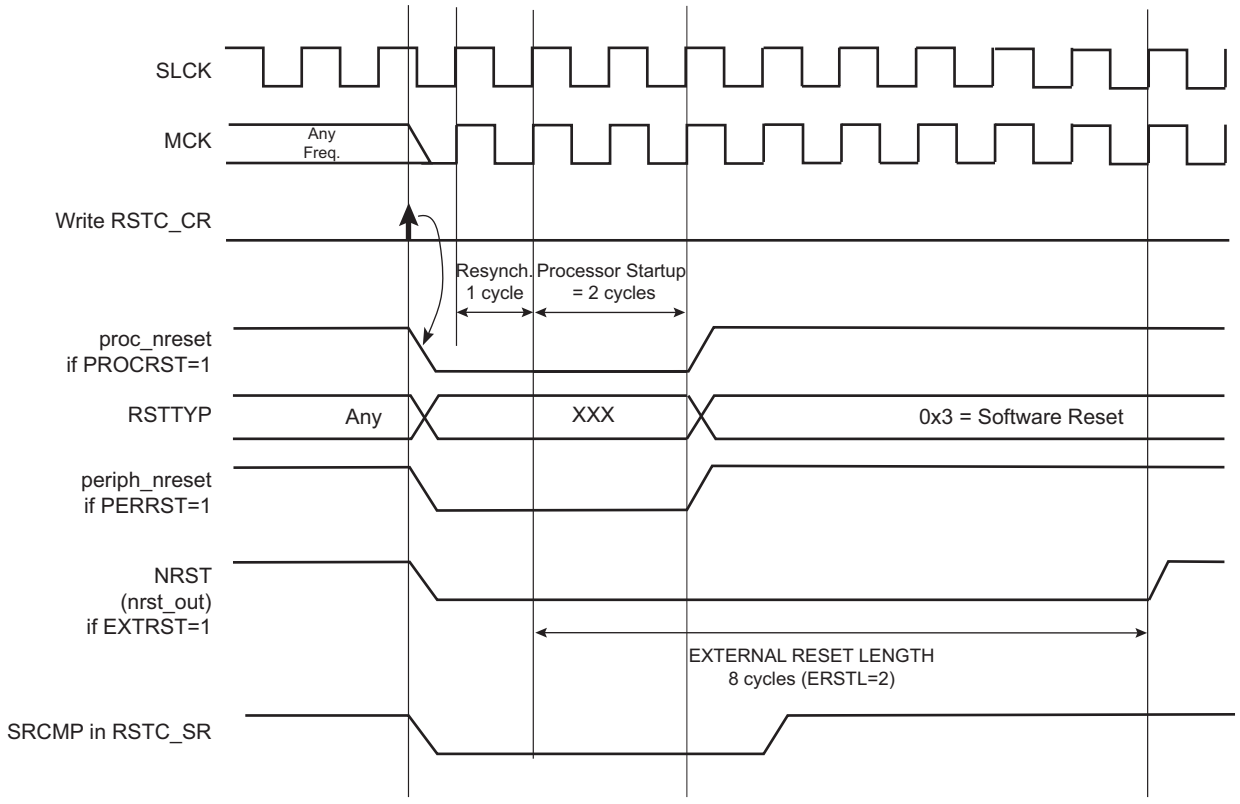
The internal reset signals are asserted as soon as the register write is performed. This is detected on the Master Clock (MCK). They are released when the software reset is left, i.e., synchronously to SLCK.

If EXTRST is set, the nrst_out signal is asserted depending on the programming of the field ERSTL. However, the resulting falling edge on NRST does not lead to a User Reset.

If and only if the PROCRST bit is set, the Reset Controller reports the software status in the field RSTTYP of the Status Register (RSTC_SR). Other Software Resets are not reported in RSTTYP.

As soon as a software operation is detected, the bit SRCMP (Software Reset Command in Progress) is set in the RSTC_SR. It is cleared as soon as the software reset is left. No other software reset can be performed while the SRCMP bit is set, and writing any value in the RSTC_CR has no effect.

Figure 12-5. Software Reset



12.4.4.5 Watchdog Reset

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

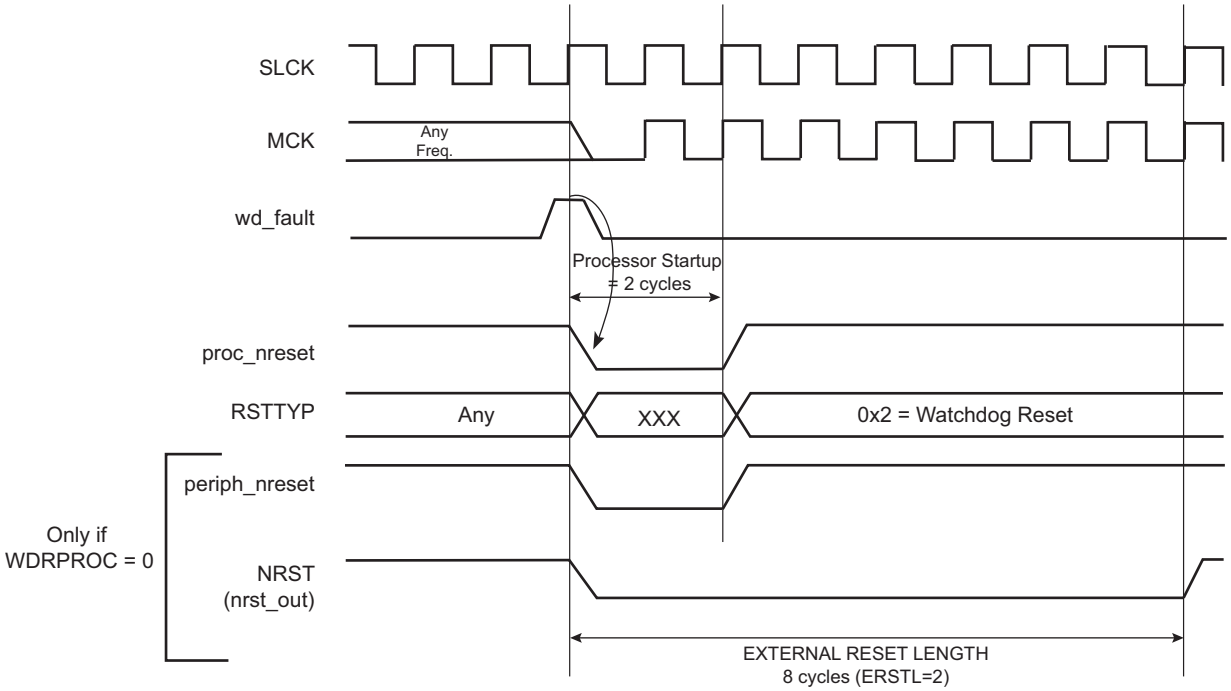
When in Watchdog Reset, assertion of the reset signals depends on the WDRPROC bit in the WDT_MR:

- If WDRPROC is 0, the Processor Reset and the Peripheral Reset are asserted. The NRST line is also asserted, depending on the programming of the field ERSTL. However, the resulting low level on NRST does not result in a User Reset state.
- If WDRPROC = 1, only the processor reset is asserted.

The Watchdog Timer is reset by the proc_nreset signal. As the watchdog fault always causes a processor reset if WDRSTEN is set, the Watchdog Timer is always reset after a Watchdog Reset, and the Watchdog is enabled by default and with a period set to a maximum.

When the WDRSTEN in the WDT_MR bit is reset, the watchdog fault has no impact on the reset controller.

Figure 12-6. Watchdog Reset



12.4.5 Reset State Priorities

The Reset State Manager manages the following priorities between the different reset sources, given in descending order:

- General Reset
- Backup Reset
- Watchdog Reset
- Software Reset
- User Reset

Particular cases are listed below:

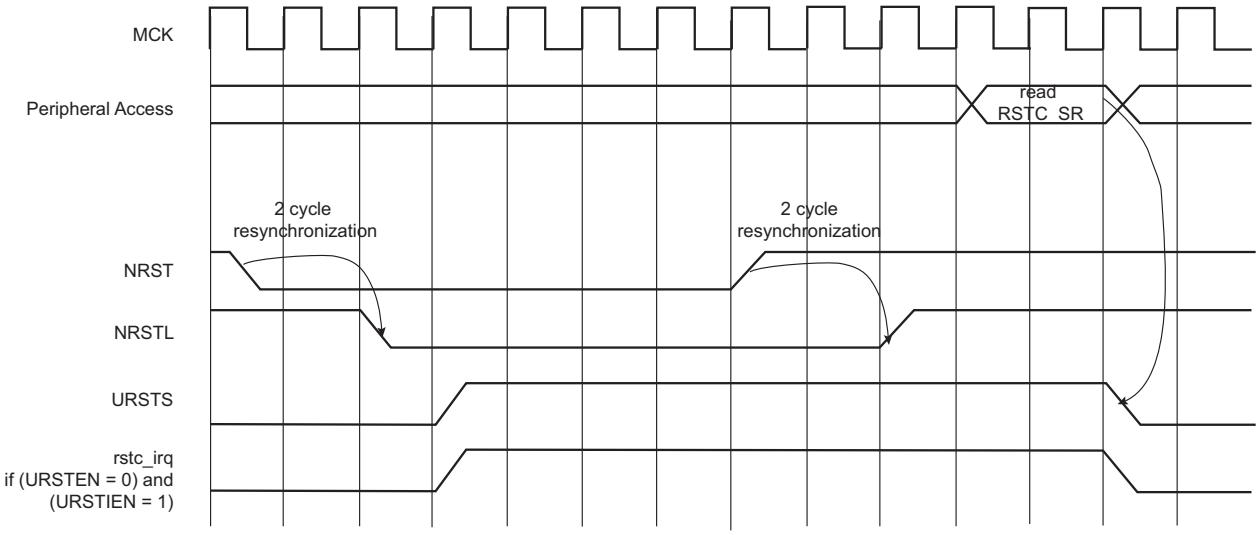
- When in User Reset:
  - A watchdog event is impossible because the Watchdog Timer is being reset by the proc_nreset signal.
  - A software reset is impossible, since the processor reset is being activated.
- When in Software Reset:
  - A watchdog event has priority over the current state.
  - The NRST has no effect.
- When in Watchdog Reset:
  - The processor reset is active and so a Software Reset cannot be programmed.
  - A User Reset cannot be entered.

12.4.6 Reset Controller Status Register

The Reset Controller status register (RSTC_SR) provides several status fields:

- RSTTYP field: This field gives the type of the last reset, as explained in previous sections.
- SRCMP bit: This field indicates that a Software Reset Command is in progress and that no further software reset should be performed until the end of the current one. This bit is automatically cleared at the end of the current software reset.
- NRSTL bit: The NRSTL bit of the Status Register gives the level of the NRST pin sampled on each MCK rising edge.
- URSTS bit: A high-to-low transition of the NRST pin sets the URSTS bit of the RSTC_SR. This transition is also detected on the Master Clock (MCK) rising edge (see Figure 12-7). If the User Reset is disabled (URSTEN = 0) and if the interruption is enabled by the URSTIEN bit in the RSTC_MR, the URSTS bit triggers an interrupt. Reading the RSTC_SR resets the URSTS bit and clears the interrupt.

Figure 12-7. Reset Controller Status and Interrupt



12.5 Reset Controller (RSTC) User Interface

Table 12-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	RSTC_CR	Write-only	–
0x04	Status Register	RSTC_SR	Read-only	0x0000_0000
0x08	Mode Register	RSTC_MR	Read/Write	0x0000 0001

12.5.1 Reset Controller Control Register

Name: RSTC_CR

Access: Write-only

31	30	29	28	27	26	25	24
KEY							

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	EXTRST	PERRST	–	PROCRST

- **PROCRST: Processor Reset**

0: No effect.

1: If KEY is correct, resets the processor.

- **PERRST: Peripheral Reset**

0: No effect.

1: If KEY is correct, resets the peripherals.

- **EXTRST: External Reset**

0: No effect.

1: If KEY is correct, asserts the NRST pin.

- **KEY: System Reset Key**

Value	Name	Description
0xA5	PASSWD	Writing any other value in this field aborts the write operation.

12.5.2 Reset Controller Status Register

Name: RSTC_SR

Access: Read-only

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

- **URSTS: User Reset Status**

0: No high-to-low edge on NRST happened since the last read of RSTC_SR.

1: At least one high-to-low transition of NRST has been detected since the last read of RSTC_SR.

- **RSTTYP: Reset Type**

Value	Name	Description
0	GENERAL_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

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

- **NRSTL: NRST Pin Level**

Registers the NRST Pin Level at Master Clock (MCK)

- **SRCMP: Software Reset Command in Progress**

0: No software command is being performed by the reset controller. The reset controller is ready for a software command.

1: A software reset command is being performed by the reset controller. The reset controller is busy.

12.5.3 Reset Controller Mode Register

Name: RSTC_MR

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 pin NRST does not generate a User Reset.

1: The detection of a low level on the pin NRST triggers a User Reset.

- **URSTIEN: User Reset Interrupt Enable**

0: USRTS bit in RSTC_SR at 1 has no effect on rstc_irq.

1: USRTS bit in RSTC_SR at 1 asserts rstc_irq if URSTEN = 0.

- **ERSTL: External Reset Length**

This field defines the external reset length. The external reset is asserted during a time of $2^{(ERSTL+1)}$ Slow Clock cycles. This allows assertion duration to be programmed between 60 µs and 2 seconds. 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.

13. Real-time Timer (RTT)

13.1 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 RTC 1 Hz signal, thus taking advantage of a calibrated 1 Hz clock.

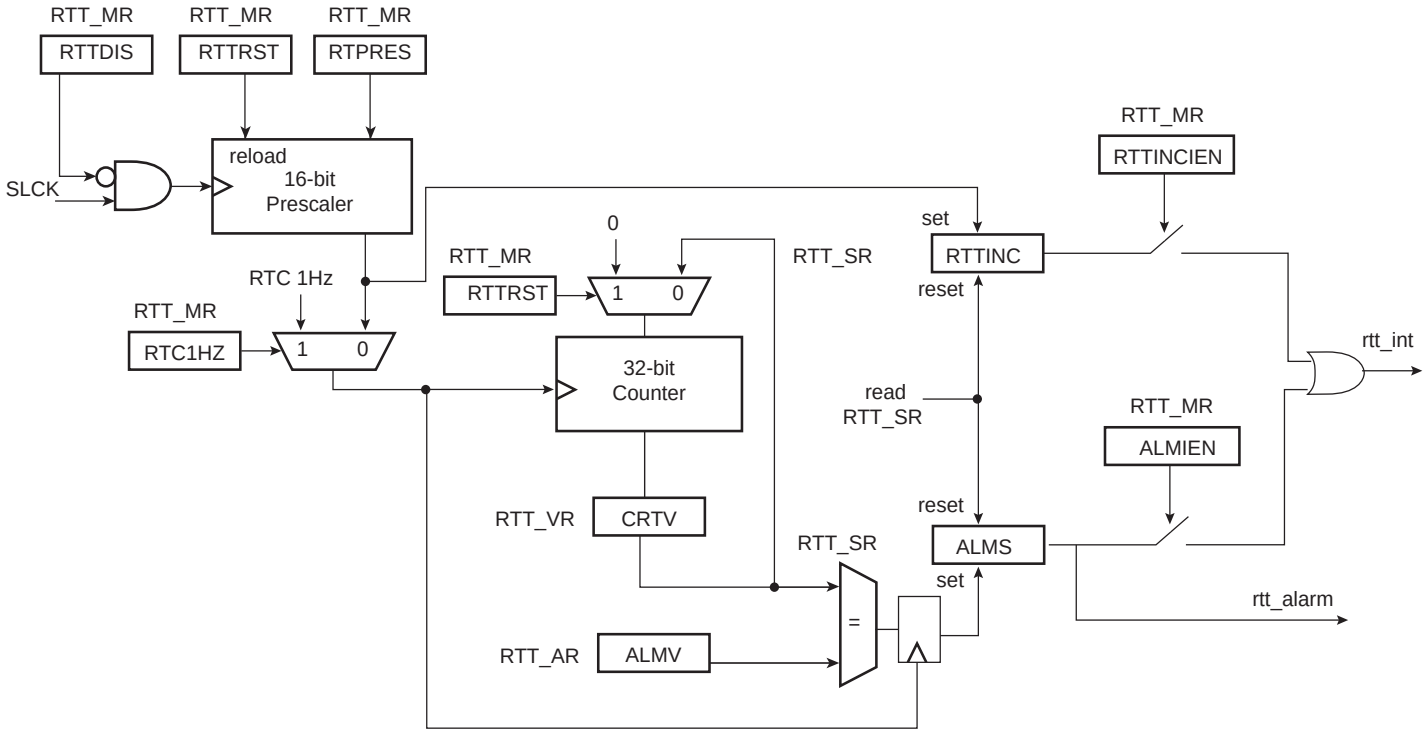
The slow clock source can be fully disabled to reduce power consumption when only an elapsed seconds count is required.

13.2 Embedded Characteristics

- 32-bit Free-running Counter on prescaled slow clock or RTC calibrated 1 Hz clock
- 16-bit Configurable Prescaler
- Interrupt on Alarm

13.3 Block Diagram

Figure 13-1. Real-time Timer



13.4 Functional Description

The programmable 16-bit prescaler value can be configured in the field RTPRES of 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 1 Hz signal (if the slow clock is 32.768 kHz). The 32-bit counter can count up to 2^{32} seconds, corresponding to more than 136 years, then roll over to 0. Bit RTTINC in the Real-time Timer Status Register (RTT_SR) is set each time there is a prescaler roll-over.

The real-time 32-bit counter can also be supplied by the RTC 1 Hz clock. This mode is interesting when the RTC 1 Hz is calibrated (CORRECTION field ≠ 0 in RTC_MR) in order to guaranty the synchronism between RTC and RTT counters.

Setting bit RTC1HZ in the RTT_MR drives the 32-bit RTT counter from the RTC 1 Hz 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, else if RTC1HZ = 1, the Real-time Timer counter is incremented every second. Bit RTTINC 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. Programming RTPRES to 1 or 2 is possible, but may result in losing status events because the Real-time Time Status Register (RTT_SR) is cleared two slow clock cycles after read. Thus 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 current real-time value (CRTV) can be read at any time in the Real-time Timer Value Register (RTT_VR). As this value can be updated asynchronously from the Master Clock, it is advisable to read this register twice at the same value to improve accuracy of the returned value.

The current value of the counter is compared with the value written in the Real-time Timer Alarm Register (RTT_AR). If the counter value matches the alarm, the bit ALMS in the RTT_SR is set. The RTT_AR is set to its maximum value (0xFFFF_FFFF) after a reset.

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 Hz.

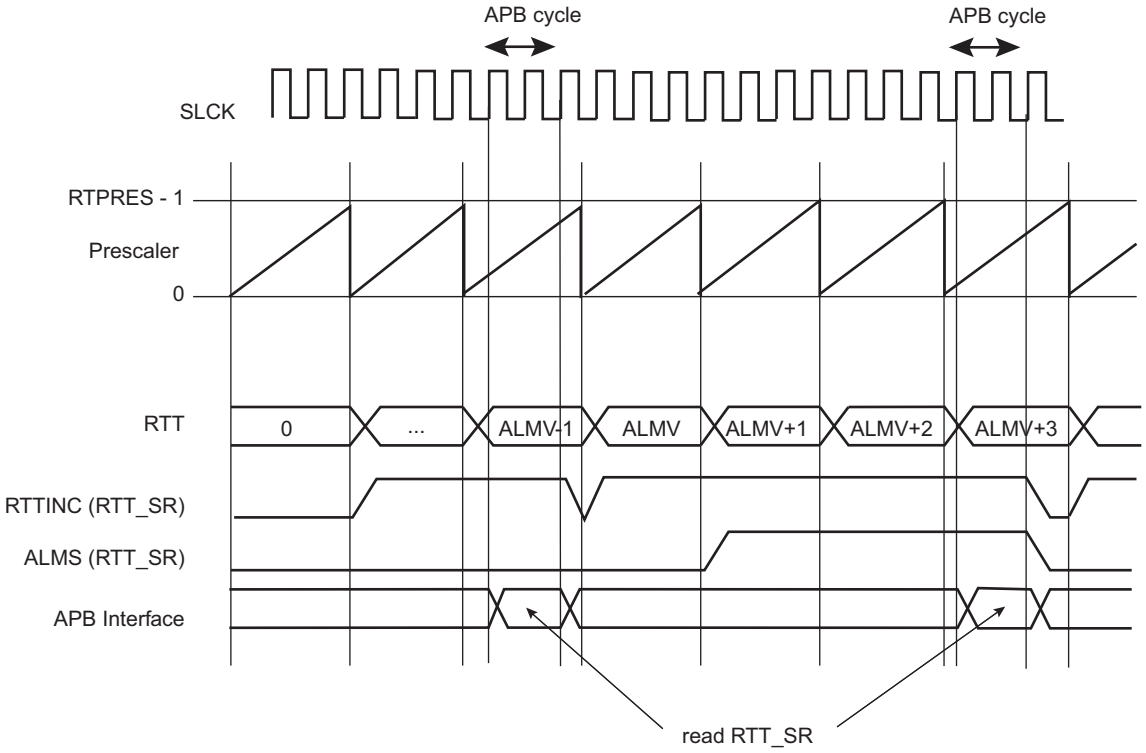
The RTTINCIEN bit must be cleared prior to writing a new RTPRES value in the RTT_MR.

Reading the RTT_SR automatically clears the RTTINC and ALMS bits.

Writing the bit RTTRST 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 13-2. RTT Counting



13.5 Real-time Timer (RTT) User Interface

Table 13-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Mode Register	RTT_MR	Read/Write	0x0000_8000
0x04	Alarm Register	RTT_AR	Read/Write	0xFFFF_FFFF
0x08	Value Register	RTT_VR	Read-only	0x0000_0000
0x0C	Status Register	RTT_SR	Read-only	0x0000_0000

13.5.1 Real-time Timer Mode Register

Name: RTT_MR

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 period.

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

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 1 Hz Clock Selection**

0: The RTT 32-bit counter is driven by the 16-bit prescaler roll-over events.

1: The RTT 32-bit counter is driven by the RTC 1 Hz clock.

Note: RTC1HZ is write only.

13.5.2 Real-time Timer Alarm Register

Name: RTT_AR

Access: Read/Write

31	30	29	28	27	26	25	24
ALMV							

23	22	21	20	19	18	17	16
ALMV							

15	14	13	12	11	10	9	8
ALMV							

7	6	5	4	3	2	1	0
ALMV							

- **ALMV: Alarm Value**

Defines the alarm value (ALMV+1) compared with the Real-time Timer.

Note: The alarm interrupt must be disabled (ALMIEN must be cleared in RTT_MR) when writing a new ALMV value.

13.5.3 Real-time Timer Value Register

Name: RTT_VR

Access: Read-only

31	30	29	28	27	26	25	24
CRTV							

23	22	21	20	19	18	17	16
CRTV							

15	14	13	12	11	10	9	8
CRTV							

7	6	5	4	3	2	1	0
CRTV							

- **CRTV: Current Real-time Value**

Returns the current value of the Real-time Timer.

13.5.4 Real-time Timer Status Register

Name: RTT_SR

Access: Read-only

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

- **ALMS: Real-time Alarm Status**

0: The Real-time Alarm has not occurred since the last read of RTT_SR.

1: The Real-time Alarm occurred since the last read of RTT_SR.

- **RTTINC: Prescaler Roll-over Status**

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.

14. Real-time Clock (RTC)

14.1 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 two-hundred-year Gregorian or Persian calendar, complemented by a programmable periodic interrupt. The alarm and calendar registers are accessed by a 32-bit data bus.

The time and calendar values are coded in binary-coded decimal (BCD) format. The time format can be 24-hour mode or 12-hour mode with an AM/PM indicator.

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

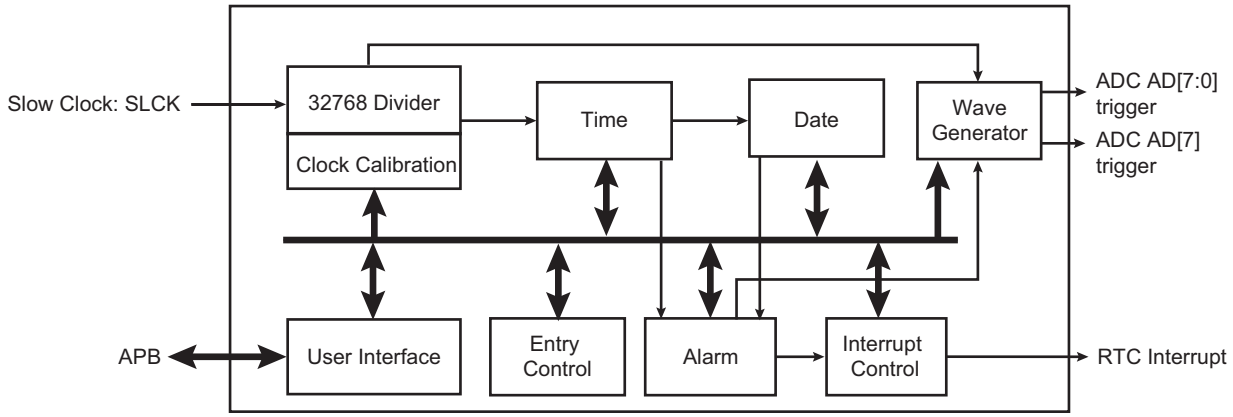
A clock divider calibration circuitry enables to compensate crystal oscillator frequency inaccuracy.

14.2 Embedded Characteristics

- Ultra Low Power Consumption
- Full Asynchronous Design
- Gregorian Calendar up to 2099 or Persian Calendar
- Programmable Periodic Interrupt
- Safety/security features:
  - Valid Time and Date Programmation Check
  - On-The-Fly Time and Date Validity Check
- Crystal Oscillator Clock Calibration
- Waveform Generation for trigger event
- Register Write Protection

14.3 Block Diagram

Figure 14-1. RTC Block Diagram



14.4 Product Dependencies

14.4.1 Power Management

The Real-time Clock is continuously clocked at 32.768 kHz. The Power Management Controller has no effect on RTC behavior.

14.4.2 Interrupt

RTC interrupt line is connected on one of the internal sources of the interrupt controller. RTC interrupt requires the interrupt controller to be programmed first.

14.5 Functional Description

The RTC provides a full binary-coded decimal (BCD) clock that includes century (19/20), year (with leap years), month, date, day, hours, minutes and seconds.

The valid year range is 1900 to 2099 in Gregorian mode, a two-hundred-year calendar(or 1300 to 1499 in Persian mode).

The RTC can operate in 24-hour mode or in 12-hour mode with an AM/PM indicator.

Corrections for leap years are included (all years divisible by 4 being leap years). This is correct up to the year 2099.

The RTC can generate events to trigger ADC measurements.

14.5.1 Reference Clock

The reference clock is Slow Clock (SLCK). It can be driven internally or by an external 32.768 kHz crystal.

During low power modes of the processor, the oscillator runs and power consumption is critical. The crystal selection has to take into account the current consumption for power saving and the frequency drift due to temperature effect on the circuit for time accuracy.

14.5.2 Timing

The RTC is updated in real time at one-second intervals in normal mode for the counters of seconds, at one-minute intervals for the counter of minutes and so on.

Due to the asynchronous operation of the RTC with respect to the rest of the chip, to be certain that the value read in the RTC registers (century, year, month, date, day, hours, minutes, seconds) are valid and stable, it is necessary to read these registers twice. If the data is the same both times, then it is valid. Therefore, a minimum of two and a maximum of three accesses are required.

14.5.3 Alarm

The RTC has five programmable fields: month, date, hours, minutes and seconds.

Each of these fields can be enabled or disabled to match the alarm condition:

- If all the fields are enabled, an alarm flag is generated (the corresponding flag is asserted and an interrupt generated if enabled) at a given month, date, hour/minute/second.
- If only the “seconds” field is enabled, then an alarm is generated every minute.

Depending on the combination of fields enabled, a large number of possibilities are available to the user ranging from minutes to 365/366 days.

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.

14.5.4 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_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.

14.5.5 RTC Internal Free Running Counter Error Checking

To improve the reliability and security of the RTC, a permanent check is performed on the internal free running counters to report non-BCD or invalid date/time values.

An error is reported by TDERR bit in the status register (RTC_SR) if an incorrect value has been detected. The flag can be cleared by programming the TDERRCLR in the RTC status clear control register (RTC_SCCR).

Anyway the TDERR error flag will be set again if the source of the error has not been cleared before clearing the TDERR flag. The clearing of the source of such error can be done either by reprogramming a correct value on RTC_CALR and/or RTC_TIMR.

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.

14.5.6 Updating Time/Calendar

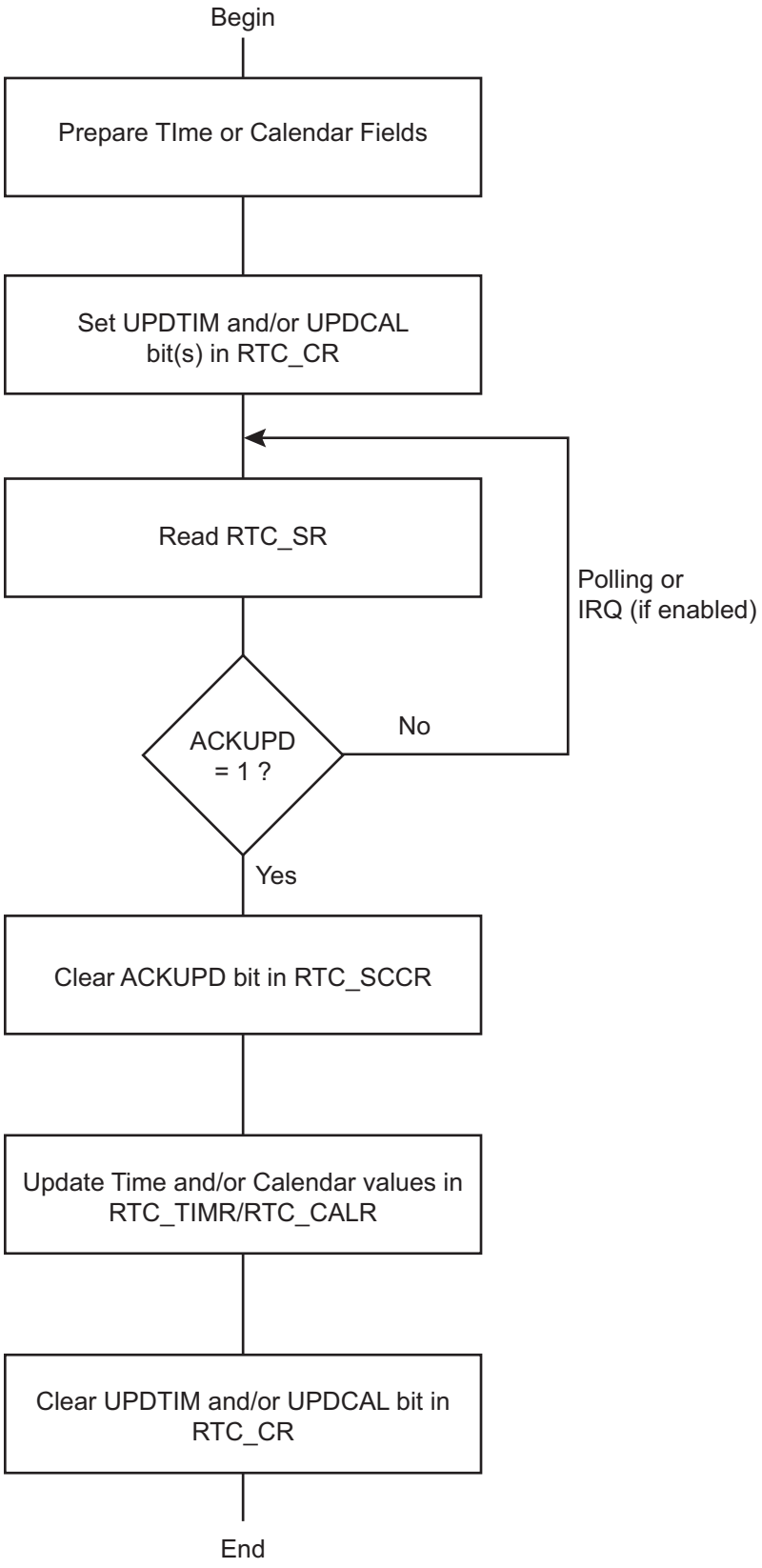
To update any of the time/calendar fields, the user must first stop the RTC by setting the corresponding field in the Control Register (RTC_CR). Bit UPDTIM must be set to update time fields (hour, minute, second) and bit UPDCAL must be set to update calendar fields (century, year, month, date, day).

The ACKUPD bit is automatically set within a second after setting the UPDTIM and/or UPDCAL bit (meaning one second is the maximum duration of the polling or wait for interrupt period). Once ACKUPD is set, 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 reset (0) UPDTIM and/or UPDCAL in the RTC_CR.

When entering programming mode of the calendar fields, the time fields remain enabled. When entering the programming mode of the time fields, both time and calendar fields are stopped. This is due to the location of the calendar logic circuity (downstream for low-power considerations). It is highly recommended to prepare all the fields to be updated before entering programming mode. In successive update operations, the user must wait at least one second after resetting the UPDTIM/UPDCAL bit in the RTC_CR before setting these bits again. This is done by waiting for the SEC flag in the RTC_SR before setting UPDTIM/UPDCAL bit. After resetting UPDTIM/UPDCAL, the SEC flag must also be cleared.

Figure 14-2. Update Sequence



14.5.7 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. After correction, the remaining crystal drift is as follows:

- Below 1 ppm, for an initial crystal drift between 1.5 ppm up to 90 ppm
- Below 2 ppm, for an initial crystal drift between 90 ppm up to 130 ppm
- Below 5 ppm, for an initial crystal drift between 130 ppm up to 200 ppm

The calibration circuitry acts by slightly modifying the 1 Hz clock period from time to time. When the period is modified, depending on the sign of the correction, the 1 Hz clock period increases or reduces by around 4 ms. According to the CORRECTION, NEGPPM and HIGHPPM values configured in the RTC Mode Register (RTC_MR), the period interval between two correction events differs.

The inaccuracy of a crystal oscillator at typical room temperature (±20 ppm at 20–25 degrees Celsius) can be compensated if a reference clock/signal is used to measure such inaccuracy. This kind of calibration operation can be set up during the final product manufacturing by means of measurement equipment embedding such a reference clock. The correction of value must be programmed into the (RTC_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.

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.

14.5.8 Waveform Generation

Waveforms can be generated by the RTC in order to take advantage of the RTC inherent prescalers while the RTC is the only powered circuitry (low power mode of operation, backup mode) or in any active modes. Going into backup or low power operating modes does not affect the waveform generation outputs.

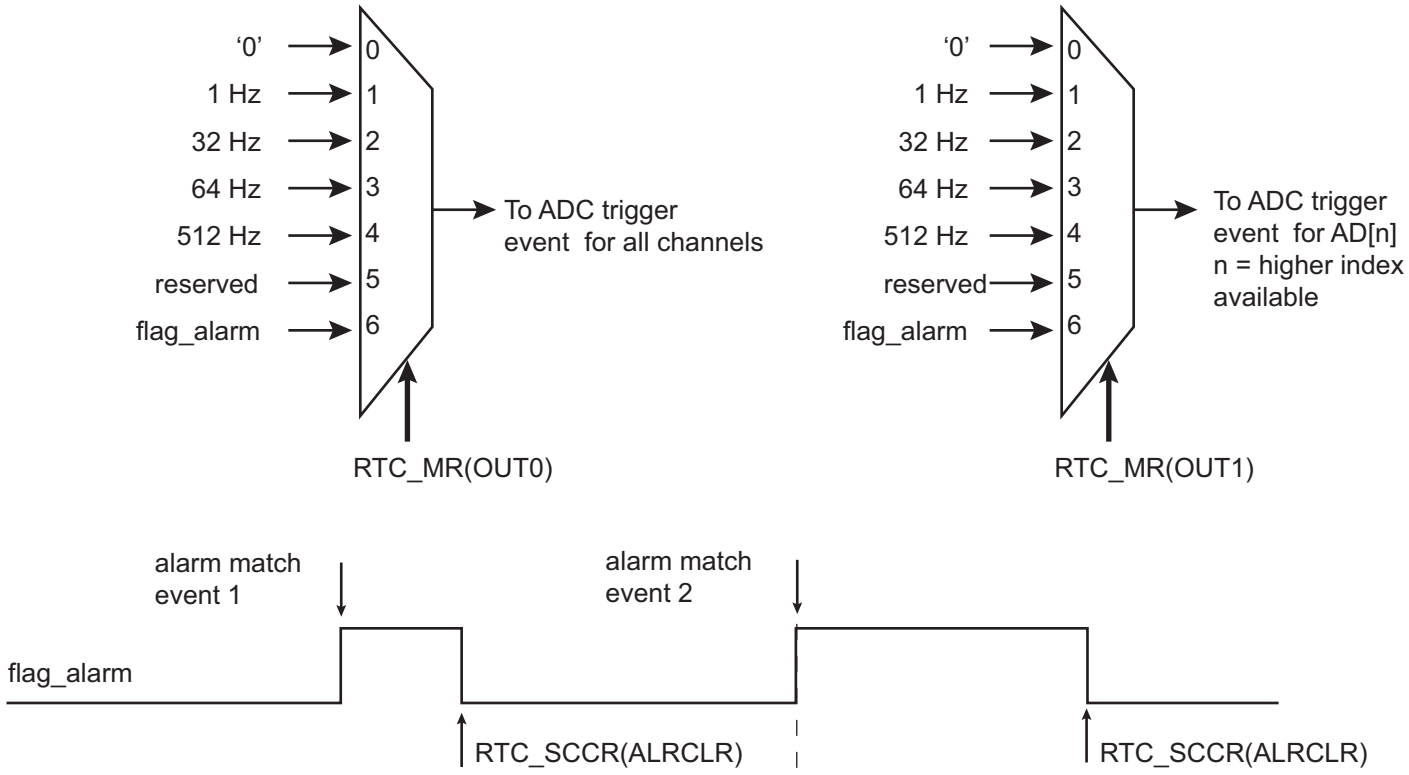
The RTC waveforms are internally routed to ADC trigger events and those events have a source driver selected among five possibilities. Two different triggers can be generated at a time, the first one is configurable through field OUT0 in RTC_MR while the second trigger is configurable through field OUT1 in RTC_MR. OUT0 field manages the trigger for channel AD[7:0] while OUT1 manages the channel AD[7] only for specific modes. See the ADC section for selection of the measurement triggers and associated mode of operations.

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.

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.

Figure 14-3. Waveform Generation for ADC trigger event



14.6 Real-time Clock (RTC) User Interface

Table 14-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	RTC_CR	Read/Write	0x0
0x04	Mode Register	RTC_MR	Read/Write	0x0
0x08	Time Register	RTC_TIMR	Read/Write	0x0
0x0C	Calendar Register	RTC_CALR	Read/Write	0x01A11020
0x10	Time Alarm Register	RTC_TIMALR	Read/Write	0x0
0x14	Calendar Alarm Register	RTC_CALALR	Read/Write	0x01010000
0x18	Status Register	RTC_SR	Read-only	0x0
0x1C	Status Clear Command Register	RTC_SCCR	Write-only	–
0x20	Interrupt Enable Register	RTC_IER	Write-only	–
0x24	Interrupt Disable Register	RTC_IDR	Write-only	–
0x28	Interrupt Mask Register	RTC_IMR	Read-only	0x0
0x2C	Valid Entry Register	RTC_VER	Read-only	0x0
0x30–0xC4	Reserved Register	–	–	–
0xC8–0xF8	Reserved Register	–	–	–
0xFC	Reserved Register	–	–	–

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

14.6.1 RTC Control Register

Name: RTC_CR

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the System Controller Write Protection Mode Register (SYSC_WPMR).

- **UPDTIM: Update Request Time Register**

0: No effect.

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.

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)

14.6.2 RTC Mode Register

Name: RTC_MR

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	OUT1			–	OUT0		
15	14	13	12	11	10	9	8
HIGHPPM	CORRECTION						
7	6	5	4	3	2	1	0
–	–	–	NEGPPM	–	–	PERSIAN	HRMOD

This register can only be written if the WPEN bit is cleared in the System Controller Write Protection Mode Register (SYSC_WPMR).

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

0: 24-hour mode is selected.

1: 12-hour mode is selected.

- **PERSIAN: PERSIAN Calendar**

0: Gregorian calendar.

1: Persian calendar.

- **NEGPPM: NEGative PPM Correction**

0: Positive correction (the divider will be slightly lower than 32768).

1: Negative correction (the divider will be slightly higher than 32768).

Refer to CORRECTION and HIGHPPM field descriptions.

- **CORRECTION: Slow Clock Correction**

0: No correction

1..127 = The slow clock will be corrected according to the formula given below in HIGHPPM description.

- **HIGHPPM: HIGH PPM Correction**

0: Lower range ppm correction with accurate correction.

1: Higher range ppm correction with accurate correction.

If the absolute value of the correction to be applied is lower than 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.

- **OUT0: All ADC Channel Trigger Event 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
6	ALARM_FLAG	output is a copy of the alarm flag

- **OUT1: ADC Last Channel Trigger Event 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
6	ALARM_FLAG	output is a copy of the alarm flag

14.6.3 RTC Time Register

Name: RTC_TIMR

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	AMPM	HOUR					

15	14	13	12	11	10	9	8
–	MIN						

7	6	5	4	3	2	1	0
–	SEC						

- **SEC: Current Second**

The range that can be set is 0–59 (BCD).

The lowest four bits encode the units. The higher bits encode the tens.

- **MIN: Current Minute**

The range that can be set is 0–59 (BCD).

The lowest four bits encode the units. The higher bits encode the tens.

- **HOUR: Current Hour**

The range that can be set is 1–12 (BCD) in 12-hour mode or 0–23 (BCD) in 24-hour mode.

- **AMPM: Ante Meridiem Post Meridiem Indicator**

This bit is the AM/PM indicator in 12-hour mode.

0: AM.

1: PM.

All non-significant bits read zero.

14.6.4 RTC Calendar Register

Name: RTC_CALR

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.

All non-significant bits read zero.

14.6.5 RTC Time Alarm Register

Name: RTC_TIMALR

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
HOUREN	AMPM	HOUR					
15	14	13	12	11	10	9	8
MINEN	MIN						
7	6	5	4	3	2	1	0
SECEN	SEC						

This register can only be written if the WPEN bit is cleared in the System Controller Write Protection Mode Register (SYSC_WPMR).

Note: To change one of the SEC, MIN, HOUR fields, it is recommended to disable the field before changing the value and then re-enable it after the change has been made. This requires up to three accesses to the RTC_TIMALR. The first access clears the enable corresponding to the field to change (SECEN, MINEN, HOUREN). If the field is already cleared, this access is not required. The second access performs the change of the value (SEC, MIN, HOUR). The third access is required to re-enable the field by writing 1 in SECEN, MINEN, HOUREN fields.

- **SEC: Second Alarm**

This field is the alarm field corresponding to the BCD-coded second counter.

- **SECEN: Second Alarm Enable**

0: The second-matching alarm is disabled.

1: The second-matching alarm is enabled.

- **MIN: Minute Alarm**

This field is the alarm field corresponding to the BCD-coded minute counter.

- **MINEN: Minute Alarm Enable**

0: The minute-matching alarm is disabled.

1: The minute-matching alarm is enabled.

- **HOUR: Hour Alarm**

This field is the alarm field corresponding to the BCD-coded hour counter.

- **AMPM: AM/PM Indicator**

This field is the alarm field corresponding to the BCD-coded hour counter.

- **HOUREN: Hour Alarm Enable**

0: The hour-matching alarm is disabled.

1: The hour-matching alarm is enabled.

14.6.6 RTC Calendar Alarm Register

Name: RTC_CALALR

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the System Controller Write Protection Mode Register (SYSC_WPMR).

Note: To change one of the DATE, MONTH fields, it is recommended to disable the field before changing the value and then re-enable it after the change has been made. This requires up to three accesses to the RTC_CALALR. The first access clears the enable corresponding to the field to change (DATEEN, MTHEN). If the field is already cleared, this access is not required. The second access performs the change of the value (DATE, MONTH). The third access is required to re-enable the field by writing 1 in DATEEN, MTHEN fields.

- **MONTH: Month Alarm**

This field is the alarm field corresponding to the BCD-coded month counter.

- **MTHEN: Month Alarm Enable**

0: The month-matching alarm is disabled.

1: The month-matching alarm is enabled.

- **DATE: Date Alarm**

This field is the alarm field corresponding to the BCD-coded date counter.

- **DATEEN: Date Alarm Enable**

0: The date-matching alarm is disabled.

1: The date-matching alarm is enabled.

14.6.7 RTC Status Register

Name: RTC_SR

Access: Read-only

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

- **ACKUPD: Acknowledge for Update**

0 (FREERUN): Time and calendar registers cannot be updated.

1 (UPDATE): Time and calendar registers can be updated.

- **ALARM: Alarm Flag**

0 (NO_ALARMEVENT): No alarm matching condition occurred.

1 (ALARMEVENT): An alarm matching condition has occurred.

- **SEC: Second Event**

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

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

- **TIMEV: Time Event**

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

1 (TIMEVENT): At least one time event has occurred since the last clear.

The time event is selected in the TIMEVSEL field in 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**

0 (NO_CALEVENT): No calendar event has occurred since the last clear.

1 (CALEVENT): At least one calendar event has occurred since the last clear.

The calendar event is selected in the CALEVSEL field in 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**

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.

14.6.8 RTC Status Clear Command Register

Name: RTC_SCCR

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).

14.6.9 RTC Interrupt Enable Register

Name: RTC_IER

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.

14.6.10 RTC Interrupt Disable Register

Name: RTC_IDR

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.

14.6.11 RTC Interrupt Mask Register

Name: RTC_IMR

Access: Read-only

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

- **ACK: Acknowledge Update Interrupt Mask**

0: The acknowledge for update interrupt is disabled.

1: The acknowledge for update interrupt is enabled.

- **ALR: Alarm Interrupt Mask**

0: The alarm interrupt is disabled.

1: The alarm interrupt is enabled.

- **SEC: Second Event Interrupt Mask**

0: The second periodic interrupt is disabled.

1: The second periodic interrupt is enabled.

- **TIM: Time Event Interrupt Mask**

0: The selected time event interrupt is disabled.

1: The selected time event interrupt is enabled.

- **CAL: Calendar Event Interrupt Mask**

0: The selected calendar event interrupt is disabled.

1: The selected calendar event interrupt is enabled.

14.6.12 RTC Valid Entry Register

Name: RTC_VER

Access: Read-only

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

- **NVTIM: Non-valid Time**

0: No invalid data has been detected in RTC_TIMR (Time Register).

1: RTC_TIMR has contained invalid data since it was last programmed.

- **NVCAL: Non-valid Calendar**

0: No invalid data has been detected in RTC_CALR (Calendar Register).

1: RTC_CALR has contained invalid data since it was last programmed.

- **NVTIMALR: Non-valid Time Alarm**

0: No invalid data has been detected in RTC_TIMALR (Time Alarm Register).

1: RTC_TIMALR has contained invalid data since it was last programmed.

- **NVCALALR: Non-valid Calendar Alarm**

0: No invalid data has been detected in RTC_CALALR (Calendar Alarm Register).

1: RTC_CALALR has contained invalid data since it was last programmed.

15. Watchdog Timer (WDT)

15.1 Description

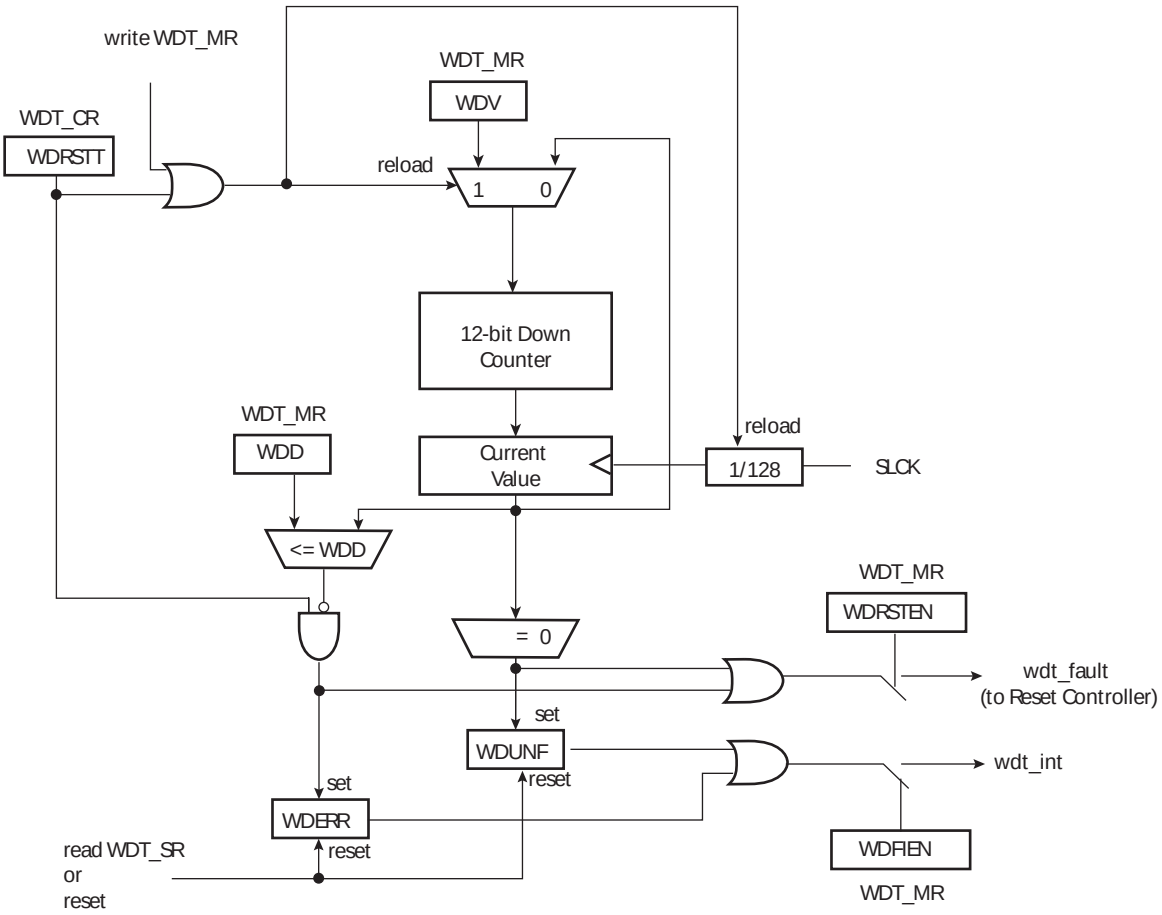
The Watchdog Timer (WDT) can be used to prevent system lock-up if the software becomes trapped in a deadlock. It features a 12-bit down counter that allows a watchdog period of up to 16 seconds (slow clock around 32 kHz). It can generate a general reset or a processor reset only. In addition, it can be stopped while the processor is in debug mode or idle mode.

15.2 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

15.3 Block Diagram

Figure 15-1. Watchdog Timer Block Diagram



15.4 Functional Description

The Watchdog Timer can be used to prevent system lock-up if the software becomes trapped in a deadlock. It is supplied with VDDCORE. It restarts with initial values on processor reset.

The Watchdog is built around a 12-bit down counter, which is loaded with the value defined in the field WDV of the Mode Register (WDT_MR). The Watchdog Timer uses the Slow Clock divided by 128 to establish the maximum Watchdog period to be 16 seconds (with a typical Slow Clock of 32.768 kHz).

After a Processor Reset, the value of WDV is 0xFFF, corresponding to the maximum value of the counter with the external reset generation enabled (field WDRSTEN at 1 after a Backup Reset). This means that a default Watchdog is running at reset, i.e., at power-up. The user must either disable it (by setting the WDDIS bit in WDT_MR) if he does not expect to use it or must reprogram it to meet the maximum Watchdog period the application requires.

If the watchdog is restarted by writing into the WDT_CR register, the WDT_MR register must not be programmed during a period of time of 3 slow clock periods following the WDT_CR write access. In any case, programming a new value in the WDT_MR register automatically initiates a restart instruction.

The Watchdog Mode Register (WDT_MR) can be written only once . Only a processor reset resets it. Writing the WDT_MR register reloads the timer with the newly programmed mode parameters.

In normal operation, the user reloads the Watchdog at regular intervals before the timer underflow occurs, by writing the Control Register (WDT_CR) with the bit WDRSTT to 1. The Watchdog counter is then immediately reloaded from WDT_MR and restarted, and the Slow Clock 128 divider is reset and restarted. The WDT_CR register is write-protected. As a result, writing WDT_CR without the correct hard-coded key has no effect. If an underflow does occur, the “wdt_fault” signal to the Reset Controller is asserted if the bit WDRSTEN is set in the Mode Register (WDT_MR). Moreover, the bit WDUNF is set in the Watchdog Status Register (WDT_SR).

To prevent a software deadlock that continuously triggers the Watchdog, the reload of the Watchdog must occur while the Watchdog counter is within a window between 0 and WDD, WDD is defined in the WatchDog Mode Register WDT_MR.

Any attempt to restart the Watchdog while the Watchdog counter is between WDV and WDD results in a Watchdog error, even if the Watchdog is disabled. The bit WDERR is updated in the WDT_SR and the “wdt_fault” signal to the Reset Controller is asserted.

Note that this feature can be disabled by programming a WDD value greater than or equal to the WDV value. In such a configuration, restarting the Watchdog Timer is permitted in the whole range [0; WDV] and does not generate an error. This is the default configuration on reset (the WDD and WDV values are equal).

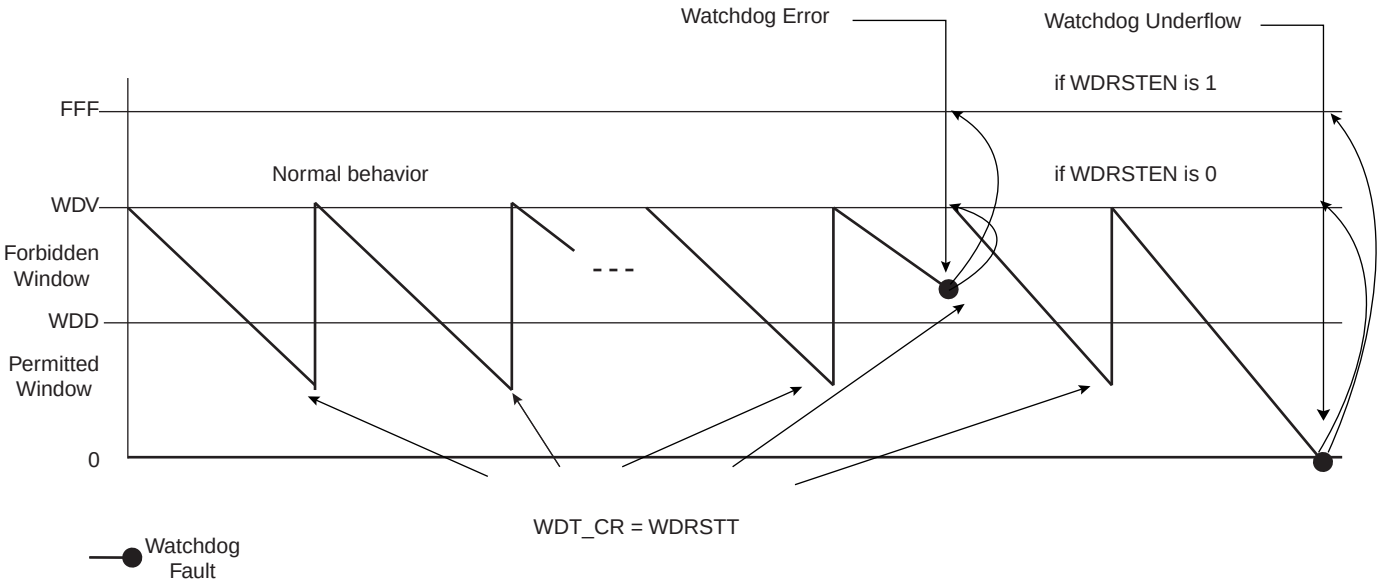
The status bits WDUNF (Watchdog Underflow) and WDERR (Watchdog Error) trigger an interrupt, provided the bit WDFIEN is set in the mode register. The signal “wdt_fault” to the reset controller causes a Watchdog reset if the WDRSTEN bit is set as already explained in the reset controller programmer Datasheet. In that case, the processor and the Watchdog Timer are reset, and the WDERR and WDUNF flags are reset.

If a reset is generated or if WDT_SR is read, the status bits are reset, the interrupt is cleared, and the “wdt_fault” signal to the reset controller is deasserted.

Writing the WDT_MR reloads and restarts the down counter.

While the processor is in debug state or in idle mode, the counter may be stopped depending on the value programmed for the bits WDIDLEHLT and WDDBGHLT in the WDT_MR.

Figure 15-2. Watchdog Behavior



15.5 Watchdog Timer (WDT) User Interface

Table 15-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	WDT_CR	Write-only	–
0x04	Mode Register	WDT_MR	Read-write Once	0x3FFF_2FFF
0x08	Status Register	WDT_SR	Read-only	0x0000_0000

15.5.1 Watchdog Timer Control Register

Name: WDT_CR

Address: 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

- **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.

15.5.2 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							

Note: The first write access prevents any further modification of the value of this register, read accesses remain possible.

Note: The WDD and WDV values must not be modified within a period of time of 3 slow clock periods following a restart of the watchdog performed by means of a write access in the WDT_CR register, else the watchdog may trigger an end of period earlier than expected.

- **WDV: Watchdog Counter Value**

Defines the value loaded in the 12-bit Watchdog Counter.

- **WDFIEN: Watchdog Fault Interrupt Enable**

0: A Watchdog fault (underflow or error) has no effect on interrupt.

1: A Watchdog fault (underflow or error) asserts interrupt.

- **WDRSTEN: Watchdog Reset Enable**

0: A Watchdog fault (underflow or error) has no effect on the resets.

1: A Watchdog fault (underflow or error) triggers a Watchdog reset.

- **WDRPROC: Watchdog Reset Processor**

0: If WDRSTEN is 1, a Watchdog fault (underflow or error) activates all resets.

1: If WDRSTEN is 1, a Watchdog fault (underflow or error) activates the processor reset.

- **WDD: Watchdog Delta Value**

Defines the permitted range for reloading the Watchdog Timer.

If the Watchdog Timer value is less than or equal to WDD, writing WDT_CR with WDRSTT = 1 restarts the timer.

If the Watchdog Timer value is greater than WDD, writing WDT_CR with WDRSTT = 1 causes a Watchdog error.

- **WDDBGHLT: Watchdog Debug Halt**

0: The Watchdog runs when the processor is in debug state.

1: The Watchdog stops when the processor is in debug state.

- **WDIDLEHLT: Watchdog Idle Halt**

0: The Watchdog runs when the system is in idle mode.

1: The Watchdog stops when the system is in idle state.

- **WDDIS: Watchdog Disable**

0: Enables the Watchdog Timer.

1: Disables the Watchdog Timer.

15.5.3 Watchdog Timer Status Register

Name: WDT_SR

Address: 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**

0: No Watchdog underflow occurred since the last read of WDT_SR.

1: At least one Watchdog underflow occurred since the last read of WDT_SR.

- **WDERR: Watchdog Error**

0: No Watchdog error occurred since the last read of WDT_SR.

1: At least one Watchdog error occurred since the last read of WDT_SR.

16. Supply Controller (SUPC)

16.1 Description

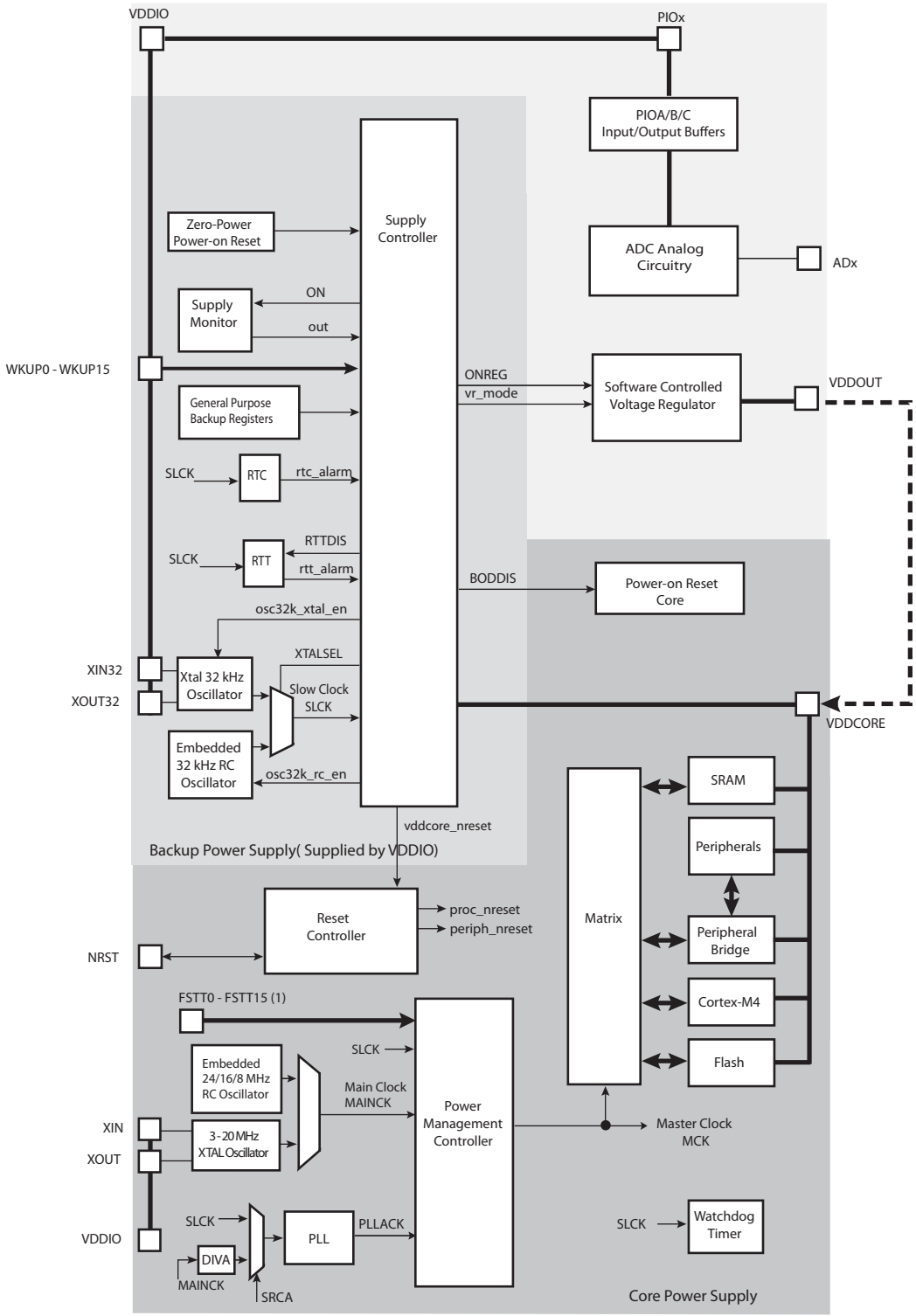
The Supply Controller (SUPC) controls the supply voltages of the system. The SUPC also generates the slow clock by selecting either the low-power RC oscillator or the low-power crystal oscillator.

16.2 Embedded Characteristics

- Manages the Core Power Supply VDDCORE by Controlling the Embedded Voltage Regulator
- Supply Monitor Detection on VDDIO or a POR (Power-On Reset) on VDDCORE Can Trigger a Core Reset
- Generates the Slow Clock SLCK, by Selecting Either the 32 kHz Low-power RC Oscillator or the 32 kHz Low-power Crystal Oscillator

16.3 Block Diagram

Figure 16-1. Supply Controller Block Diagram



Note 1: FSTT0 - FSTT15 are possible Fast Startup sources, generated by WKUP0 - WKUP15 pins, but are not physical pins.

16.4 Supply Controller Functional Description

16.4.1 Supply Controller Overview

The device can be divided into two power supply areas:

- VDDIO power supply: includes the Supply Controller, part of the Reset Controller, the slow clock switch, the General-purpose Backup Registers, the Supply Monitor and the clock which includes the Real-time Timer and the Real-time Clock
- Core power supply: includes part of the Reset Controller, the POR core, the processor, the SRAM memory, the Flash memory and the peripherals

The Supply Controller (SUPC) controls the supply voltage of the core power supply. The SUPC intervenes when the VDDIO power supply rises (when the system is starting).

The SUPC also integrates the slow clock generator which is based on a 32 kHz crystal oscillator and an embedded 32 kHz RC oscillator. The slow clock defaults to the RC oscillator, but the software can enable the crystal oscillator and select it as the slow clock source.

The SUPC and the VDDIO power supply have a reset circuitry based on a zero-power power-on reset cell. The zero-power power-on reset allows the SUPC to start properly as soon as the VDDIO voltage becomes valid.

At start-up of the system, once the backup voltage VDDIO is valid and the embedded 32 kHz RC oscillator is stabilized, the SUPC starts up the core by sequentially enabling the internal voltage regulator, waiting for the core voltage VDDCORE to be valid, then releasing the reset signal of the core “vddcore_nreset” signal.

Once the system has started, the user can program a supply monitor and/or a brownout detector. If the supply monitor detects a voltage on VDDIO that is too low, the SUPC can assert the reset signal of the core “vddcore_nreset” signal until VDDIO is valid. Likewise, if the POR core detects a core voltage VDDCORE that is too low, the SUPC can assert the reset signal “vddcore_nreset” until VDDCORE is valid.

16.4.2 Slow Clock Generator

The SUPC embeds a slow clock generator that is supplied with the VDDIO power supply. As soon as the VDDIO is supplied, both the crystal oscillator and the embedded RC oscillator are powered up, but only the embedded RC oscillator is enabled. This allows the slow clock to be valid in a short time (about 100 µs).

The user can select the crystal oscillator to be the source of the slow clock, as it provides a more accurate frequency. The command is made by writing the SUPC Control Register (SUPC_CR) with the XTALSEL bit at 1.This results in a sequence which first configures the PIO lines multiplexed with XIN32 and XOUT32 to be driven by the oscillator, then enables the crystal oscillator, then counts a number of slow RC oscillator clock periods to cover the start-up time of the crystal oscillator (refer to electrical characteristics for details of 32KHz crystal oscillator start-up time), then switches the slow clock on the output of the crystal oscillator and then disables the RC oscillator to save power. The switching time may vary according to the slow RC oscillator clock frequency range. The switch of the slow clock source is glitch free. The OSCSEL bit of the SUPC Status Register (SUPC_SR) allows knowing when the switch sequence is done.

Coming back on the RC oscillator is only possible by shutting down the VDDIO power supply.

If the user does not need the crystal oscillator, the XIN32 and XOUT32 pins should be left unconnected.

The user can also set the crystal oscillator in bypass mode instead of connecting a crystal. In this case, the user has to provide the external clock signal on XIN32. The input characteristics of the XIN32 pin are given in the product electrical characteristics section. In order to set the bypass mode, the OSCBYPASS bit of the SUPC Mode Register (SUPC_MR) needs to be set at 1.

16.4.3 Supply Monitor

The SUPC embeds a supply monitor which is located in the VDDIO Power Supply and which monitors VDDIO power supply.

The supply monitor can be used to prevent the processor from falling into an unpredictable state if the Main power supply drops below a certain level.

The threshold of the supply monitor is programmable. It can be selected from 1.62V to 2V by steps of 100 mV. This threshold is programmed in the SMTH field of the SUPC Supply Monitor Mode Register (SUPC_SMMR).

The supply monitor can also be enabled during one slow clock period on every one of **either** 32, 256 or 2048 slow clock periods, according to the choice of the user. This can be configured by programming the SMSMPL field in SUPC_SMMR.

Enabling the supply monitor for such reduced times allows to divide the typical supply monitor power consumption respectively by factors of 2, 16 and 128, if the user does not need a continuous monitoring of the VDDIO power supply.

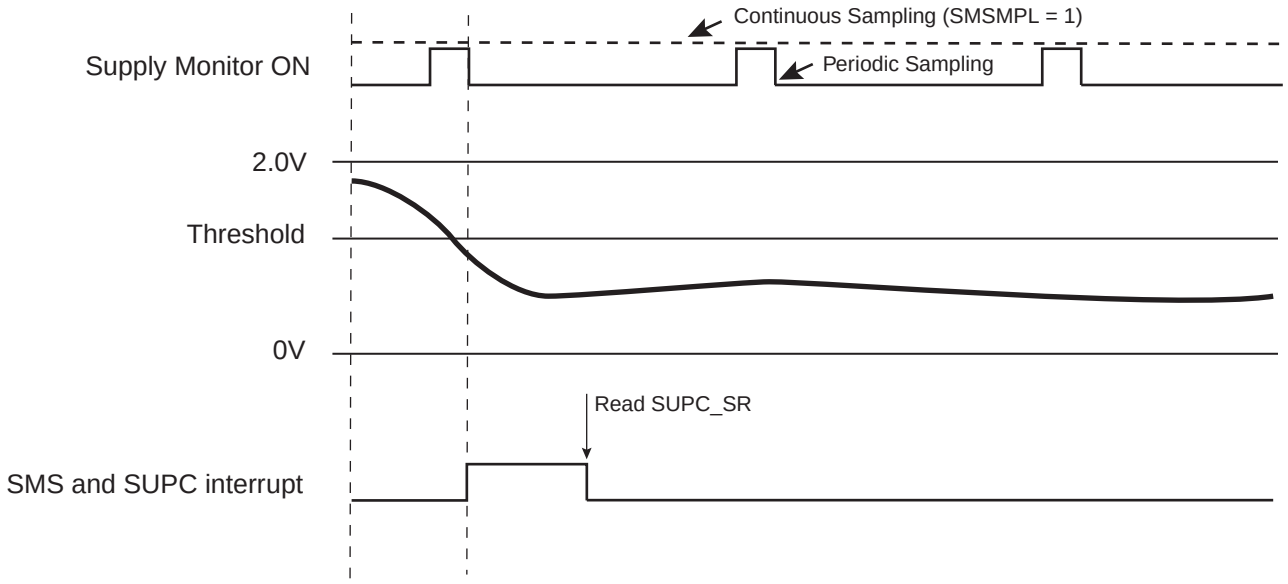
A supply monitor detection can either generate a reset of the core power supply or a wake-up of the core power supply. Generating a core reset when a supply monitor detection occurs is enabled by writing the SMRSTEN bit to 1 in SUPC_SMMR.

The SUPC provides two status bits in the SUPC_SR for the supply monitor:

- The SMOS bit provides real time information, which is updated at each measurement cycle or updated at each slow clock cycle, if the measurement is continuous.
- The SMS bit provides saved information and shows a supply monitor detection has occurred since the last read of SUPC_SR.

The SMS bit can generate an interrupt if the SMIEN bit is set to 1 in SUPC_SMMR.

Figure 16-2. Supply Monitor Status Bit and Associated Interrupt



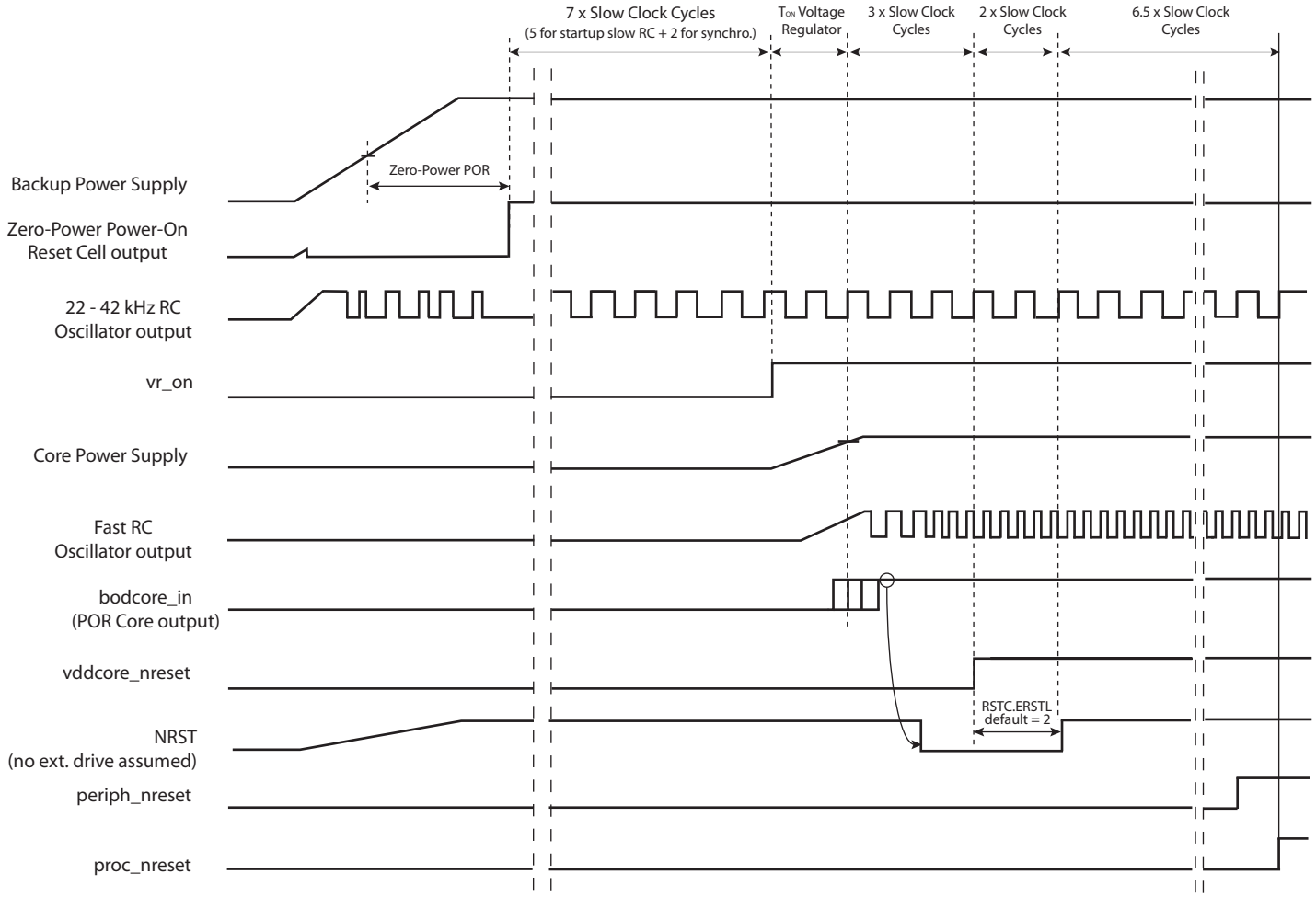
16.4.4 Power Supply Reset

16.4.4.1 Raising the Power Supply

As soon as the voltage VDDIO rises, the RC oscillator is powered up and the zero-power power-on reset cell maintains its output low as long as VDDIO has not reached its target voltage. During this time, the SUPC is entirely reset. When the VDDIO voltage becomes valid and zero-power power-on reset signal is released, a counter is started for 5 slow clock cycles. This is the time it takes for the 32 kHz RC oscillator to stabilize.

After this time, the voltage regulator is enabled. The core power supply rises and the POR core provides the bodcore_in signal as soon as the core voltage VDDCORE is valid. This results in releasing the vddcore_nreset signal to the Reset Controller after the bodcore_in signal has been confirmed as being valid for at least one slow clock cycle.

Figure 16-3. Raising the VDDIO Power Supply



Note: After "proc_nreset" rising, the core starts fetching instructions from Flash at 8 MHz.

16.4.5 Core Reset

The SUPC manages the vddcore_nreset signal to the Reset Controller, as described in Section 16.4.4 "Power Supply Reset". The vddcore_nreset signal is normally asserted before shutting down the core power supply and released as soon as the core power supply is correctly regulated.

There are two additional sources which can be programmed to activate vddcore_nreset:

- VDDIO supply monitor detection
- POR core detection

16.4.5.1 Supply Monitor Reset

The supply monitor is capable of generating a reset of the system. This can be enabled by setting the SMRSTEN bit in SUPC_SMMR.

If SMRSTEN is set and if a supply monitor detection occurs, the vddcore_nreset signal is immediately activated for a minimum of one slow clock cycle.

16.4.5.2 POR Core Reset

The POR Core provides the bodcore_in signal to the SUPC which indicates that the voltage regulation is operating as programmed. If this signal is lost for longer than 1 slow clock period while the voltage regulator is enabled, the SUPC can assert vddcore_nreset. This feature is enabled by writing the bit BODRSTEN (POR Reset Enable) to 1 in SUPC_MR.

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

While the POR core output (bodcore_in) is cleared, the vddcore_nreset signal remains active.

16.4.6 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
- SUPC Control Register
- SUPC Supply Monitor Mode Register
- SUPC Mode Register

16.5 Supply Controller (SUPC) User Interface

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

16.5.1 System Controller (SYSC) User Interface

Table 16-1. System Controller Registers

Offset	System Controller Peripheral	Name
0x00-0x0c	Reset Controller	RSTC
0x10-0x2C	Supply Controller	SUPC
0x30-0x3C	Real-Time Timer	RTT
0x50-0x5C	Watchdog 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	–

16.5.2 Supply Controller (SUPC) User Interface

Table 16-2. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Supply Controller Control Register	SUPC_CR	Write-only	N/A
0x04	Supply Controller Supply Monitor Mode Register	SUPC_SMMR	Read-write	0x0000_0000
0x08	Supply Controller Mode Register	SUPC_MR	Read-write	0x00E0_5A00
0x0C-0x10	Reserved	–	–	–
0x14	Supply Controller Status Register	SUPC_SR	Read-only	0x0000_0000
0x18-0x1C	Reserved	–	–	–

16.5.3 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	ZERO	–	–

- **ZERO: Zero**

This bit must always be written to 0.

- **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. Always reads as 0.

16.5.4 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			

- **SMTH: Supply Monitor Threshold**

Allows to select the threshold voltage of the supply monitor. Refer to the section Electrical Characteristics for voltage values.

- **SMSMPL: Supply Monitor Sampling Period**

Value	Name	Description
0x0	SMD	Supply Monitor disabled
0x1	CSM	Continuous Supply Monitor
0x2	32SLCK	Supply Monitor enables one SLCK period every 32 SLCK periods
0x3	256SLCK	Supply Monitor enables one SLCK period every 256 SLCK periods
0x4	2048SLCK	Supply Monitor enables one SLCK period every 2,048 SLCK periods

- **SMRSTEN: Supply Monitor Reset Enable**

0 (NOT_ENABLE): The core reset signal “vddcore_nreset” is not affected when a supply monitor detection occurs.

1 (ENABLE): The core reset signal, vddcore_nreset is asserted when a supply monitor detection occurs.

- **SMIEN: Supply Monitor Interrupt Enable**

0 (NOT_ENABLE): The SUPC interrupt signal is not affected when a supply monitor detection occurs.

1 (ENABLE): The SUPC interrupt signal is asserted when a supply monitor detection occurs.

16.5.5 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
–	–	BODDIS	BODRSTEN	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

- **BODRSTEN: POR Core Reset Enable**

0 (NOT_ENABLE): The core reset signal “vddcore_nreset” is not affected when a brownout detection occurs.

1 (ENABLE): The core reset signal, vddcore_nreset is asserted when a brownout detection occurs.

- **BODDIS: POR Core Disable**

0 (ENABLE): The core brownout detector is enabled.

1 (DISABLE): The core brownout detector is disabled.

- **OSCBYPASS: Oscillator Bypass**

0 (NO_EFFECT): No effect. Clock selection depends on XTALSEL value.

1 (BYPASS): The 32 kHz crystal oscillator is selected and put in bypass mode.

- **KEY: Password Key**

Value	Name	Description
0xA5	PASSWD	Writing any other value in this field aborts the write operation. Always reads as 0.

16.5.6 Supply Controller Status Register

Name: SUPC_SR

Address: 0x400E1424

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
OSCSEL	SMOS	SMS	SMRSTS	BODRSTS	–	–	–

Note: Because of the asynchronism between the slow clock (SCLK) and the system clock (MCK), the status register flag reset is taken into account only two slow clock cycles after the read of the SUPC_SR.

- **BODRSTS: Brownout Detector Reset Status**

0 (NO): No core brownout rising edge event has been detected since the last read of the SUPC_SR.

1 (PRESENT): At least one brownout output rising edge event has been detected since the last read of the SUPC_SR.

When the voltage remains below the defined threshold, there is no rising edge event at the output of the brownout detection cell. The rising edge event occurs only when there is a voltage transition below the threshold.

- **SMRSTS: Supply Monitor Reset Status**

0 (NO): No supply monitor detection has generated a core reset since the last read of the SUPC_SR.

1 (PRESENT): At least one supply monitor detection has generated a core reset since the last read of the SUPC_SR.

- **SMS: Supply Monitor Status**

0 (NO): No supply monitor detection since the last read of SUPC_SR.

1 (PRESENT): At least one supply monitor detection since the last read of SUPC_SR.

- **SMOS: Supply Monitor Output Status**

0 (HIGH): The supply monitor detected VDDIO higher than its threshold at its last measurement.

1 (LOW): The supply monitor detected VDDIO lower than its threshold at its last measurement.

- **OSCSEL: 32-kHz Oscillator Selection Status**

0 (RC): The slow clock, SLCK is generated by the embedded 32 kHz RC oscillator.

1 (CRYST): The slow clock, SLCK is generated by the 32 kHz crystal oscillator.

16.5.7 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

- **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 16.4.6 ”Register Write Protection” for the list of registers that can be write-protected.

- **WPKEY: Write Protection Key**

Value	Name	Description
0x525443	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

17. General-Purpose Backup Registers (GPBR)

17.1 Description

The System Controller embeds 8 General-purpose Backup registers.

It is possible to generate an immediate clear of the content of General-purpose Backup registers 0 to 3 (first half) if a Low-power Debounce event is detected on one of the wakeup pins, WKUP0 or WKUP1. The content of the other General-purpose Backup registers (second half) remains unchanged.

The Supply Controller module must be programmed accordingly. In the register SUPC_WUMR in the Supply Controller module, LPDBCCLR, LPDBCEN0 and/or LPDBCEN1 bit must be configured to 1 and LPDBC must be other than 0.

If a Tamper event has been detected, it is not possible to write to the General-purpose Backup registers while the LPDBCS0 or LPDBCS1 flags are not cleared in the Supply Controller Status register SUPC_SR.

17.2 Embedded Characteristics

- 8 32-bit General Purpose Backup Registers

17.3 General Purpose Backup Registers (GPBR) User Interface

Table 17-1. Register Mapping

Offset	Register	Name	Access	Reset
0x0	General Purpose Backup Register 0	SYS_GPBR0	Read-write	–
...	...	...	...	...
0xAC	General Purpose Backup Register 7	SYS_GPBR7	Read-write	–

17.3.1 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							

- **GPBR_VALUE: Value of GPBR x**

If a Tamper event has been detected, it is not possible to write GPBR_VALUE as long as the LPDBCS0 or LPDBCS1 flags have not been cleared in Supply Controller Status register SUPC_SR.

18. Memory to Memory (MEM2MEM)

18.1 Description

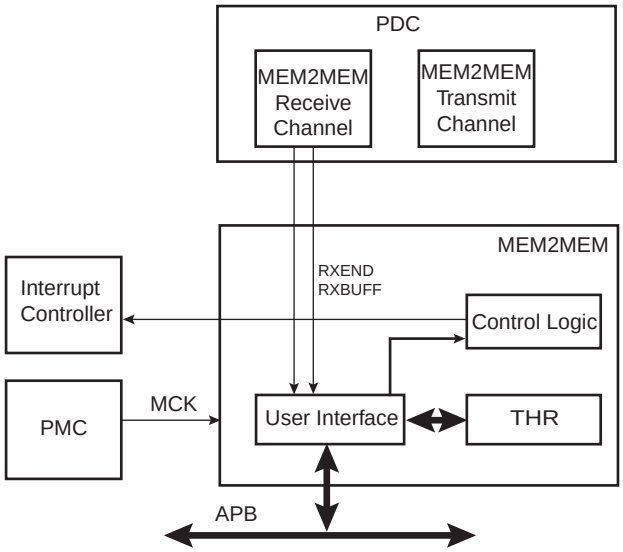
The Memory to Memory (MEM2MEM) module allows the PDC to perform memory to memory transfer without CPU intervention. The transfer size can be configured in byte, half-word or word. Two PDC channels are required to perform the transfer, one channel defines the source of the transfer while the other defines the destination.

18.2 Embedded Characteristics

- Allows PDC to perform memory to memory transfer
- Supports byte, half-word and word transfer
- Interrupt for End of Transfer

18.3 Block Diagram

Figure 18-1. Memory to Memory Block Diagram



18.4 Product Dependencies

18.4.1 Power Management

The MEM2MEM is not continuously clocked. So the programmer must first configure the PMC to enable the MEM2MEM clock through the Power Management Controller (PMC).

18.4.2 Interrupt

The MEM2MEM interface has an interrupt line connected to the Interrupt Controller.

Handling the MEM2MEM interrupt requires programming the Interrupt Controller before configuring the MEM2MEM.

Table 18-1. Peripheral IDs

Instance	ID
MEM2MEM	15

18.5 Functional Description

The memory to memory transfer requires 2 operations.

The PDC receive channel associated to the MEM2MEM module must be configured with the transfer destination address and buffer size.

The PDC transmit channel associated to the MEM2MEM module must be configured with the source address and buffer size. The transmit channel buffer size must be equal to the receive channel buffer size.

The 2 PDC channels exchange data through the MEM2MEM_THR register which appears fully transparent from configuration. This register can be used as a general purpose register in case the memory to memory transfer capability is not used.

The size of each element of the data buffer can be configured in byte, half-word or word by writing TSIZE field in MEM2MEM_MR register. Word transfer (32-bit) is the default size.

The transfer ends when either RXEND rises and/or RXBUFF rises in the MEM2MEM_ISR register.

An interrupt can be triggered at the end of transfer by programming MEM2MEM_IER register. Refer to PDC section for detailed information.

18.6 Memory to Memory (MEM2MEM) User Interface

Table 18-2. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Memory to Memory Transfer Holding Register	MEM2MEM_THR	Read-write	0x0000_0000
0x04	Memory to Memory Mode Register	MEM2MEM_MR	Read-write	0x0000_0002
0x08	Memory to Memory Interrupt Enable Register	MEM2MEM_IER	Write-only	–
0x0C	Memory to Memory Interrupt Disable Register	MEM2MEM_IDR	Write-only	–
0x10	Memory to Memory Interrupt Mask Register	MEM2MEM_IMR	Read-only	0x0000_0000
0x14	Memory to Memory Interrupt Status Register	MEM2MEM_ISR	Read-only	–
0x100-0x124	Reserved for PDC Registers	–	–	–

18.6.1 Memory to Memory Transfer Holding Register

Register Name: MEM2MEM_THR

Address: 0x40028000

Access Type: Read-write

31	30	29	28	27	26	25	24
THDATA							

23	22	21	20	19	18	17	16
THDATA							

15	14	13	12	11	10	9	8
THDATA							

7	6	5	4	3	2	1	0
THDATA							

- **THDATA: Transfer Holding Data**

Must be written by the PDC transmit channel and read by the PDC receive channel.

18.6.2 Memory to Memory Mode Register

Register Name: MEM2MEM_MR

Address: 0x40028004

Access Type: 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
–	–	–	–	–	–	TSIZE	

- **TSIZE: Transfer Size**

Value	Name	Description
0	T_8BIT	The buffer size is defined in byte.
1	T_16BIT	The buffer size is defined in half-word (16-bit).
2	T_32BIT	The buffer size is defined in word (32-bit). Default value.

18.6.3 Memory to Memory Interrupt Enable Register

Register Name: MEM2MEM_IER

Address: 0x40028008

Access Type: Write-only

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

- **RXEND: End of Transfer Interrupt Enable**
- **RXBUFF: Buffer Full Interrupt Enable**

0: No effect

1: Enables the corresponding interrupt.

18.6.4 Memory to Memory Interrupt Disable Register

Register Name: MEM2MEM_IDR

Address: 0x4002800C

Access Type: Write-only

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

- **RXEND: End of Transfer Interrupt Disable**
- **RXBUFF: Buffer Full Interrupt Disable**

0: No effect

1: Disables the corresponding interrupt.

18.6.5 Memory to Memory Interrupt Mask Register

Register Name: MEM2MEM_IMR

Address: 0x40028010

Access Type: Read-only

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

- **RXEND: End of Transfer Interrupt Mask**
- **RXBUFF: Buffer Full Interrupt Mask**

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

18.6.6 Memory to Memory Interrupt Status Register

Register Name: MEM2MEM_ISR

Address: 0x40028014

Access Type: Read-only

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

- **RXEND: End of Transfer**

0: The End of Transfer signal from the Receive PDC channel is inactive.

1: The End of Transfer signal from the Receive PDC channel is active.

- **RXBUFF: Buffer Full**

0: The signal Buffer Full from the Receive PDC channel is inactive.

1: The signal Buffer Full from the Receive PDC channel is active.

19. Enhanced Embedded Flash Controller (EEFC)

19.1 Description

The Enhanced Embedded Flash Controller (EEFC) ensures the interface of the Flash block with the 32-bit internal bus.

Its 128-bit or 64-bit wide memory interface increases performance. It also manages the programming, erasing, locking and unlocking sequences of the Flash using a full set of commands. One of the commands returns the embedded Flash descriptor definition that informs the system about the Flash organization, thus making the software generic.

19.2 Embedded Characteristics

- Interface of the Flash Block with the 32-bit Internal Bus
- Increases Performance in Thumb-2 Mode with 128-bit or 64-bit-wide Memory Interface up to 48 MHz
- Code Loop Optimization
- 32 Lock Bits, Each Protecting a Lock Region
- 8 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 Pages
- Possibility of Erasing before Programming
- Locking and Unlocking Operations
- Possibility to Read the Calibration Bits

19.3 Product Dependencies

19.3.1 Power Management

The Enhanced Embedded Flash Controller (EEFC) is continuously clocked. The Power Management Controller has no effect on its behavior.

19.3.2 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 on FRDY bit rising.

Table 19-1. Peripheral IDs

Instance	ID
EFC	6

19.4 Functional Description

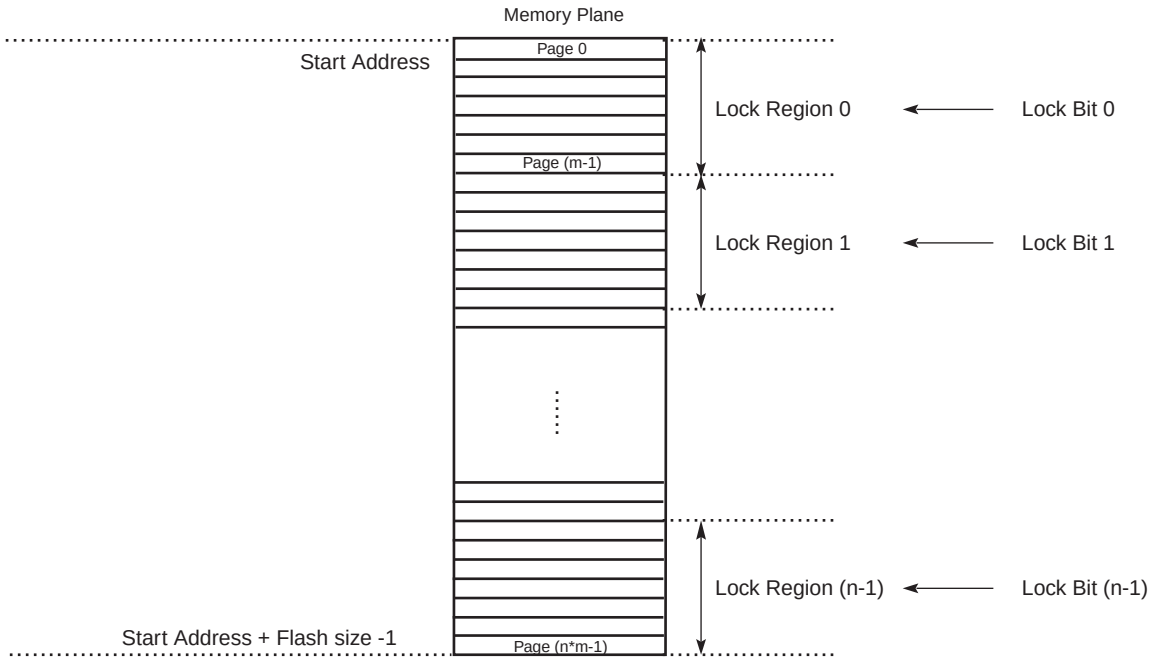
19.4.1 Embedded Flash Organization

The embedded Flash interfaces directly with the 32-bit internal bus. The embedded Flash is composed of:

- One memory plane organized in several pages of the same size
- Two 128-bit or 64-bit read buffers used for code read optimization
- One 128-bit or 64-bit read buffer used for data read optimization
- One write buffer that manages page programming. The write buffer size is equal to the page size. This buffer is write-only and accessible all along the 1 MByte address space, so that each word can be written to its final address.
- Several lock bits used to protect write/erase operation on several pages (lock region). A lock bit is associated with a lock region composed of several pages in the memory plane.
- Several bits that may be set and cleared through the 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 controlled after a Get descriptor command issued by the application (see “Get Flash Descriptor Command” on page 379).

Figure 19-1. Embedded Flash Organization



19.4.2 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. Refer to the Electrical Characteristics section for more details.

19.4.2.1 128-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 FAM bit in EEFC_FMR.

Refer to the Electrical Characteristics section for more details.

19.4.2.2 Code Read Optimization

Code read optimization is enabled if the SCOD bit in EEFC_FMR is cleared.

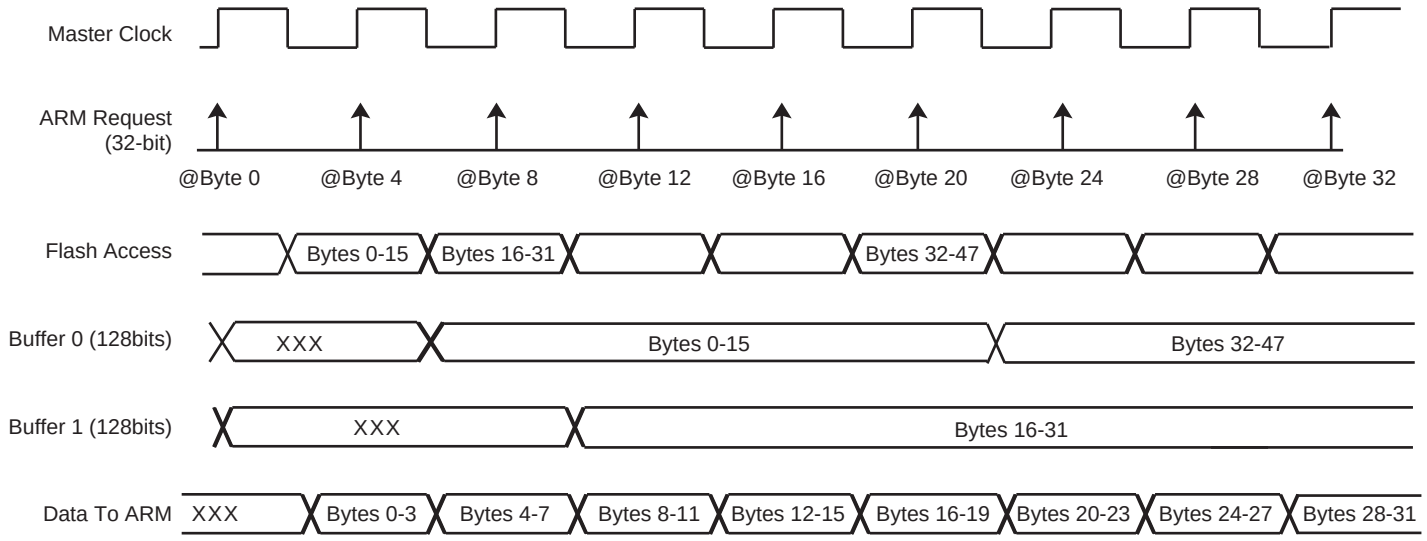
A system of 2 x 128-bit or 2 x 64-bit buffers is added in order to optimize sequential code fetch.

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 SCOD bit in EEFC_FMR is set to 1, these buffers are disabled and the sequential code read is no longer optimized.

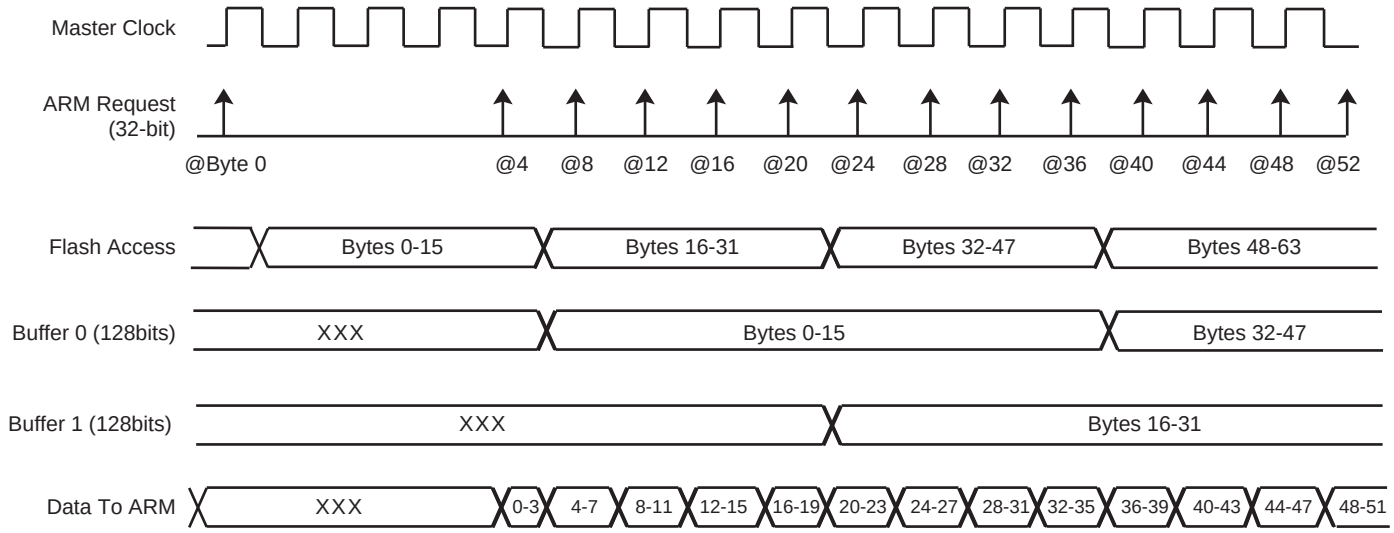
Another system of 2 x 128-bit or 2 x 64-bit buffers is added in order to optimize loop code fetch. Refer to “Code Loop Optimization” on page 376 for more details.

Figure 19-2. Code Read Optimization for FWS = 0



Note: When FWS is equal to 0, all the accesses are performed in a single-cycle access.

Figure 19-3. Code Read Optimization for FWS = 3



Note: When FWS is included between 1 and 3, in case of sequential reads, the first access takes (FWS+1) cycles, the other ones only 1 cycle.

19.4.2.3 Code Loop Optimization

Code loop optimization is enabled when the CLOE bit of EEFC_FMR is set to 1.

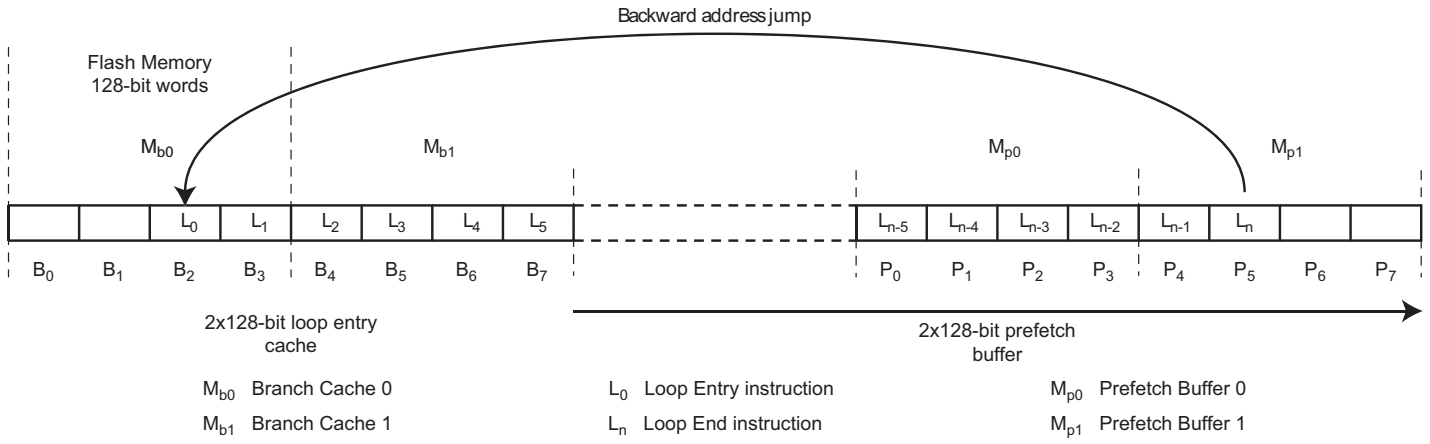
When a backward jump is inserted in the code, the pipeline of the sequential optimization is broken and becomes inefficient. In this case, the loop code read optimization takes over from the sequential code read optimization to prevent the insertion of wait states. The loop code read optimization is enabled by default. In EEFC_FMR, if the bit CLOE is reset to 0 or the bit SCOD is set to 1, 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 19.4.2.2 ”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 19-4 illustrates code loop optimization.

Figure 19-4. Code Loop Optimization

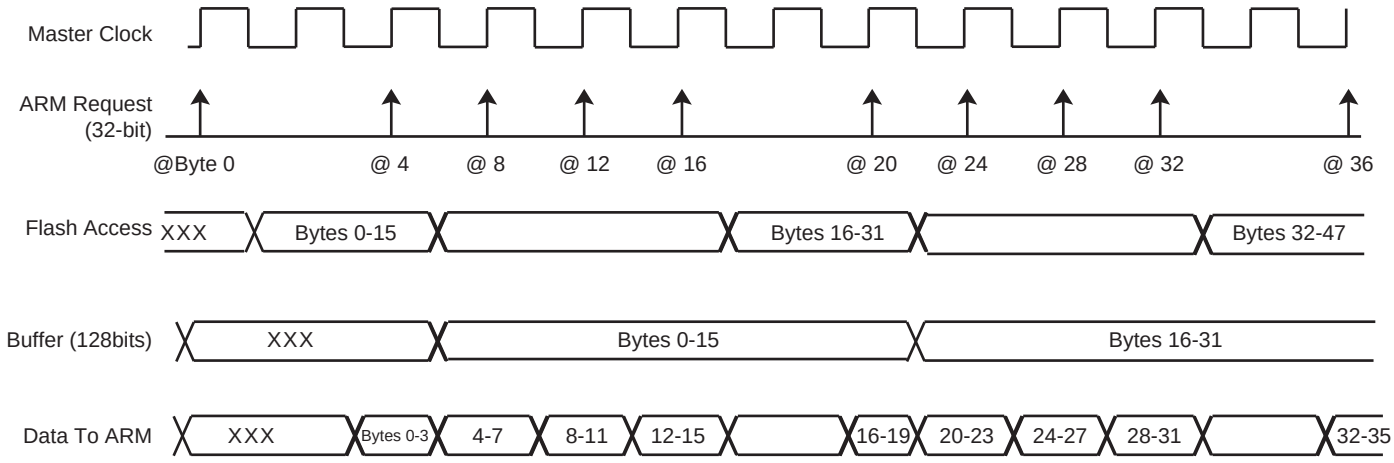


19.4.2.4 Data Read Optimization

The organization of the Flash in 128 bits (or 64 bits) is associated with two 128-bit (or 64-bit) prefetch buffers and one 128-bit (or 64-bit) data read buffer, thus providing maximum system performance. This buffer is added in order to store the requested data plus all the data contained in the 128-bit (64-bit) aligned data. This speeds up sequential data reads if, for example, FWS is equal to 1 (see Figure 19-5). The data read optimization is enabled by default. If the SCOD bit in EEFC_FMR is set to 1, 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 19-5. Data Read Optimization for FWS = 1



19.4.3 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.

Table 19-2. 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

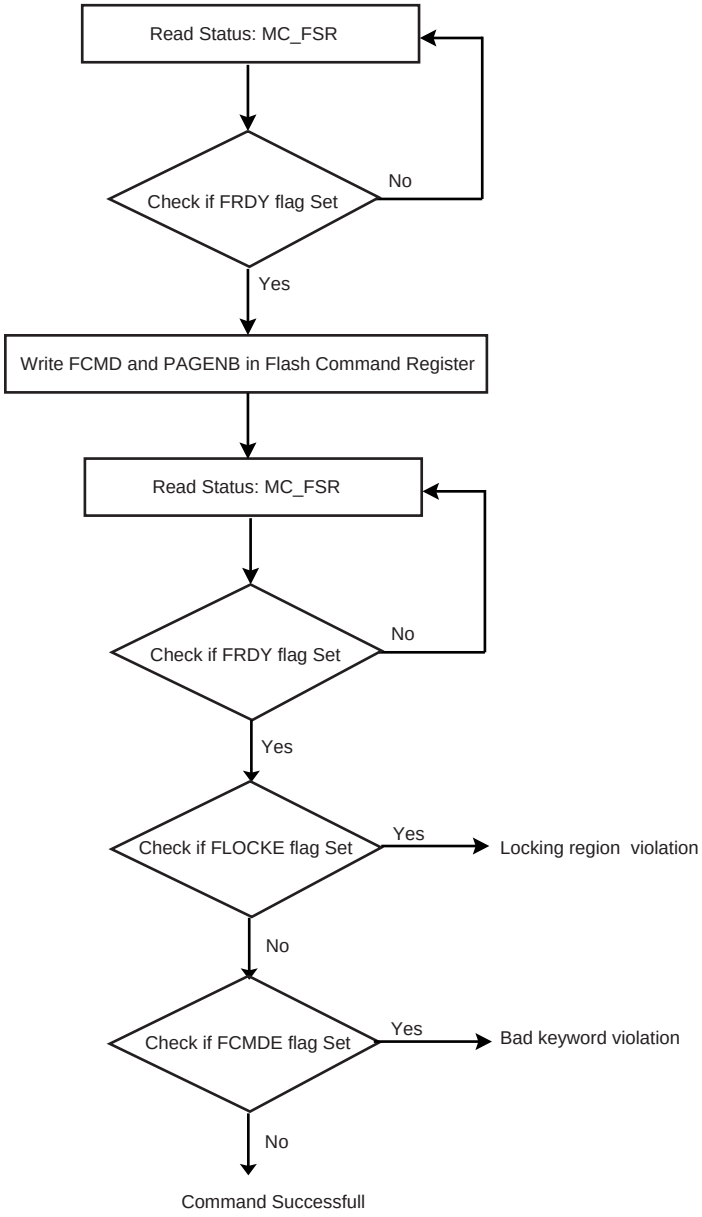
In order to perform one of these commands, the Flash Command register (EEFC_FCR) must be written with the correct command using the FCMD field. As soon as EEFC_FCR is written, the FRDY flag and the FVALUE field in the Flash Result register (EEFC_FRR) are automatically cleared. Once the current command is achieved, then the FRDY flag is automatically set. If an interrupt has been enabled by setting the FRDY bit in EEFC_FMR, the corresponding interrupt line of the 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 is achieved.)

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 19-6. Command State Chart



19.4.3.1 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, then EEFC_FRR value is 0 until the next valid command.

Table 19-3. 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.

19.4.3.2 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 as follows:

- Write the data to be programmed in the latch buffer.
- Write the programming command in EEFC_FCR. This will automatically clear the FRDY bit in EEFC_TSR.
- When Flash programming is completed, the FRDY bit in EEFC_FSR rises. If an interrupt has been enabled by setting the bit FRDY in EEFC_FMR, 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 the Programming Page command is given, 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 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 launching the WP command. The latch buffer must be written in ascending order, starting from the first address of the page. See Figure 19-7, "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 19-8, "Partial Page Programming"

Optimized Partial Page Programming

The EEFC automatically detects the number of 128-bit words to be programmed. If only one 128-bit aligned word is to be programmed in the Flash array, the process is optimized to reduce the time needed for programming.

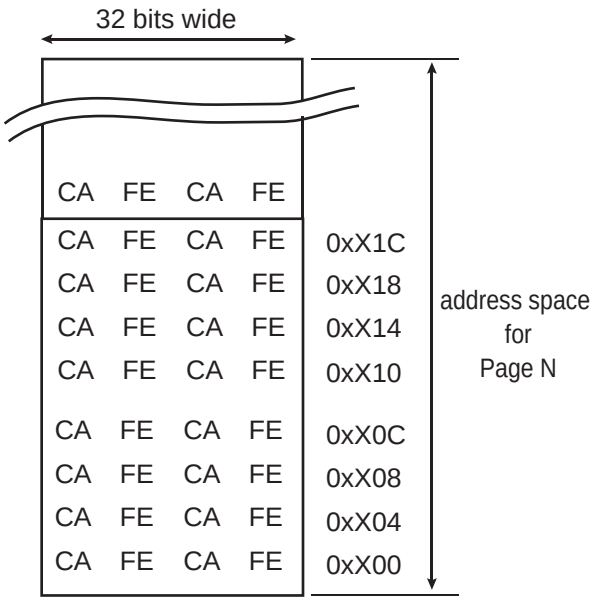
If several 128-bit words are to be programmed, a standard page programming operation is performed.

See Figure 19-9, "Optimized Partial Page Programming".

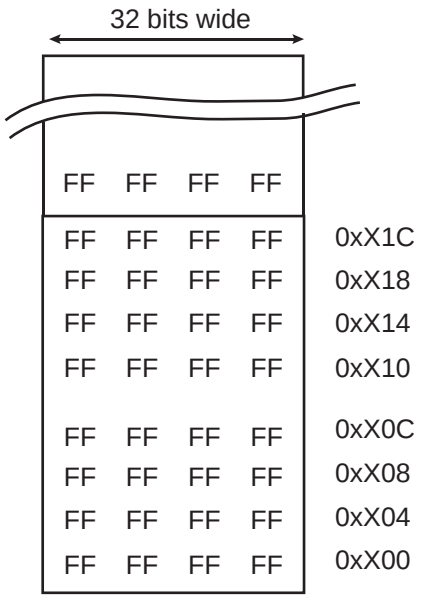
Programming Bytes

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, as shown in Figure 19-10, "Programming Bytes in the Flash".

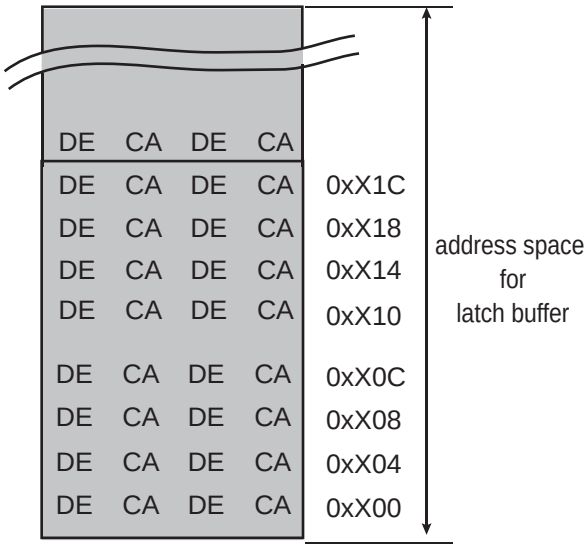
Figure 19-7. Full Page Programming



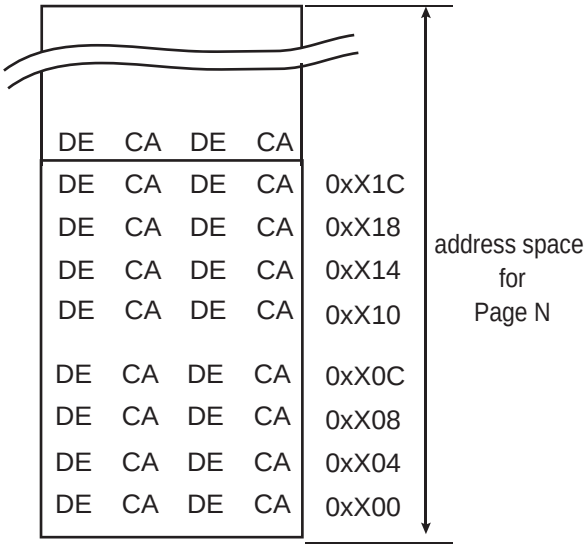
Before programming: Unerased page in Flash array



Step 1: Flash array after page erase

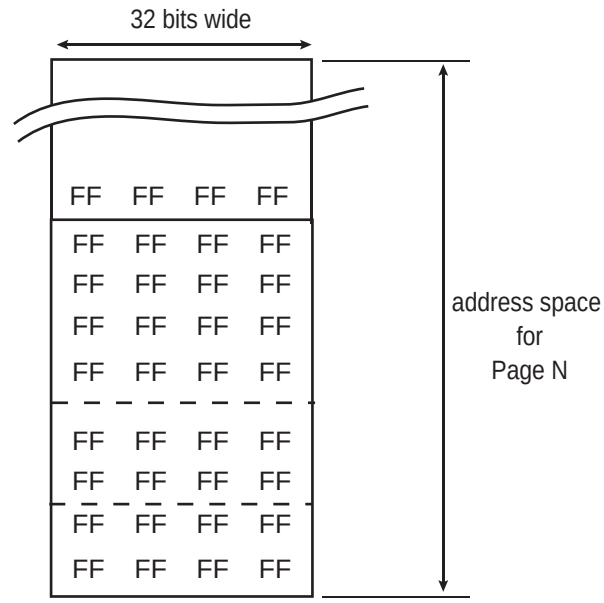


Step 2: Writing a page in the latch buffer

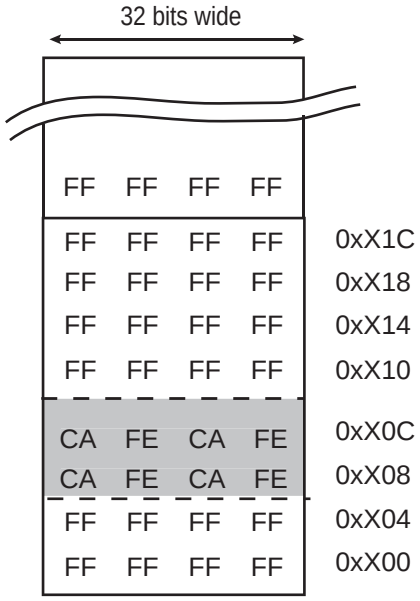


Step 3: Page in Flash array after issuing WP command and FRDY=1

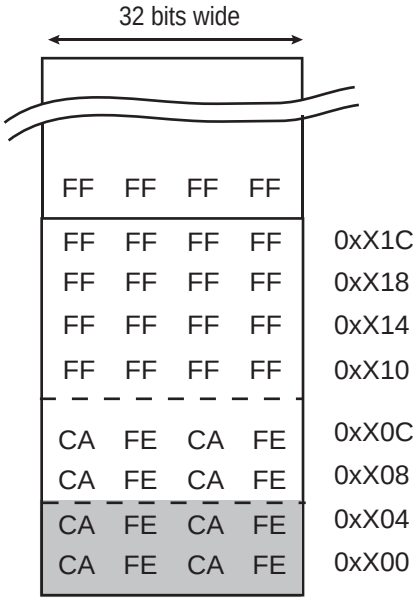
Figure 19-8. Partial Page Programming



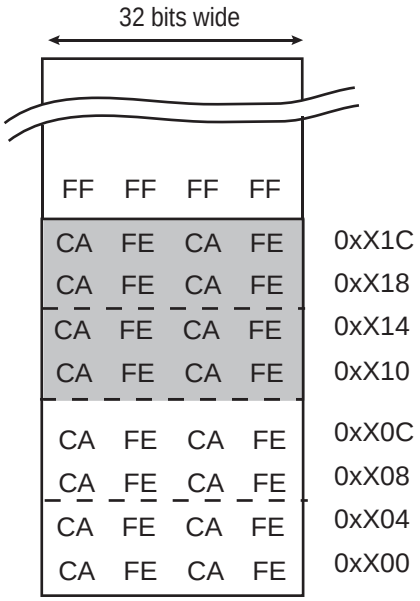
Step 1: Flash array after page erase



Step 2: Flash array after programming 64-bit at address 0xX08 (write latch buffer + WP)

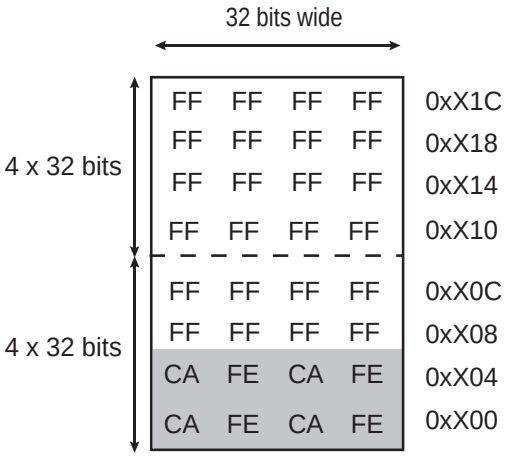


Step 3: Flash array after programming a second 64-bit data at address 0xX00 (write latch buffer + WP)

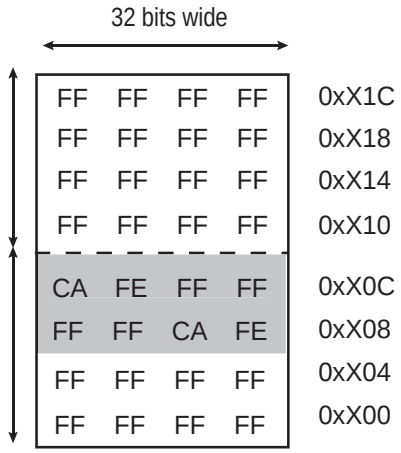


Step 4: Flash array after programming a 128-bit data word at address 0xX10 (write latch buffer + WP)

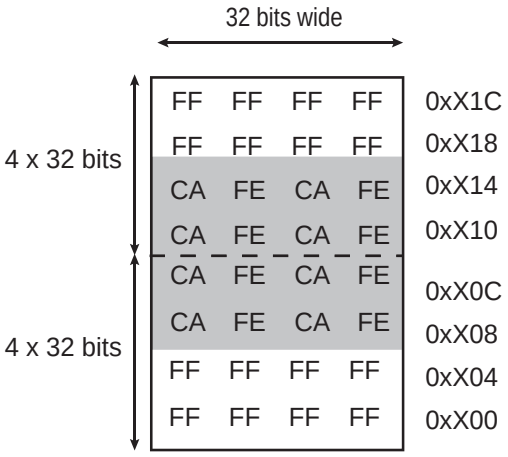
Figure 19-9. Optimized Partial Page Programming



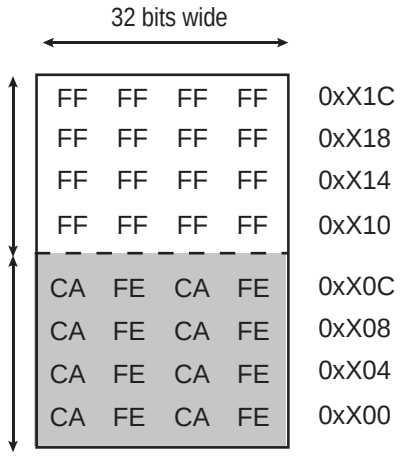
Case 1: 2 x 32 bits modified, not crossing 128-bit boundary
User programs WP, Flash Controller sends Write Word
=> Only 1 word programmed => programming period reduced



Case 2: 2 x 32 bits modified, not crossing 128-bit boundary
User programs WP, Flash Controller sends Write Word
=> Only 1 word programmed => programming period reduced

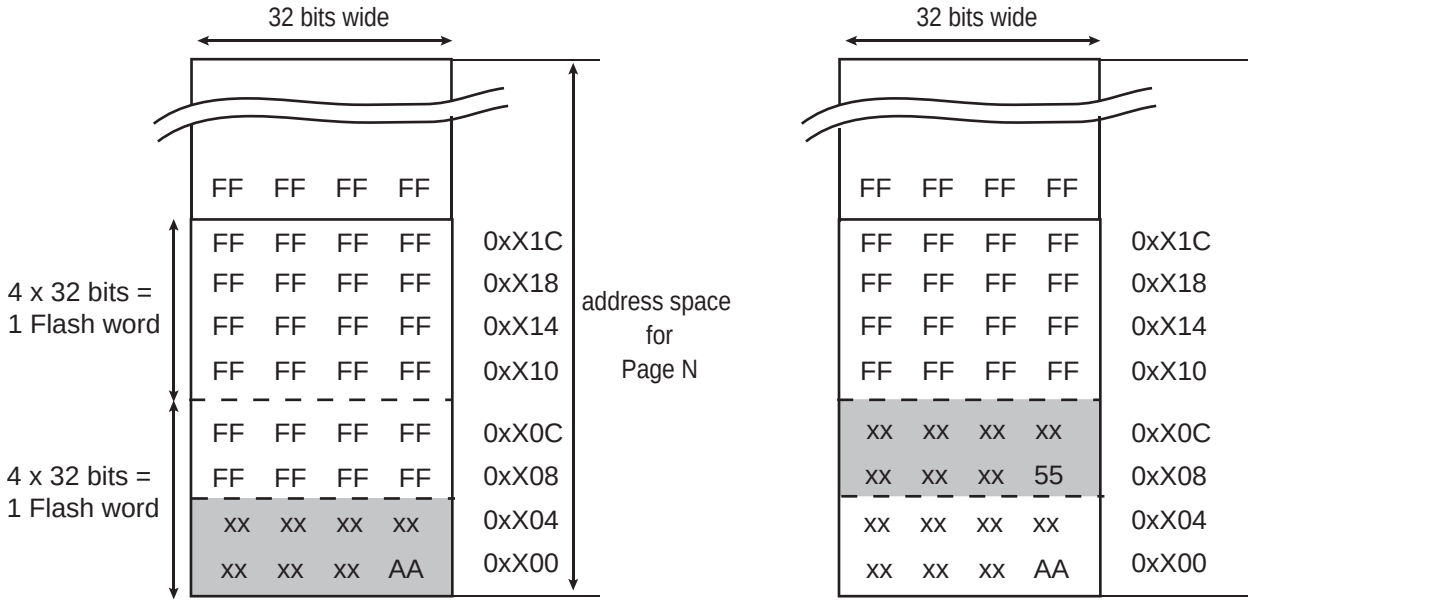


Case 3: 4 x 32 bits modified across 128-bit boundary
User programs WP, Flash Controller sends WP
=> Whole page programmed



Case 4: 4 x 32 bits modified, not crossing 128-bit boundary
User programs WP, Flash Controller sends Write Word
=> Only 1 word programmed => programming period reduced

Figure 19-10. 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.

19.4.3.3 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 MC_FCR. The first page number must be modulo 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. FARG must be set with a page number that is in the sector to be erased.

Note: If one subsector is locked within the first sector, the erase sector (ES) command cannot be processed on non-locked subsectors of the first sector. All the lock bits of the first sector must be cleared prior to issuing an ES command on the first sector. After the ES command has been issued, the first sector lock bits must be reverted to the state before clearing them.

If the processor is fetching code from the Flash memory while the EPA or ES command is being performed, the processor accesses will be 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:

- 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 19-4. 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)

- When programming is completed, the FRDY bit in EEFC_FSR rises. If an interrupt has been enabled by setting the FRDY bit in EEFC_FMR, the interrupt line of the interrupt controller 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: at least one page to be erased belongs to a locked region. The erase command has been refused, no page has been erased. A command must be run previously to unlock the corresponding region.
- Flash Error: at the end of the programming, the EraseVerify test of the Flash memory has failed.

19.4.3.4 Lock Bit Protection

Lock bits are associated with several pages in the embedded Flash memory plane. This defines lock regions in the embedded Flash memory plane. They prevent writing/erasing protected pages.

The lock sequence is:

- The Set lock bit command (SLB) and a page number to be protected are written in EEFC_FCR.
- When the locking completes, the FRDY bit in EEFC_FSR rises. If an interrupt has been enabled by setting the FRDY bit in EEFC_FMR, the interrupt line of the interrupt controller is activated.
- 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. Then the locked region can be erased or programmed. The unlock sequence is:

- The Clear lock bit command (CLB) and a page number to be unprotected are written in EEFC_FCR.
- When the unlock completes, the FRDY bit in EEFC_FSR rises. If an interrupt has been enabled by setting the FRDY bit in EEFC_FMR, 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 status sequence is:

- The Get lock bit command (GLB) is written in EEFC_FCR, FARG field is meaningless.
- 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, then 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, clear or get lock bit command is performed.

19.4.3.5 GPNVM Bit

GPNVM bits do not interfere with the embedded Flash memory plane. Refer to specific product details for information on GPNVM bit action.

The Set GPNVM bit sequence is:

- Start the Set GPNVM bit command (SGPB) by writing EEFC_FCR with the SGPB command and the number of the GPNVM bits to be set.
- When the GPNVM bit is set, the bit FRDY in EEFC_FSR rises. If an interrupt was enabled by setting the FRDY bit in EEFC_FMR, the interrupt line of the interrupt controller is activated.
- 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:

- Start the Clear GPNVM bit command (CGPB) by writing EEFC_FCR with CGPB and the number of the GPNVM bits to be cleared.
- When the clear completes, the FRDY bit in EEFC_FSR rises. If an interrupt has been enabled by setting the FRDY bit in EEFC_FMR, 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:

- Start the Get GPNVM bit command by writing EEFC_FCR with GGPB. The FARG field is meaningless.
- 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, then 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, clear or get GPNVM bit command is performed.

19.4.3.6 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:

- Issue the Get CALIB bit command by writing EEFC_FCR with GCALB (see Table 19-2). The FARG field is meaningless.
- 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 8/16/24 MHz fast RC oscillator is calibrated in production. This calibration can be read through the Get CALIB bit command. The table below shows the bit implementation for each frequency:

Table 19-5. Calibration Bit Indexes

RC Calibration Frequency	EEFC_FRR Bits
16 MHz output	[28 - 22]
24 MHz output	[38 - 32]

The RC calibration for the 8 MHz is set to ‘1000000’.

19.4.3.7 Security Bit Protection

When the security is enabled, access to the Flash, either through the JTAG/SWD interface or through the Fast Flash Programming interface, is forbidden. This ensures the confidentiality of the code programmed in the Flash.

The security bit is GPNVM0.

Disabling the security bit can only be achieved by asserting the ERASE pin at 1, and after a full Flash erase is performed. When the security bit is deactivated, all accesses to the Flash are permitted.

19.4.3.8 Unique Identifier

Each part is programmed with a 2 × 512-byte unique identifier. It can be used to generate keys for example. To read the unique identifier, the sequence is:

- Send the Start read unique identifier command (STUI) by writing EEFC_FCR with the STUI command.
- When the unique identifier is ready to be read, the FRDY bit in EEFC_FSR falls.
- The unique identifier is located at the address 0x00400000-0x004003FF, in the first 128 bits of the Flash memory mapping.
- To stop the unique identifier mode, the user needs to send the Stop read unique identifier command (SPUI) by writing EEFC_FCR with the SPUI command.
- When the SPUI command has been performed, the FRDY bit in EEFC_FSR rises. If an interrupt was enabled by setting the FRDY bit in EEFC_FMR, the interrupt line of the interrupt controller is activated.

Note that during the sequence, the software cannot run out of Flash.

19.4.3.9 User Signature

Each part contains a user signature of 512 bytes. It can be used for storage. Read, write and erase of this area is allowed.

To read the user signature, the sequence is as follows:

- Send the Start read user signature command (STUS) by writing EEFC_FCR with the STUS command.
- When the user signature is ready to be read, the FRDY bit in EEFC_FSR falls.
- The user signature is located in the first 512 bytes of the Flash memory mapping, thus, at the address 0x00400000-0x004001FF.

- To stop the user signature mode, the user needs to send the Stop read user signature command (SPUS) by writing EEFC_FCR with the SPUS command.
- When the SPUI command has been performed, the FRDY bit in EEFC_FSR rises. If an interrupt was enabled by setting the FRDY bit in EEFC_FMR, the interrupt line of the interrupt controller is activated.

Note that during the sequence, the software cannot run out of Flash or the second plane in case of dual plane.

One error can be detected in EEFC_FSR after this sequence:

- Command Error: a bad keyword has been written in EEFC_FCR.

To write the user signature, the sequence is:

- Write the full page, at any page address, within the internal memory area address space.
- Send the Write user signature command (WUS) by writing EEFC_FCR with the WUS command.
- When programming is completed, the FRDY bit in EEFC_FSR rises. If an interrupt has been enabled by setting the FRDY bit in EEFC_FMR, 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.

To erase the user signature, the sequence is:

- Send the Erase user signature command (EUS) by writing EEFC_FCR with the EUS command.
- When programming is completed, the FRDY bit in EEFC_FSR rises. If an interrupt has been enabled by setting the FRDY bit in EEFC_FMR, 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.

19.5 Enhanced Embedded Flash Controller (EEFC) User Interface

The User Interface of the Embedded Flash Controller (EEFC) is integrated within the System Controller with base address 0x400E0800.

Table 19-6. Register Mapping

Offset	Register	Name	Access	Reset State
0x00	EEFC Flash Mode Register	EEFC_FMR	Read/Write	0x0400_0000
0x04	EEFC Flash Command Register	EEFC_FCR	Write-only	–
0x08	EEFC Flash Status Register	EEFC_FSR	Read-only	0x00000001
0x0C	EEFC Flash Result Register	EEFC_FRR	Read-only	0x0
0x10	Reserved	–	–	–

19.5.1 EEFC Flash Mode Register

Name: EEFC_FMR

Address: 0x400E0A00

Access: Read/Write

Offset: 0x00

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

- **FRDY: Ready Interrupt Enable**

0: Flash ready does not generate an interrupt.

1: Flash ready (to accept a new command) generates an interrupt.

- **FWS: Flash Wait State**

This field defines the number of wait states for read and write operations:

Number of cycles for Read/Write operations = FWS+1

- **SCOD: Sequential Code Optimization Disable**

0: The sequential code optimization is enabled.

1: The sequential code optimization is disabled.

No Flash read should be done during change of this register.

- **FAM: Flash Access Mode**

0: 128-bit access in read mode only, to enhance access speed.

1: 64-bit access in read mode only, to enhance power consumption.

No Flash read should be done during change of this register.

- **CLOE: Code 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 register.

19.5.2 EEFC Flash Command Register

Name: EEFC_FCR

Address: 0x400E0A04

Access: Write-only

Offset: 0x04

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

- **FCMD: Flash Command**

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 Modulo 4 FARG[1:0] = 1: Eight pages to be erased. FARG[15:2] = Page_Number Modulo 8 FARG[1:0] = 2: Sixteen pages to be erased. FARG[15:2] = Page_Number Modulo 16 FARG[1:0] = 3: Thirty-two pages to be erased. FARG[15:2] = Page_Number Modulo 32 Refer to Table 19-4 on page 386.
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.

19.5.3 EEFC Flash Status Register

Name: EEFC_FSR

Address: 0x400E0A08

Access: Read-only

Offset: 0x08

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

- **FRDY: Flash Ready Status**

0: The 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**

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.

This flag is automatically cleared when EEFC_FSR is read or EEFC_FCR is written.

- **FLOCKE: Flash Lock Error Status**

0: No programming/erase of at least one locked region has happened since the last read of EEFC_FSR.

1: Programming/erase of at least one locked region has happened since the last read of EEFC_FSR.

This flag is automatically cleared when EEFC_FSR is read or EEFC_FCR is written.

- **FLERR: Flash Error Status**

0: No Flash memory error occurred at the end of programming (EraseVerify or WriteVerify test has passed).

1: A Flash memory error occurred at the end of programming (EraseVerify or WriteVerify test has failed).

19.5.4 EEFC Flash Result Register

Name: EEFC_FRR

Address: 0x400E0A0C

Access: Read-only

Offset: 0x0C

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

- **FVALUE: Flash Result Value**

The result of a Flash command is returned in this register. If the size of the result is greater than 32 bits, then the next resulting value is accessible at the next register read.

20. Bus Matrix (MATRIX)

20.1 Description

The Bus Matrix implements a multi-layer AHB that enables parallel access paths between multiple AHB masters and slaves in a system, thus increasing overall bandwidth. The Bus Matrix interconnects three AHB masters to four AHB slaves. The normal latency to connect a master to a slave is one cycle. The exception is the default master of the accessed slave which is connected directly (zero cycle latency).

The Bus Matrix user interface also provides a System I/O Configuration user interface with registers that support application-specific features.

20.2 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
- Register Write Protection

20.3 Master/Slave Management

20.3.1 Matrix Masters

The Bus Matrix manages three masters. Each master can perform an access concurrently with others to an available slave.

Each master has its own specifically-defined decoder. In order to simplify the addressing, all the masters have the same decoding.

Table 20-1. List of Bus Matrix Masters

Master 0	Processor Instruction/Data
Master 1	Processor System
Master 2	Peripheral DMA Controller (PDC)

20.3.2 Matrix Slaves

The Bus Matrix manages four slaves. Each slave has its own arbiter, providing a different arbitration per slave.

Table 20-2. List of Bus Matrix Slaves

Slave 0	Internal SRAM
Slave 1	Internal ROM
Slave 2	Internal Flash
Slave 3	Peripheral Bridge

20.3.3 Master to Slave Access

Table 20-3 gives valid paths for master to slave access. The paths shown as "-" are forbidden or not wired, e.g. access from the processor I/D bus to internal SRAM..

Table 20-3. Master to Slave Access

Slaves	Masters	0	1	2
		Processor I/D Bus	Processor S Bus	PDC
0	Internal SRAM	–	X	X
1	Internal ROM	X	–	X
2	Internal Flash	X	–	–
3	Peripheral Bridge	–	X	X

20.4 Memory Mapping

The Bus Matrix provides one decoder for every AHB master interface. The decoder offers each AHB master several memory mappings. Depending on the product, each memory area may be assigned to several slaves. Thus it is possible to boot at the same address while using different AHB slaves.

20.5 Special Bus Granting Techniques

The Bus Matrix provides some speculative bus granting techniques in order to anticipate access requests from some masters, reducing latency at the first access of a burst or single transfer. The bus granting technique sets a default master for every slave.

At the end of the current access, if no other request is pending, the slave remains connected to its associated default master. A slave can be associated with three kinds of default masters:

- No default master
- Last access master
- Fixed default master

20.5.1 No Default Master

At the end of the current access, if no other request is pending, the slave is disconnected from all masters. This is suitable when the device is in low-power mode.

20.5.2 Last Access Master

At the end of the current access, if no other request is pending, the slave remains connected to the last master that performed an access request.

20.5.3 Fixed Default Master

At the end of the current access, if no other request is pending, the slave connects to its fixed default master. Unlike the last access master, the fixed master does not change unless the user modifies it by software (field FIXED_DEFMSTR of the related MATRIX_SCFG).

To change from one kind of default master to another, the Bus Matrix user interface provides the Slave Configuration registers (MATRIX_SCFGx), one for each slave, used to set a default master for each slave. MATRIX_SCFGx contain the fields DEFMSTR_TYPE and FIXED_DEFMSTR. The 2-bit DEFMSTR_TYPE field selects the default master type (no default, last access master, fixed default master) whereas the 4-bit FIXED_DEFMSTR field selects a fixed default master, provided that DEFMSTR_TYPE is set to fixed default master. Refer to the Bus Matrix user interface description.

20.6 Arbitration

The Bus Matrix provides an arbitration technique that reduces latency when conflicting cases occur; for example, when two or more masters try to access the same slave at the same time. One arbiter per AHB slave is provided, so that each slave can be arbitrated differently.

The Bus Matrix provides the user with two types of arbitration for each slave:

1. Round-robin arbitration (default)
2. Fixed priority arbitration

This selection is made by the field ARBT of MATRIX_SCFG.

Each algorithm may be complemented by selecting a default master configuration for each slave.

When a re-arbitration must be done, specific conditions apply. See Section 20.6.1 “Arbitration Rules“.

20.6.1 Arbitration Rules

Each arbiter has the ability to arbitrate between two or more masters’ requests. To avoid burst breaking and to provide the maximum throughput for slave interfaces, arbitration should 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 a defined burst length, predicted end of burst matches the size of the transfer but is managed differently for undefined length burst. See Section 20.6.1.1 “Undefined Length Burst Arbitration” on page 398“.
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 20.6.1.2 “Slot Cycle Limit Arbitration” on page 398.

20.6.1.1 Undefined Length Burst Arbitration

In order to prevent slave handling during undefined length bursts (INCR), the Bus Matrix provides specific logic in order to re-arbitrate before the end of the INCR transfer.

A predicted end of burst is used as for defined length burst transfer, which is selected between the following:

1. Infinite: No predicted end of burst is generated and therefore INCR burst transfer will never be broken.
2. Four beat bursts: Predicted end of burst is generated at the end of each four beat boundary inside INCR transfer.
3. Eight beat bursts: Predicted end of burst is generated at the end of each eight beat boundary inside INCR transfer.
4. Sixteen beat bursts: Predicted end of burst is generated at the end of each sixteen beat boundary inside INCR transfer.

This selection is made through the field ULBT of the Master Configuration registers (MATRIX_MCFG).

20.6.1.2 Slot Cycle Limit Arbitration

The Bus Matrix contains specific logic to break long accesses, such as very long bursts on a very slow slave (e.g. an external low speed memory). At the beginning of the burst access, a counter is loaded with the value previously written in

the SLOT_CYCLE field of the related MATRIX_SCFG and decreased at each clock cycle. When the counter reaches zero, the arbiter has the ability to re-arbitrate at the end of the current byte, half-word or word transfer.

20.6.1.3 Round-Robin Arbitration

The Bus Matrix arbiters use the round-robin algorithm to dispatch the requests from different masters to the same slave. If two or more masters make a request at the same time, the master with the lowest number is serviced first. The others are then serviced in a round-robin manner.

Three round-robin algorithms are implemented:

- Round-robin arbitration without default master
- Round-robin arbitration with last access master
- Round-robin arbitration with fixed default master

Round-robin arbitration without default master

Round-robin arbitration without default master is the main algorithm used by Bus Matrix arbiters. Using this algorithm, the Bus Matrix dispatches requests from different masters to the same slave in a pure round-robin manner. At the end of the current access, if no other request is pending, the slave is disconnected from all masters. This configuration incurs one latency cycle for the first access of a burst. Arbitration without default master can be used for masters that perform significant bursts.

Round-robin arbitration with last access master

Round-robin arbitration with last access master is a biased round-robin algorithm used by Bus Matrix arbiters to remove one latency cycle for the last master that accessed the slave. At the end of the current transfer, if no other master request is pending, the slave remains connected to the last master that performs the access. Other non- privileged masters still get one latency cycle if they attempt to access the same slave. This technique can be used for masters that mainly perform single accesses.

Round-robin arbitration with fixed default master

Round-robin arbitration with fixed default master is an algorithm used by the Bus Matrix arbiters to remove the one latency cycle for the fixed default master per slave. At the end of the current access, the slave remains connected to its fixed default master. Every request attempted by this fixed default master will not cause any latency whereas other non-privileged masters will still get one latency cycle. This technique can be used for masters that mainly perform single accesses.

20.6.1.4 Fixed Priority Arbitration

The fixed priority algorithm is used by the Bus Matrix arbiters to dispatch the requests from different masters to the same slave by using the fixed priority defined by the user. If two or more master's requests are active at the same time, the master with the highest priority number is serviced first. If two or more master's requests with the same priority are active at the same time, the master with the highest number is serviced first.

For each slave, the priority of each master may be defined through the Priority registers for slaves (MATRIX_PRAS and MATRIX_PRBS).

20.7 System I/O Configuration

The System I/O Configuration register (CCFG_SYSIO) configures I/O lines in system I/O mode (such as JTAG, ERASE, etc.) or as general-purpose I/O lines. Enabling or disabling the corresponding I/O lines in peripheral mode or in PIO mode (PIO_PER or PIO_PDR registers) in the PIO controller has no effect. However, the direction (input or output), pull-up, pull-down and other mode control is still managed by the PIO controller.

20.8 Register Write Protection

To prevent any single software error from corrupting 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 bit is automatically cleared after reading MATRIX_WPSR.

The following registers can be write-protected:

- "Bus Matrix Master Configuration Registers"
- "Bus Matrix Slave Configuration Registers"
- "Bus Matrix Priority Registers For Slaves"
- "System I/O Configuration Register"

20.9 Bus Matrix (MATRIX) User Interface

Table 20-4. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	Master Configuration Register 0	MATRIX_MCFG0	Read/Write	0x00000000
0x0004	Master Configuration Register 1	MATRIX_MCFG1	Read/Write	0x00000000
0x0008	Master Configuration Register 2	MATRIX_MCFG2	Read/Write	0x00000000
0x000C - 0x003C	Reserved	–	–	–
0x0040	Slave Configuration Register 0	MATRIX_SCFG0	Read/Write	0x00010010
0x0044	Slave Configuration Register 1	MATRIX_SCFG1	Read/Write	0x00050010
0x0048	Slave Configuration Register 2	MATRIX_SCFG2	Read/Write	0x00000010
0x004C	Slave Configuration Register 3	MATRIX_SCFG3	Read/Write	0x00000010
0x0050 - 0x007C	Reserved	–	–	–
0x0080	Priority Register A for Slave 0	MATRIX_PRAS0	Read/Write	0x00000000
0x0084	Reserved	–	–	–
0x0088	Priority Register A for Slave 1	MATRIX_PRAS1	Read/Write	0x00000000
0x008C	Reserved	–	–	–
0x0090	Priority Register A for Slave 2	MATRIX_PRAS2	Read/Write	0x00000000
0x0094	Reserved	–	–	–
0x0098	Priority Register A for Slave 3	MATRIX_PRAS3	Read/Write	0x00000000
0x009C - 0x0110	Reserved	–	–	–
0x0114	System I/O Configuration Register	CCFG_SYSIO	Read/Write	0x00000000
0x0118- 0x011C	Reserved	–	–	–
0x0120 - 0x010C	Reserved	–	–	–
0x1E4	Write Protection Mode Register	MATRIX_WPMR	Read/Write	0x0
0x1E8	Write Protection Status Register	MATRIX_WPSR	Read-only	0x0
0x0110 - 0x01FC	Reserved	–	–	–

20.9.1 Bus Matrix Master Configuration Registers

Name: MATRIX_MCFG0..MATRIX_MCFG2

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		

- **ULBT: Undefined Length Burst Type**

Value	Name	Description
0	INFINITE	No predicted end of burst is generated and therefore INCR bursts coming from this master cannot be broken.
1	SINGLE	The undefined length burst is treated as a succession of single access allowing rearbitration at each beat of the INCR burst.
2	FOUR_BEAT	The undefined length burst is split into a 4-beat bursts allowing rearbitration at each 4-beat burst end.
3	EIGHT_BEAT	The undefined length burst is split into 8-beat bursts allowing rearbitration at each 8-beat burst end.
4	SIXTEEN_BEAT	The undefined length burst is split into 16-beat bursts allowing rearbitration at each 16-beat burst end.

20.9.2 Bus Matrix Slave Configuration Registers

Name: MATRIX_SCFG0..MATRIX_SCFG3

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	ARBT	

23	22	21	20	19	18	17	16
–	–	–	FIXED_DEFMSTR			DEFMSTR_TYPE	

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
SLOT_CYCLE							

- **SLOT_CYCLE: Maximum Number of Allowed Cycles for a Burst**

When the SLOT_CYCLE limit is reached for a burst, it may be broken by another master trying to access this slave.

This limit has been placed to avoid locking very slow slaves when very long bursts are used.

This limit should not be very small. An unreasonably small value will break every burst and the Bus Matrix will spend its time arbitrating without performing any data transfers. 16 cycles is a reasonable value for SLOT_CYCLE.

- **DEFMSTR_TYPE: Default Master Type**

Value	Name	Description
0	NO_DEFAULT	At the end of current slave access, if no other master request is pending, the slave is disconnected from all masters. This results in having a one cycle latency for the first access of a burst transfer or for a single access.
1	LAST	At the end of 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 the one cycle latency when the last master tries to access the slave again.
2	FIXED	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 the one cycle latency when the fixed master tries to access the slave again.

- **FIXED_DEFMSTR: Fixed Default Master**

The number of the default master for this slave. Only used if DEFMSTR_TYPE is 2. Specifying the number of a master which is not connected to the selected slave is equivalent to setting DEFMSTR_TYPE to 0.

- **ARBT: Arbitration Type**

Value	Name	Description
0	ROUND_ROBIN	Round-robin arbitration
1	FIXED_PRIORITY	Fixed priority arbitration

20.9.3 Bus Matrix Priority Registers For Slaves

Name: MATRIX_PRAS0..MATRIX_PRAS3

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
–	–	M3PR		–	–	M2PR	

7	6	5	4	3	2	1	0
–	–	M1PR		–	–	M0PR	

- **MxPR: Master x Priority**

Fixed priority of master x to access the selected slave. The higher the number, the higher the priority.

20.9.4 System I/O Configuration Register

Name: CCFG_SYSIO

Access Read/Write

Reset: 0x0000_0000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	SYSIO12	–	–	–	–
7	6	5	4	3	2	1	0
SYSIO7	SYSIO6	SYSIO5	SYSIO4	–	–	–	–

- **SYSIO4: PB4 or TDI Assignment**

0: TDI function selected.

1: PB4 function selected.

- **SYSIO5: PB5 or TDO/TRACESWO Assignment**

0: TDO/TRACESWO function selected.

1: PB5 function selected.

- **SYSIO6: PB6 or TMS/SWDIO Assignment**

0: TMS/SWDIO function selected.

1: PB6 function selected.

- **SYSIO7: PB7 or TCK/SWCLK Assignment**

0: TCK/SWCLK function selected.

1: PB7 function selected.

- **SYSIO12: PB12 or ERASE Assignment**

0: ERASE function selected.

1: PB12 function selected.

20.9.5 Write Protection Mode Register

Name: MATRIX_WPMR

Access: Read/Write

Reset: See Table 20-4, “Register Mapping”

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 20.8 “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 20.8 “Register Write Protection“ for the list of registers that can be write-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.

20.9.6 Write Protection Status Register

Name: MATRIX_WPSR

Access: Read-only

Reset: See Table 20-4, “Register Mapping”

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 MATRIX_WPSR.

1: A write protection violation has occurred since the last read of the MATRIX_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.

21. Peripheral DMA Controller (PDC)

21.1 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.

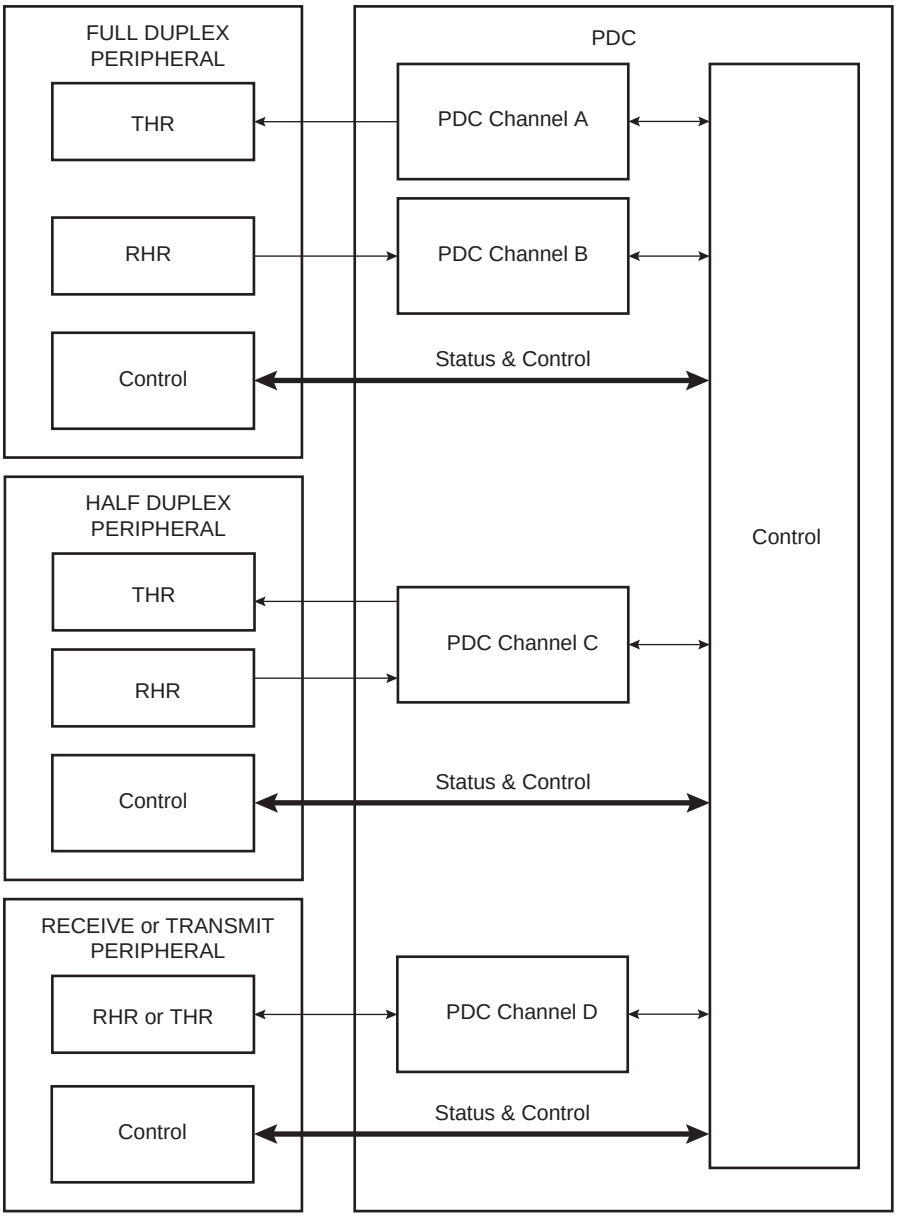
To launch a transfer, the peripheral triggers its associated PDC channels by using transmit and receive signals. When the programmed data is transferred, an end of transfer interrupt is generated by the peripheral itself.

21.2 Embedded Characteristics

- Performs Transfers to/from APB Communication Serial Peripherals
- Supports Half-duplex and Full-duplex Peripherals

21.3 Block Diagram

Figure 21-1. Block Diagram



21.4 Functional Description

21.4.1 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 21.4.3 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.

21.4.2 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.

21.4.3 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 Peripheral Status register (PERIPH_PTSR).

21.4.4 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.

21.4.5 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.

21.4.5.1 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.

21.4.5.2 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.

21.4.5.3 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.

21.4.5.4 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.

21.5 Peripheral DMA Controller (PDC) User Interface

Table 21-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Receive Pointer Register	PERIPH[(1)]_RPR	Read/Write	0
0x04	Receive Counter Register	PERIPH_RCR	Read/Write	0
0x08	Transmit Pointer Register	PERIPH_TPR	Read/Write	0
0x0C	Transmit Counter Register	PERIPH_TCR	Read/Write	0
0x10	Receive Next Pointer Register	PERIPH_RNPR	Read/Write	0
0x14	Receive Next Counter Register	PERIPH_RNCR	Read/Write	0
0x18	Transmit Next Pointer Register	PERIPH_TNPR	Read/Write	0
0x1C	Transmit Next Counter Register	PERIPH_TNCR	Read/Write	0
0x20	Transfer Control Register	PERIPH_PTCR	Write-only	0
0x24	Transfer Status Register	PERIPH_PTSR	Read-only	0

Note: 1. PERIPH: Ten registers are mapped in the peripheral memory space at the same offset. These can be defined by the user depending on the function and the desired peripheral.

21.5.1 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.

21.5.2 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.

21.5.3 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.

21.5.4 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.

21.5.5 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.

21.5.6 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.

21.5.7 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.

21.5.8 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.

21.5.9 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.

21.5.10 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.

22. Clock Generator

22.1 Description

The Clock Generator user interface is embedded within the Power Management Controller and is described in Section 23.16 ”Power Management Controller (PMC) User Interface”. However, the Clock Generator registers are named CKGR_.

22.2 Embedded Characteristics

The Clock Generator is made up of:

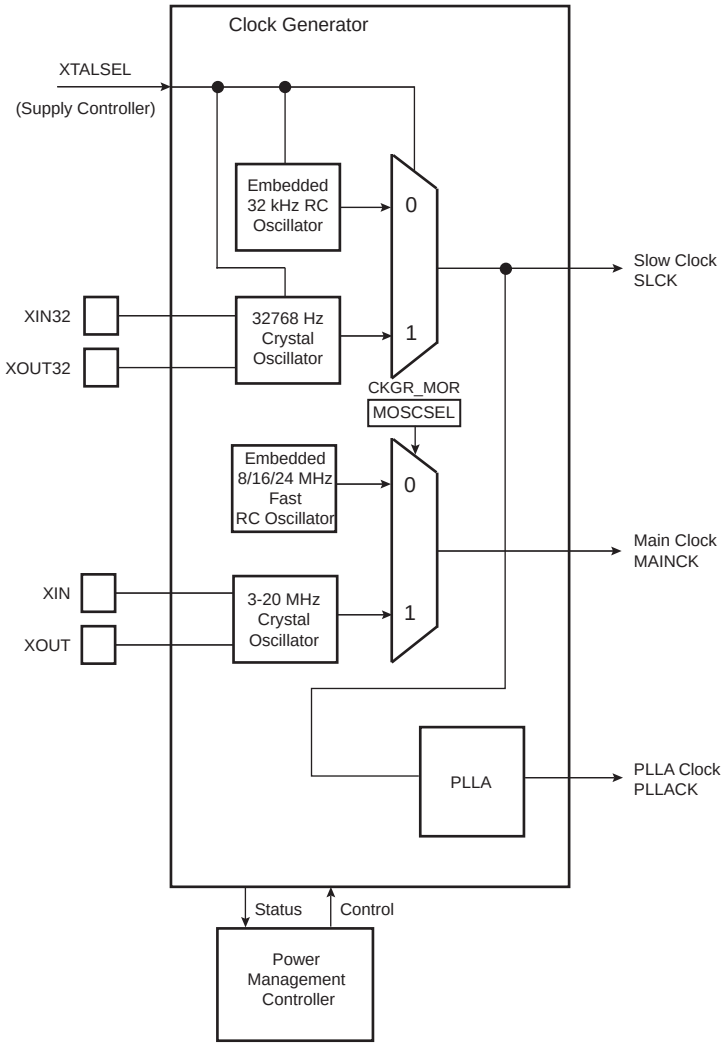
- A low-power 32768 Hz slow clock oscillator with bypass mode
- A low-power RC oscillator
- A 3 to 20 MHz crystal or ceramic resonator-based oscillator, which can be bypassed.
- A factory-programmed fast RC oscillator. Three output frequencies can be selected: 8/16/24 MHz. By default 8 MHz is selected.
- A 24 to 48 MHz programmable PLL (input from 32 kHz), capable of providing the clock MCK to the processor and to the peripherals.
- Write Protected Registers

It provides the following clocks:

- SLCK, the slow clock, which is the only permanent clock within the system.
- MAINCK is the output of the main clock oscillator selection: either the crystal or ceramic resonator-based oscillator or 8/16/24 MHz fast RC oscillator.
- PLLACK is the output of the 24 to 48 MHz programmable PLL (PLLA).

22.3 Block Diagram

Figure 22-1. Clock Generator Block Diagram



22.4 Slow Clock

The Supply Controller embeds a slow clock generator that is supplied with the VDDIO power supply. As soon as the VDDIO is supplied, both the crystal oscillator and the embedded RC oscillator are powered up, but only the embedded RC oscillator is enabled. This allows the slow clock to be valid in a short time (about 100 µs).

The slow clock is generated either by the slow clock crystal oscillator or by the slow clock RC oscillator.

The selection between the RC or the crystal oscillator is made by writing the XTALSEL bit in the Supply Controller Control register (SUPC_CR).

22.4.1 Slow Clock RC Oscillator

By default, the slow clock RC oscillator is enabled and selected. The user has to take into account the possible drifts of the RC oscillator. More details are given in the section “DC Characteristics” of the product datasheet.

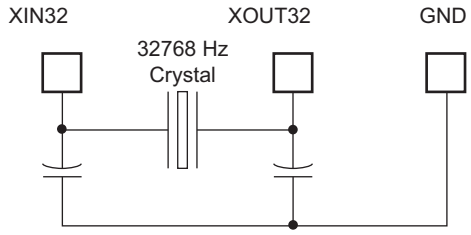
It can be disabled via the XTALSEL bit in SUPC_CR.

22.4.2 Slow Clock Crystal Oscillator

The Clock Generator integrates a 32768 Hz low-power oscillator. To use this oscillator, the XIN32 and XOUT32 pins must be connected to a 32768 Hz crystal. Two external capacitors must be wired as shown in Figure 22-2. More details are given in the section “DC Characteristics” of the product datasheet.

Note that the user is not obliged to use the slow clock crystal and can use the RC oscillator instead.

Figure 22-2. Typical Slow Clock Crystal Oscillator Connection



The user can select the crystal oscillator to be the source of the slow clock, as it provides a more accurate frequency. The command is made by writing SUPC_CR with the XTALSEL bit at 1. This results in a sequence which first configures the PIO lines multiplexed with XIN32 and XOUT32 to be driven by the oscillator, then enables the crystal oscillator and then disables the RC oscillator to save power. The switch of the slow clock source is glitch free. The OSCSEL bit of the Supply Controller Status register (SUPC_SR) or the OSCSEL bit of the PMC Status Register (PMC_SR) tracks the oscillator frequency downstream. It must be read in order to be informed when the switch sequence, initiated when a new value is written in the XTALSEL bit of SUPC_CR, is done.

Coming back on the RC oscillator is only possible by shutting down the VDDIO power supply. If the user does not need the crystal oscillator, the XIN32 and XOUT32 pins can be left unconnected since by default the XIN32 and XOUT32 system I/O pins are in PIO input mode with pull-up after reset.

The user can also set the crystal oscillator in bypass mode instead of connecting a crystal. In this case, the user has to provide the external clock signal on XIN32. The input characteristics of the XIN32 pin are given in the product electrical characteristics section. In order to set the bypass mode, the OSCBYPASS bit of the Supply Controller Mode Register (SUPC_MR) needs to be set at 1.

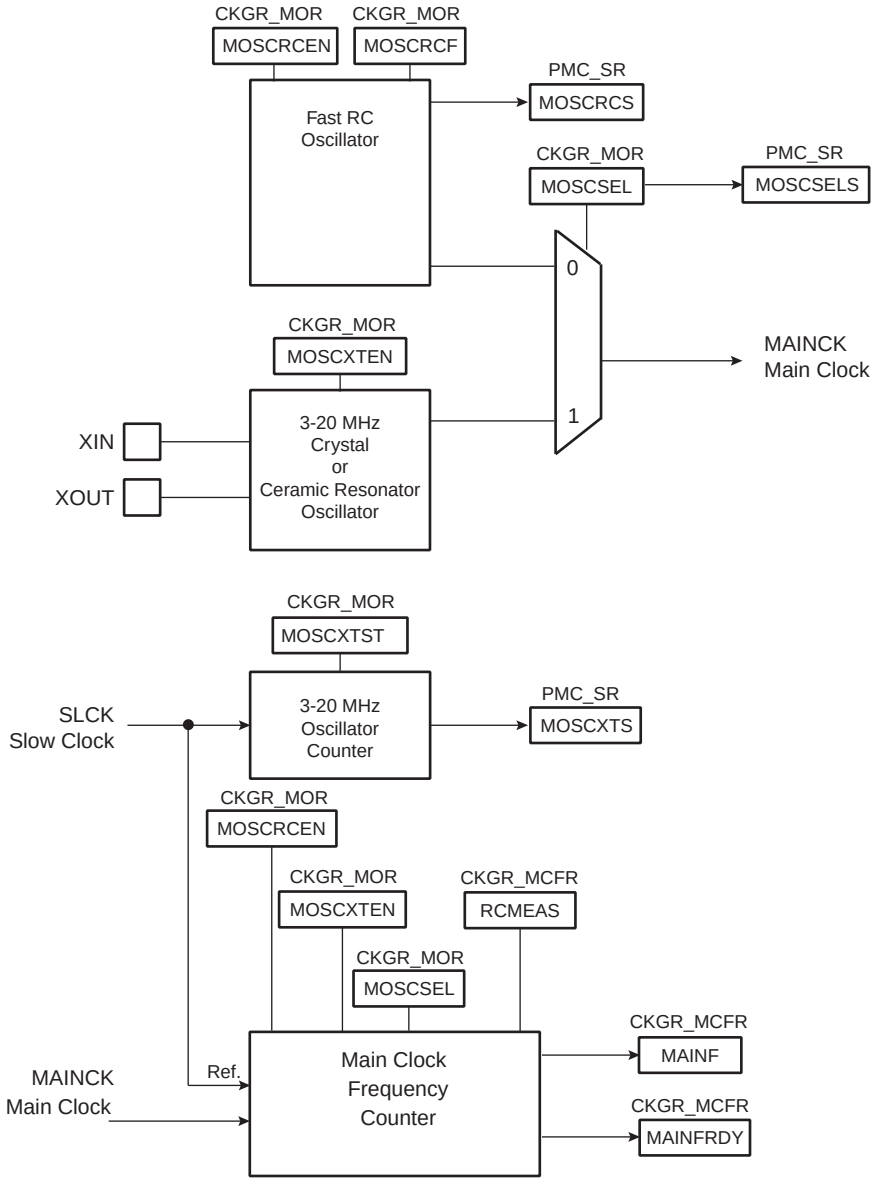
the user can set the slow clock crystal oscillator in bypass mode instead of connecting a crystal. In this case, the user has to provide the external clock signal on XIN32. The input characteristics of the XIN32 pin under these conditions are given in the product electrical characteristics section.

The programmer has to be sure to set the OSCBYPASS bit in SUPC_MR and XTALSEL bit in SUPC_CR.

22.5 Main Clock

Figure 22-3 shows the main clock block diagram.

Figure 22-3. Main Clock Block Diagram



The main clock has two sources:

- 8/16/24 MHz fast RC oscillator which starts very quickly and is used at startup.
- 3 to 20 MHz crystal or ceramic resonator-based oscillator which can be bypassed.

22.5.1 Fast RC Oscillator

After reset, the 8/16/24 MHz fast RC oscillator is enabled with the 8 MHz frequency selected and it is selected as the source of MAINCK. MAINCK is the default clock selected to start the system.

The fast RC oscillator frequencies are calibrated in production except the lowest frequency which is not calibrated.

Refer to the “DC Characteristics” section of the product datasheet.

The software can disable or enable the 8/16/24 MHz fast RC oscillator with the MOSCRCEN bit in the Clock Generator Main Oscillator Register (CKGR_MOR).

The user can also select the output frequency of the fast RC oscillator, either 8/16/24 MHz are available. It can be done through MOSCRCF bits in CKGR_MOR. When changing this frequency selection, the MOSCRCS bit in the Power Management Controller Status Register (PMC_SR) is automatically cleared and MAINCK is stopped until the oscillator is stabilized. Once the oscillator is stabilized, MAINCK restarts and MOSCRCS is set.

When disabling the main clock by clearing the MOSCRCEN bit in CKGR_MOR, the MOSCRCS bit in PMC_SR is automatically cleared, indicating the main clock is off.

Setting the MOSCRCS bit in the Power Management Controller Interrupt Enable Register (PMC_IER) can trigger an interrupt to the processor.

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 main oscillators.

The CAL8, CAL16 and CAL24 values in the PMC Oscillator Calibration Register (PMC_OCR) are the default values set by Atmel during production. These values are stored in a specific Flash memory area different from the main memory plane. These values cannot be modified by the user and cannot be erased by a Flash erase command or by the ERASE pin. Values written by the user's application in PMC_OCR are reset after each power up or peripheral reset.

22.5.2 Fast RC Oscillator Clock Frequency Adjustment

It is possible for the user to adjust the main RC oscillator frequency through PMC_OCR. By default, SEL8/16/24 are low, so the RC oscillator will be driven with Flash calibration bits which are programmed during chip production.

The user can adjust the trimming of the 8/16/24 MHz fast RC oscillator through this register in order to obtain more accurate frequency (to compensate derating factors such as temperature and voltage).

In order to calibrate the oscillator lower frequency, SEL8 must be set to 1 and a good frequency value must be configured in CAL8. Likewise, SEL16/24 must be set to 1 and a trim value must be configured in CAL16/24 in order to adjust the other frequencies of the oscillator.

It is possible to adjust the oscillator frequency while operating from this clock. For example, when running on lowest frequency it is possible to change the CAL8 value if SEL8 is set in PMC_OCR.

It is possible to restart, at anytime, a measurement of the main frequency by means of the RCMEAS bit in Main Clock Frequency Register (CKGR_MCFR). Thus, when MAINFRDY flag reads 1, another read access on CKGR_MCFR provides an image of the frequency of the main clock on MAINF field. The software can calculate the error with an expected frequency and correct the CAL8 (or CAL16/CAL24) field accordingly. This may be used to compensate frequency drift due to derating factors such as temperature and/or voltage.

22.5.3 3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator

After reset, the 3 to 20 MHz crystal or ceramic resonator-based oscillator is disabled and it is not selected as the source of MAINCK.

The user can select the 3 to 20 MHz crystal or ceramic resonator-based oscillator to be the source of MAINCK, as it provides a more accurate frequency. The software enables or disables the main oscillator so as to reduce power consumption by clearing the MOSCXTEN bit in CKGR_MOR.

When disabling the main oscillator by clearing the MOSCXTEN bit in CKGR_MOR, the MOSCXTS bit in PMC_SR is automatically cleared, indicating the main clock is off.

When enabling the main oscillator, the user must initiate the main oscillator counter with a value corresponding to the start-up time of the oscillator. This start-up time depends on the crystal frequency connected to the oscillator.

When the MOSCXTEN bit and the MOSCXTST are written in CKGR_MOR to enable the main oscillator, the XIN and XOUT pins are automatically switched into oscillator mode and MOSCXTS bit in PMC_SR is cleared and the counter starts counting down on the slow clock divided by 8 from the MOSCXTST value. Since the MOSCXTST value is coded with 8 bits, the maximum start-up time is about 62 ms.

When the counter reaches 0, the MOSCXTS bit is set, indicating that the main clock is valid. Setting the MOSCXTS bit in the Interrupt Mask Register (PMC_IMR) can trigger an interrupt to the processor.

22.5.4 Main Clock Oscillator Selection

The user can select either the 8/16/24 MHz fast RC oscillator or the 3 to 20 MHz crystal or ceramic resonator-based oscillator to be the source of main clock.

The advantage of the 8/16/24 MHz fast RC oscillator is that it provides fast start-up time, this is why it is selected by default (to start the system) and when entering wait mode.

The advantage of the 3 to 20 MHz crystal or ceramic resonator-based oscillator is that it is very accurate.

The selection is made by writing the MOSCSEL bit in CKGR_MOR. The switch of the main clock source is glitch free, so there is no need to run out of SLCK, PLLACK in order to change the selection. The MOSCSELS bit of PMC_SR indicates when the switch sequence is done.

Setting the MOSCSELS bit in PMC_IMR can trigger an interrupt to the processor.

Enabling the fast RC oscillator (MOSCRCEN = 1) and changing the fast RC frequency (MOSCCRF) at the same time is not allowed.

The fast RC must be enabled first and its frequency changed in a second step.

22.5.5 Switching Main Clock between the Main RC Oscillator and Fast Crystal Oscillator

Both sources must be enabled during the switchover operation. Only after completion can the unused oscillator be disabled. If switching to fast crystal oscillator, the clock presence must first be checked according to what is described in Section 22.5.6 ”Software Sequence to Detect the Presence of Fast Crystal” because the source may not be reliable (crystal failure or bypass on a non-existent clock).

22.5.6 Software Sequence to Detect the Presence of Fast Crystal

The frequency meter carried on CKGR_MCFR is operating on the selected main clock and not on the fast crystal clock nor on the fast RC oscillator clock.

Therefore, to check for the presence of the fast crystal clock, it is necessary to have the main clock (MAINCK) driven by the fast crystal clock (MOSCSEL=1).

The following software sequence order must be followed:

- MCK must select the slow clock (CSS=0 in the Master Clock register (PMC_MCKR) register).
- Wait for the MCKRDY flag in PMC_SR to be 1.
- The fast crystal must be enabled by programming 1 in the MOSCXTEN field in the CKGR_MOR register with the MOSCXTST field being programmed to the appropriate value (see the Electrical Characteristics chapter).
- Wait for the MOSCXTS flag to be 1 in PMC_SR to get the end of a start-up period of the fast crystal oscillator.
- Then, MOSCSEL must be programmed to 1 in CKGR_MOR to select fast main crystal oscillator for the main clock.
- MOSCSEL must be read until its value equals 1.
- Then the MOSCSELS status flag must be checked in PMC_SR.

At this point, 2 cases may occur (either MOSCSELS = 0 or MOSCSELS = 1).

- If MOSCSELS = 1, there is a valid crystal connected and its frequency can be determined by initiating a frequency measure by programming RCMEAS in CKGR_MCFR.
- If MOSCSELS = 0, there is no fast crystal clock (either no crystal connected or a crystal clock out of specification).
  A frequency measure can reinforce this status by initiating a frequency measure by programming RCMEAS in CKGR_MCFR.

- If MOSCSELS=0, the selection of the main clock must be programmed back to the main RC oscillator by writing MOSCSEL to 0 prior to disabling the fast crystal oscillator.
- If MOSCSELS=0, the crystal oscillator can be disabled (MOSCXTEN=0 in CKGR_MOR).

22.5.7 Main Clock Frequency Counter

The device features a main clock frequency counter that provides the frequency of the main clock.

The main clock frequency counter is reset and starts incrementing at the main clock speed after the next rising edge of the slow clock in the following cases:

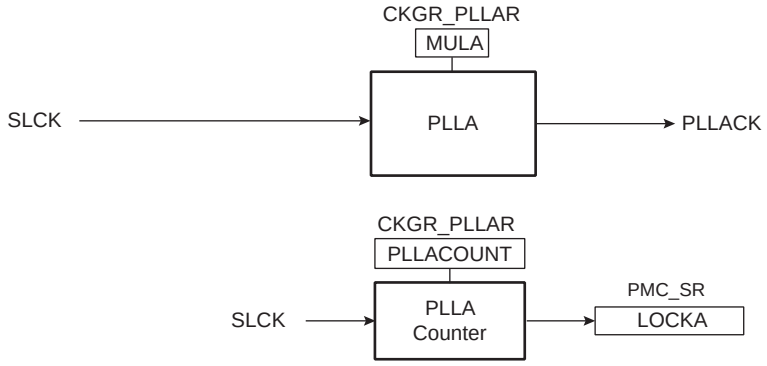
- When the 8/16/24 MHz fast RC oscillator clock is selected as the source of main clock and when this oscillator becomes stable (i.e., when the MOSCRCS bit is set)
- When the 3 to 20 MHz crystal or ceramic resonator-based oscillator is selected as the source of main clock and when this oscillator becomes stable (i.e., when the MOSCXTS bit is set)
- When the main clock oscillator selection is modified
- When the RCMEAS bit of CKGR_MFCR is written to 1.

Then, at the 16th falling edge of slow clock, the MAINFRDY bit in CKGR_MCFR) is set and the counter stops counting. Its value can be read in the MAINF field of CKGR_MCFR and gives the number of main clock cycles during 16 periods of slow clock, so that the frequency of the 8/16/24 MHz fast RC oscillator or 3 to 20 MHz crystal or ceramic resonator-based oscillator can be determined.

22.6 Divider and PLL Block

The device features one divider/one PLL block that permits a wide range of frequencies to be selected on either the master clock, the processor clock or the programmable clock outputs. Figure 22-4 shows the block diagram of the dividers and PLL blocks.

Figure 22-4. PLL Block Diagram



22.6.1 Phase Lock Loop Programming

The PLL (PLLA) allows multiplication of the SLCK clock source. The PLL clock signal has a frequency that depends on the respective source signal frequency and MUL (MULA). The factor applied to the source signal frequency is MUL + 1. When MUL is written to 0or PLLAEN=0,the PLL is disabled and its power consumption is saved. Re-enabling the PLL can be performed by writing a value higher than 0 in the MUL field and PLLAEN higher than 0.

To change the frequency of the PLLA, the PLLA must be first disabled by writing 0 in MULA field and 0 in PLLACOUNT field. Then, the PLLA can be configured to generate the new frequency by programming a new multiplier in MULA and the PLLACOUNT field in the same register access. See 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, the LOCK (LOCKA) bit in PMC_SR is automatically cleared. The values written in the PLLCOUNT field (PLLACOUNT) in CKGR_PLLR (CKGR_PLLAR) are loaded in the PLL counter. The PLL counter then decrements at the speed of the slow clock until it reaches 0. At this time, the LOCK bit is set in PMC_SR and can trigger an interrupt to the processor. The user has to load the number of slow clock cycles required to cover the PLL transient time into the PLLCOUNT field.

The PLL clock can be divided by 2 by writing the PLLDIV2 (PLLADIV2) bit in PMC_MCKR.

To avoid programming the PLL with a multiplication factor that is too high, the user can saturate the multiplication factor value sent to the PLL by setting the PLLA_MMAX field in PMC_PMMR.

It is forbidden to change the 8/16/24 MHz fast RC oscillator, or the main selection in CKGR_MOR while the master clock source is the PLL and the PLL reference clock is the fast RC oscillator.

The user must:

- Switch on the main RC oscillator by writing 1 in CSS field of PMC_MCKR.
- Change the frequency (MOSCRCF) or oscillator selection (MOSCSEL) in CKGR_MOR.
- Wait for MOSCRCS (if frequency changes) or MOSCSELS (if oscillator selection changes) in PMC_SR.
- Disable and then enable the PLL (LOCK in PMC_IDR and PMC_IER).
- Wait for LOCK flag in PMC_SR.
- Switch back to PLL by writing the appropriate value to CSS field of PMC_MCKR.

23. Power Management Controller (PMC)

23.1 Description

The Power Management Controller (PMC) optimizes power consumption by controlling all system and user peripheral clocks. The PMC enables/disables the clock inputs to many of the peripherals and the Cortex-M4 processor.

The Supply Controller selects between the 32 kHz RC oscillator or the slow crystal oscillator. The unused oscillator is disabled automatically so that power consumption is optimized.

By default, at startup, the chip runs out of the master clock using the fast RC oscillator running at 8 MHz.

The user can trim the 16 and 24 MHz RC oscillator frequencies by software.

23.2 Embedded Characteristics

The Power Management Controller provides the following clocks:

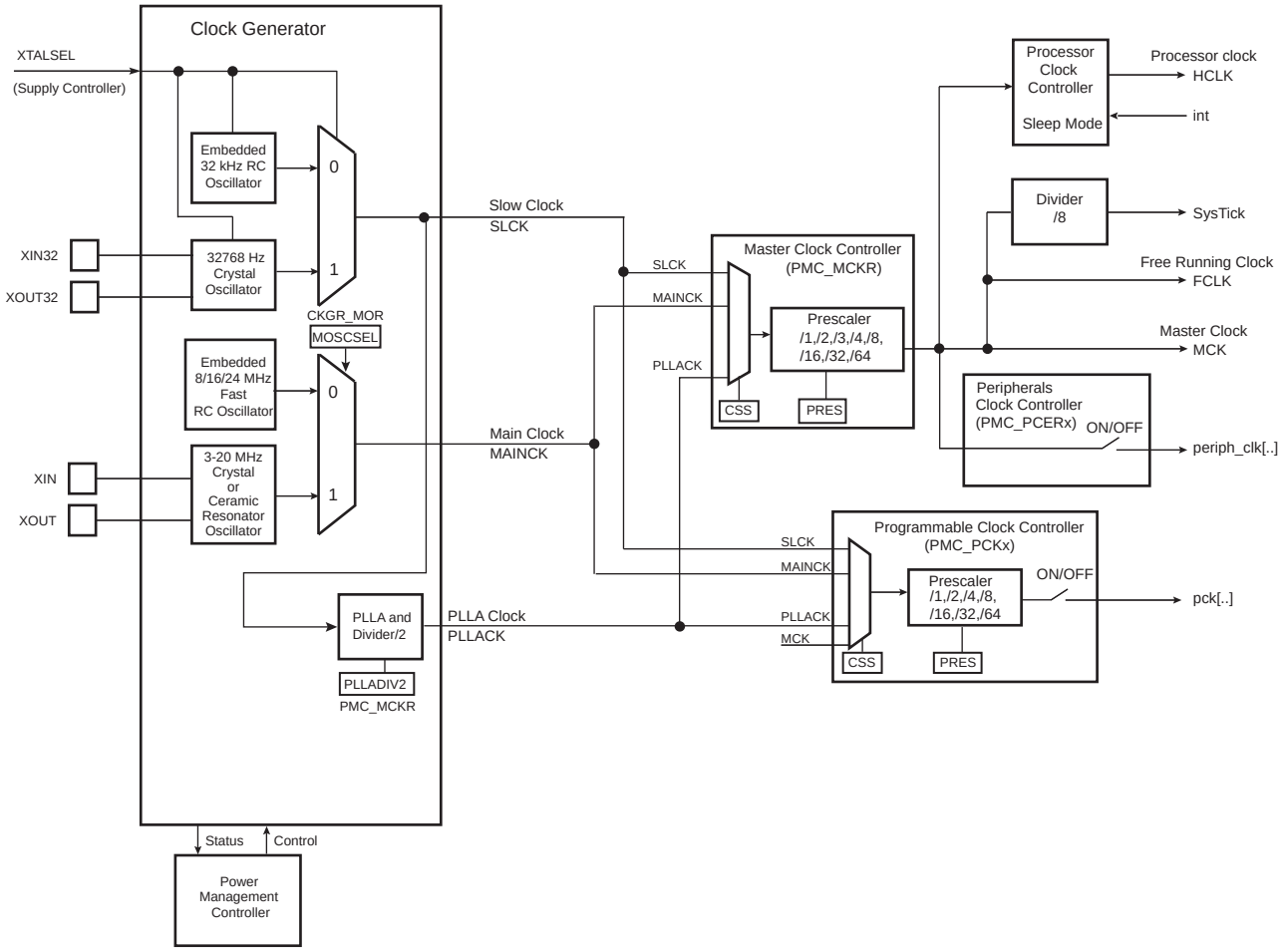
- MCK, the Master Clock, programmable from a few hundred Hz to the maximum operating frequency of the device. It is available to the modules running permanently, such as the Enhanced Embedded Flash Controller.
- Processor Clock (HCLK) , automatically switched off when entering the processor in Sleep Mode.
- Free running processor Clock (FCLK)
- The Cortex-M4 SysTick external clock
- Peripheral Clocks, typically MCK, provided to the embedded peripherals (USART, SPI, TWI, TC, etc.) and independently controllable. In order to reduce the number of clock names in a product, the Peripheral Clocks are named MCK in the product datasheet.
- Programmable Clock Outputs can be selected from the clocks provided by the clock generator and driven on the PCKx pins.
- Write Protected Registers

The Power Management Controller also provides the following operations on clocks:

- A main crystal oscillator clock failure detector.
- A frequency counter on main clock and an on-the-fly adjustable main RC oscillator frequency.

23.3 Block Diagram

Figure 23-1. General Clock Block Diagram



23.4 Master Clock Controller

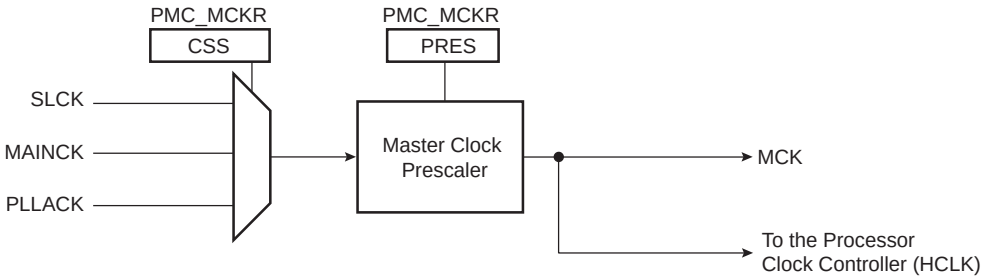
The Master Clock Controller provides selection and division of the master clock (MCK). MCK is the clock provided to all the peripherals. The master clock is selected from one of the clocks provided by the Clock Generator.

Selecting the slow clock provides a slow clock signal to the whole device. Selecting the main clock saves power consumption of the PLL. The Master Clock Controller is made up of a clock selector and a prescaler.

The master clock selection is made by writing the CSS field (Clock Source Selection) in PMC_MCKR. The prescaler supports the division by a power of 2 of the selected clock between 1 and 64, and the division by 3. The PRES field in PMC_MCKR programs the prescaler.

Each time PMC_MCKR is written to define a new master clock, the MCKRDY bit is cleared in PMC_SR. It reads 0 until the master clock is established. Then, the MCKRDY bit is set and can trigger an interrupt to the processor. This feature is useful when switching from a high-speed clock to a lower one to inform the software when the change is actually done.

Figure 23-2. Master Clock Controller



23.5 Processor Clock Controller

The PMC features a Processor Clock Controller (HCLK) that implements the processor sleep mode. The processor clock can be disabled by executing the WFI (WaitForInterrupt) or the WFE (WaitForEvent) processor instruction while the LPM bit is at 0 in the PMC Fast Startup Mode Register (PMC_FSMR).

The processor clock HCLK is enabled after a reset and is automatically re-enabled by any enabled interrupt. The processor sleep mode is achieved by disabling the processor clock, which is automatically re-enabled by any enabled fast or normal interrupt, or by the reset of the product.

When processor sleep mode is entered, the current instruction is finished before the clock is stopped, but this does not prevent data transfers from other masters of the system bus.

23.6 SysTick Clock

The SysTick calibration value is fixed to 6000 which allows the generation of a time base of 1 ms with SysTick clock to the maximum frequency on MCK divided by 8.

23.7 Peripheral Clock Controller

The Power Management Controller controls the clocks of each embedded peripheral by means of the Peripheral Clock Controller. The user can individually enable and disable the clock on the peripherals.

The user can also enable and disable these clocks by writing Peripheral Clock Enable 0 (PMC_PCER0), Peripheral Clock Disable 0 (PMC_PCDR0). The status of the peripheral clock activity can be read in the Peripheral Clock Status Register (PMC_PCSR0) .

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, PMC_PCDR0, and PMC_PCSR0) is the Peripheral Identifier defined at the product level. The bit number corresponds to the interrupt source number assigned to the peripheral.

23.8 Free-Running Processor Clock

The free-running processor clock (FCLK) used for sampling interrupts and clocking debug blocks ensures that interrupts can be sampled, and sleep events can be traced, while the processor is sleeping. It is connected to master clock (MCK).

23.9 Programmable Clock Output Controller

The PMC controls 3 signals to be output on external pins, PCKx. Each signal can be independently programmed via the Programmable Clock Registers (PMC_PCKx).

PCKx can be independently selected between the slow clock (SLCK), the main clock (MAINCK), the PLLA clock (PLLACK),and the master clock (MCK) by writing the CSS field in PMC_PCKx. Each output signal can also be divided by a power of 2 between 1 and 64 by writing the PRES (Prescaler) field in PMC_PCKx.

Each output signal can be enabled and disabled by writing 1 in the corresponding bit, PCKx of PMC_SCER and PMC_SCDR, respectively. Status of the active programmable output clocks are given in the PCKx bits of PMC_SCSR.

Moreover, like the PCK, a status bit in PMC_SR indicates that the programmable clock is actually what has been programmed in the programmable clock registers.

As the Programmable Clock Controller does not manage with glitch prevention when switching clocks, it is strongly recommended to disable the programmable clock before any configuration change and to re-enable it after the change is actually performed.

23.10 Fast Startup

The device allows the processor to restart in less than 10 microseconds while the device exits Wait Mode only if the C-code function managing the wait mode entry and exit is linked to and executed from on-chip SRAM.

The fast startup time cannot be achieved when 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 23.11 ”Startup from Embedded Flash”.

Prior to instructing the device to enter wait mode, the internal sources of wake-up must be cleared. It must be verified that none of the enabled external wake-up inputs (WKUP) hold an active polarity.

The system enters wait mode either by setting the WAITMODE bit in CKGR_MOR, or by executing the WaitForEvent (WFE) instruction of the processor while the LPM bit is at 1 in PMC_FSMR. Immediately after setting the WAITMODE bit or using the WFE instruction, wait for the MCKRDY bit to be set in PMC_SR.

A fast startup is enabled upon the detection of a programmed level on one of the 16 wake-up inputs (WKUP) or upon an active alarm from the RTC and RTT. The polarity of the 16 wake-up inputs is programmable by writing the PMC Fast Startup Polarity Register (PMC_FSPR).

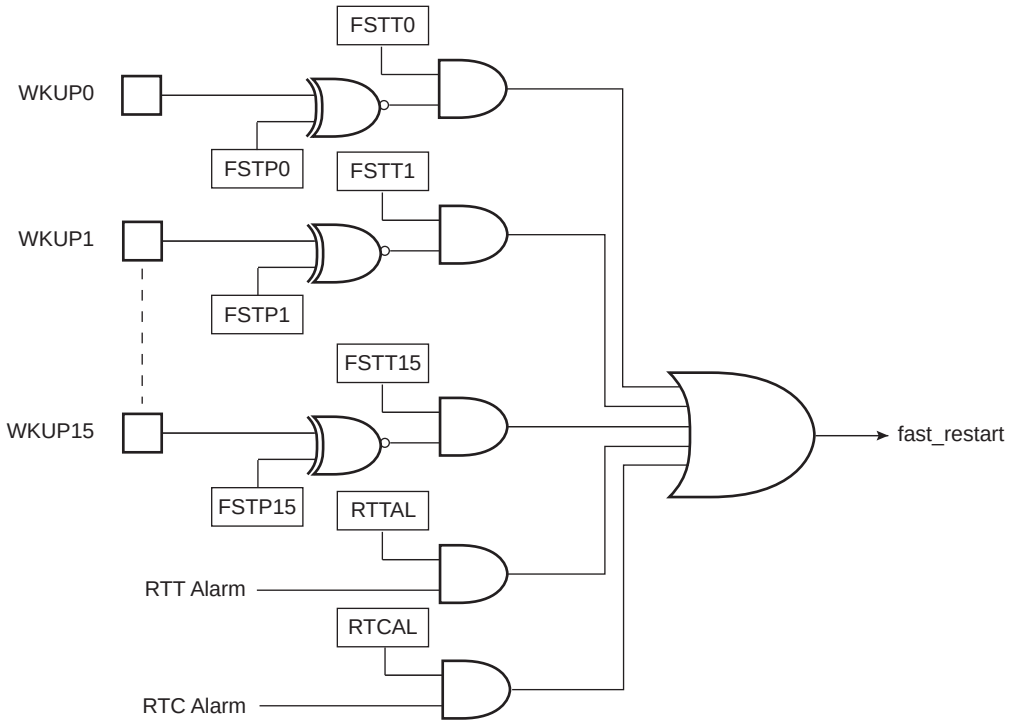
The fast startup circuitry, as shown in Figure 23-3, is fully asynchronous and provides a fast startup signal to the Power Management Controller. As soon as the fast startup signal is asserted, the embedded 8/16/24 MHz fast RC oscillator restarts automatically.

When entering wait mode, the embedded Flash can be placed in one of the low-power modes (deep-power-down or standby) depending on the configuration of the FLPM field in the PMC_FSMR. The FLPM field can be programmed at anytime and its value will be applied to the next wait mode period.

The power consumption reduction is optimal when configuring 1 (deep-power-down mode) in field FLPM. If 0 is programmed (standby mode), the power consumption is slightly higher than in deep-power-down mode.

When programming 2 in field FLPM, the wait mode Flash power consumption is equivalent to that of the active mode when there is no read access on the Flash.

Figure 23-3. 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, but the user can easily recover this information by reading the PIO Controller and the status registers of the RTC and RTT.

23.11 Startup from Embedded Flash

The inherent start-up time of the embedded Flash cannot provide a fast startup of the system.

If system fast start-up time is not required, the first instruction after a wait mode exit can be located in the embedded Flash. Under these conditions, prior to entering wait mode, the Flash controller must be programmed to perform access in 0 wait-state (see Flash controller section).

If the fast RC oscillator is configured to generate 16 MHz or 24 MHz (MOSCRCF=1 or 2 in CKGR_MOR), the first instruction after an exit must not be located in the embedded Flash and the fast startup procedure must be used (see Section 23.10 ”Fast Startup”). If the fast RC oscillator is configured to generate 8 MHz (MOSCRCF= 0 in CKGR_MOR), the instructions managing start-up time can be located in any on-chip memory.

The procedure and conditions to enter wait mode and the circuitry to exit wait mode are strictly the same as fast startup (see Section 23.10 ”Fast Startup”).

23.12 Main Clock Failure Detector

The clock failure detector monitors the main crystal oscillator or ceramic resonator-based oscillator to identify an eventual failure of this oscillator.

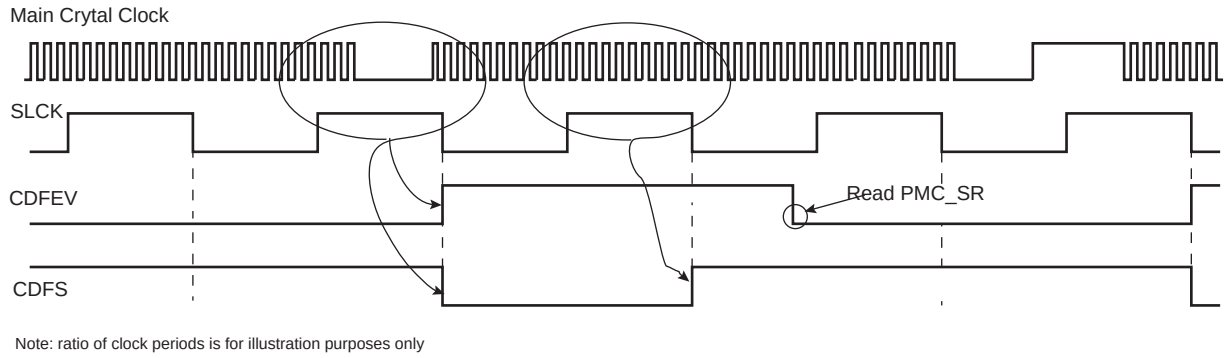
The clock failure detector can be enabled or disabled by bit CFDEN in CKGR_MOR. After a VDDCORE reset, the detector is disabled. However, if the oscillator is disabled (MOSCXTEN = 0), the detector is disabled too.

A failure is detected by means of a counter incrementing on the main oscillator clock edge and timing logic clocked on the slow RC oscillator controlling the counter. Thus, the slow RC oscillator must be enabled.

The counter is cleared when the slow RC oscillator clock signal is low and enabled when the signal is high. Thus the failure detection time is 1 slow RC oscillator clock period. If, during the high level period of the slow RC oscillator clock signal, less than 8 fast crystal oscillator clock periods have been counted, then a failure is reported.

If a failure of the main oscillator is detected, bit CFDEV in PMC_SR indicates a failure event and generates an interrupt if the corresponding interrupt source is enabled. The interrupt remains active until a read occurs in PMC_SR. The user can know the status of the clock failure detection at any time by reading the CFDS bit in PMC_SR.

Figure 23-4. Clock Failure Detection (Example)



Note: ratio of clock periods is for illustration purposes only

If the main oscillator is selected as the source clock of MAINCK (MOSCSEL in CKGR_MOR = 1), and if the master clock source is PLLACK (CSS = 2), a clock failure detection automatically forces MAINCK to be the source clock for the master clock (MCK).Then, regardless of the PMC configuration, a clock failure detection automatically forces the fast RC oscillator to be the source clock for MAINCK. If the fast RC oscillator is disabled when a clock failure detection occurs, it is automatically re-enabled by the clock failure detection mechanism.

It takes 2 slow RC oscillator clock cycles to detect and switch from the main oscillator, to the fast RC oscillator if the source master clock (MCK) is main clock (MAINCK), or three slow clock RC oscillator cycles if the source of MCK is PLLACK.

The user can know the status of the clock failure detector at any time by reading the FOS bit in PMC_SR.

This fault output remains active until the defect is detected and until it is cleared by the bit FOCLR in the PMC Fault Output Clear Register (PMC_FOCR).

23.13 Programming Sequence

1. If the fast crystal oscillator is not required, the PLL and divider can be directly configured (Step 6.) else the fast crystal oscillator must be started (Step 2.).
2. Enable the fast crystal oscillator:

   The fast crystal oscillator is enabled by setting the MOSCXTEN field in CKGR_MOR. The user can define a start-up time. This can be achieved by writing a value in the MOSCXTST field in CKGR_MOR. Once this register has been correctly configured, the user must wait for MOSCXTS field in PMC_SR to be set. This can be done either by polling MOSCXTS in PMC_SR, or by waiting for the interrupt line to be raised if the associated interrupt source (MOSCXTS) has been enabled in PMC_IER.
3. Switch the MAINCK to the main crystal oscillator by setting MOSCSEL in CKGR_MOR.
4. Wait for the MOSCSELS to be set in PMC_SR to ensure the switchover is complete.
5. Check the main clock frequency:

   This main clock frequency can be measured via CKGR_MCFR.

   Read CKGR_MCFR until the MAINFRDY field is set, after which the user can read the MAINF field in CKGR_MCFR by performing an additional read. This provides the number of main clock cycles that have been counted during a period of 16 slow clock cycles.

If MAINF = 0, switch the MAINCK to the Fast RC Oscillator by clearing MOSCSEL in CKGR_MOR. If MAINF ≠ 0, proceed to Step 6.

6. Set PLLx and Divider (if not required, proceed to Step 7.):

   In the names PLLx, DIVx, MULx, LOCKx, PLLxCOUNT, and CKGR_PLLxR, 'x' represents A.

   All parameters needed to configure PLLx and the divider are located in CKGR_PLLxR.

   The DIVx field is used to control the divider itself. This parameter can be programmed between 0 and 127. Divider output is divider input divided by DIVx parameter. By default, DIVx field is set to 0 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 1500. If MULx is set to 0, PLLx will be turned off, otherwise the PLLx output frequency is PLLx input frequency multiplied by (MULx + 1).

   The PLLxCOUNT field specifies the number of slow clock cycles before the LOCKx bit is set in the PMC_SR after CKGR_PLLxR has been written.

   Once CKGR_PLLxR has been written, the user must wait for the LOCKx bit to be set in the PMC_SR. This can be done either by polling LOCKx in PMC_SR or by waiting for the interrupt line to be raised if the associated interrupt source (LOCKx) has been enabled in PMC_IER. All fields in CKGR_PLLxR can be programmed in a single write operation. If at some stage one of the following parameters, MULx or DIVx is modified, the LOCKx bit goes low to indicate that PLLx is not yet ready. When PLLx is locked, LOCKx is set again. The user must wait for the LOCKx bit to be set before using the PLLx output clock.

7. Select the master clock and processor clock

   The master clock and the processor clock are configurable via PMC_MCKR.

   The CSS field is used to select the clock source of the master clock and processor clock dividers. By default, the selected clock source is the main clock.

   The PRES field is used to define the processor clock and master clock prescaler. The user can choose between different values (1, 2, 3, 4, 8, 16, 32, 64). Prescaler output is the selected clock source frequency divided by the PRES value.

   Once the PMC_MCKR has been written, the user must wait for the MCKRDY bit to be set in the PMC_SR. This can be done either by polling MCKRDY in PMC_SR or by waiting for the interrupt line to be raised if the associated interrupt source (MCKRDY) has been enabled in PMC_IER. PMC_MCKR must not be programmed in a single write operation. The programming sequence for PMC_MCKR is as follows:

- If a new value for CSS field corresponds to PLL clock,
  - Program the PRES field in PMC_MCKR.
  - Wait for the MCKRDY bit to be set in PMC_SR.
  - Program the CSS field in PMC_MCKR.
  - Wait for the MCKRDY bit to be set in PMC_SR.
- If a new value for CSS field corresponds to main clock or slow clock,
  - Program the CSS field in PMC_MCKR.
  - Wait for the MCKRDY bit to be set in the PMC_SR.
  - Program the PRES field in PMC_MCKR.
  - Wait for the MCKRDY bit to be set in PMC_SR.

If at some stage parameters CSS or PRES is modified, the MCKRDY bit goes low to indicate that the master clock and the processor clock are not yet ready. The user must wait for MCKRDY bit to be set again before using the master and processor clocks.

Note: IF PLLx clock was selected as the master clock and the user decides to modify it by writing in CKGR_PLLxR, the MCKRDY flag will go low while PLLx is unlocked. Once PLLx is locked again, LOCKx goes high and MCKRDY is set.

While PLLx is unlocked, the master clock selection is automatically changed to slow clock for PLLA. For further information, see Section 23.14.2 ”Clock Switching Waveforms”.

Code Example:

```
write_register(PMC_MCKR,0x00000001)
wait (MCKRDY=1)
write_register(PMC_MCKR,0x00000011)
wait (MCKRDY=1)
```

The master clock is main clock divided by 2.

8. Select the programmable clocks

   Programmable clocks are controlled via registers, PMC_SCER, PMC_SCDR and PMC_SCSR.

   Programmable clocks can be enabled and/or disabled via PMC_SCER and PMC_SCDR. Three programmable clocks can be used. PMC_SCSR indicates which programmable clock is enabled. By default all programmable clocks are disabled.

   PMC_PCKx registers are used to configure programmable clocks.

   The CSS field is used to select the programmable clock divider source. Several clock options are available: main clock, slow clock, master clock, PLLACK, . The slow clock is the default clock source.

   The PRES field is used to control the programmable clock prescaler. It is possible to choose between different values (1, 2, 4, 8, 16, 32, 64). Programmable clock output is prescaler input divided by PRES parameter. By default, the PRES value is set to 0 which means that PCKx is equal to slow clock.

   Once PMC_PCKx register has been configured, the corresponding programmable clock must be enabled and the user is constrained to wait for the PCKRDYx bit to be set in the PMC_SR. This can be done either by polling PCKRDYx in PMC_SR or by waiting for the interrupt line to be raised if the associated interrupt source (PCKRDYx) has been enabled in PMC_IER. All parameters in PMC_PCKx can be programmed in a single write operation.

   If the CSS and PRES parameters are to be modified, the corresponding programmable clock must be disabled first. The parameters can then be modified. Once this has been done, the user must re-enable the programmable clock and wait for the PCKRDYx bit to be set.

9. Enable the peripheral clocks

   Once all of the previous steps have been completed, the peripheral clocks can be enabled and/or disabled via registers PMC_PCER0, PMC_PCDR0.

23.14 Clock Switching Details

23.14.1 Master Clock Switching Timings

Table 23-1 and give the worst case timings required for the master clock to switch from one selected clock to another one. This is in the event that the prescaler is de-activated. When the prescaler is activated, an additional time of 64 clock cycles of the newly selected clock has to be added.

Table 23-1. Clock Switching Timings (Worst Case)

From / To	Main Clock	SLCK	PLL Clock
Main Clock	–	4 x SLCK + 2.5 x Main Clock	3 x PLL Clock + 4 x SLCK + 1 x Main Clock
SLCK	0.5 x Main Clock + 4.5 x SLCK	–	3 x PLL Clock + 5 x SLCK
PLL Clock	0.5 x Main Clock + 4 x SLCK + PLLCOUNT x SLCK + 2.5 x PLLx Clock	2.5 x PLL Clock + 5 x SLCK + PLLCOUNT x SLCK	2.5 x PLL Clock + 4 x SLCK + PLLCOUNT x SLCK

Notes: 1. PLL designates the PLLA .
2. PLLCOUNT designates PLLACOUNT .

23.14.2 Clock Switching Waveforms

Figure 23-5. Switch Master Clock from Slow Clock to PLLx Clock

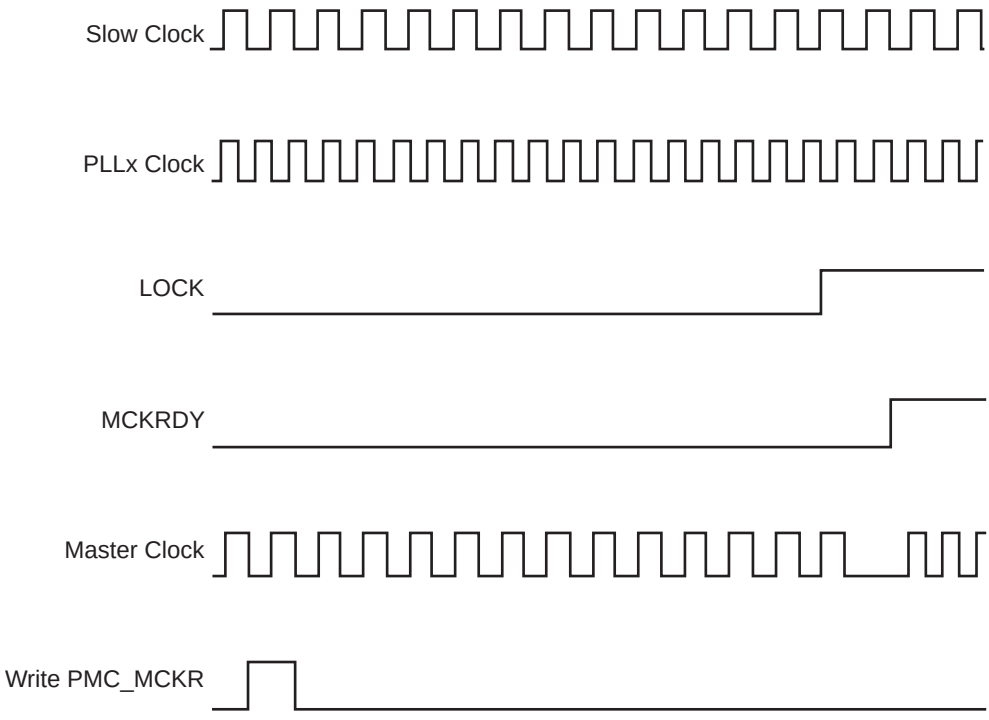


Figure 23-6. Switch Master Clock from Main Clock to Slow Clock

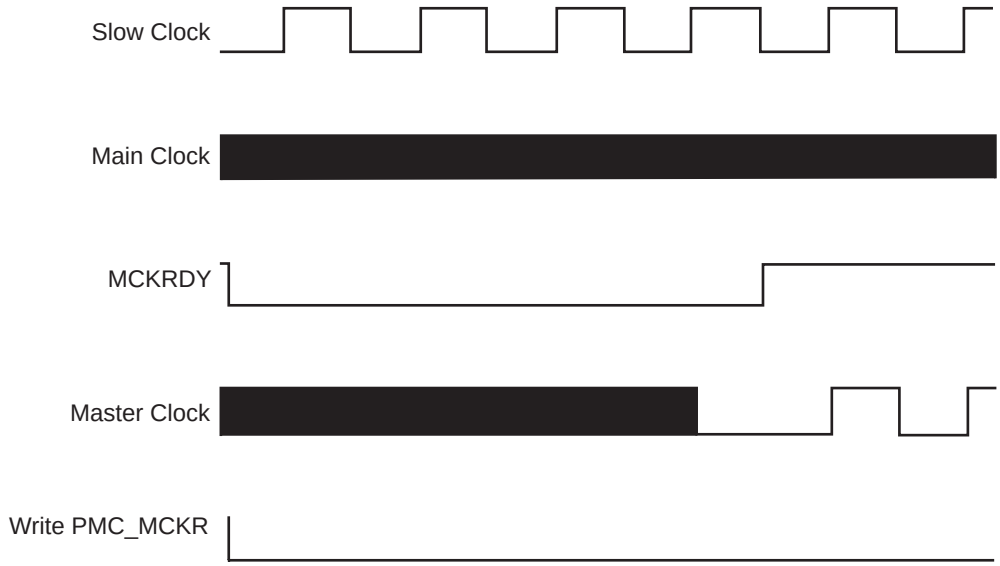


Figure 23-7. Change PLLx Programming

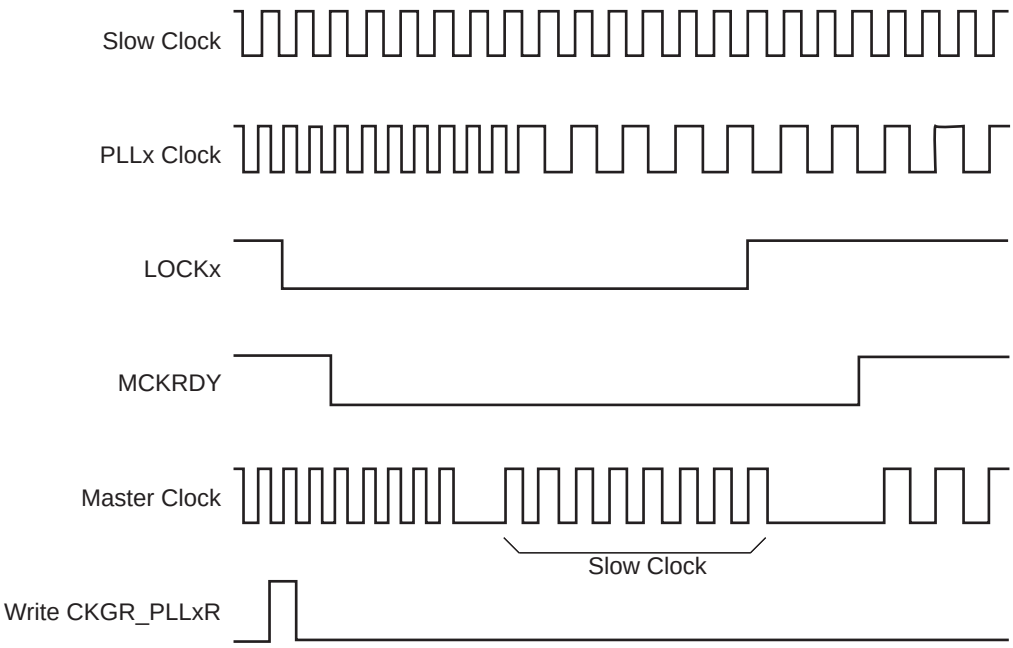
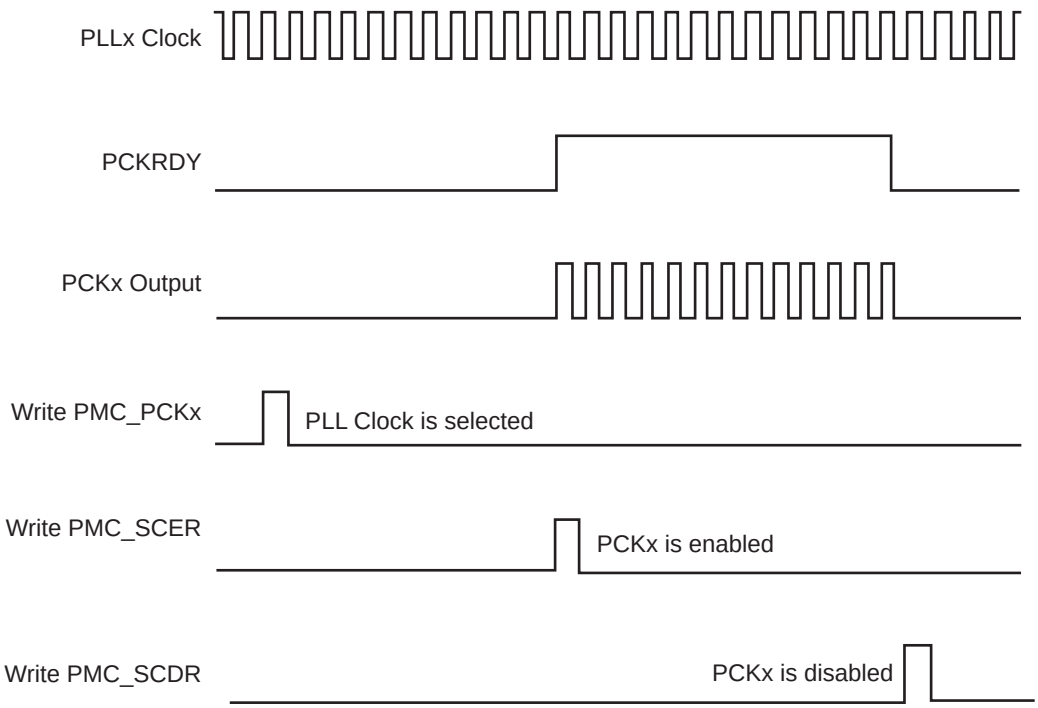


Figure 23-8. Programmable Clock Output Programming



23.15 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 PLLA Register"
- "PMC Master Clock Register"
- "PMC Programmable Clock Register"
- "PMC Fast Startup Mode Register"
- "PMC Fast Startup Polarity Register"
- "PMC Oscillator Calibration Register"
- "PLL Maximum Multiplier Value Register"

23.16 Power Management Controller (PMC) User Interface

Table 23-2. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	System Clock Enable Register	PMC_SCER	Write-only	–
0x0004	System Clock Disable Register	PMC_SCDR	Write-only	–
0x0008	System Clock Status Register	PMC_SCSR	Read-only	0x0000_0001
0x000C	Reserved	–	–	–
0x0010	Peripheral Clock Enable Register 0	PMC_PCER0	Write-only	–
0x0014	Peripheral Clock Disable Register 0	PMC_PCDR0	Write-only	–
0x0018	Peripheral Clock Status Register 0	PMC_PCSR0	Read-only	0x0000_0000
0x0020	Main Oscillator Register	CKGR_MOR	Read/Write	0x0000_0008
0x0024	Main Clock Frequency Register	CKGR_MCFR	Read/Write	0x0000_0000
0x0028	PLLA Register	CKGR_PLLAR	Read/Write	0x0000_3F00
0x002C	Reserved	–	–	–
0x0030	Master Clock Register	PMC_MCKR	Read/Write	0x0000_0001
0x0034 - 0x003C	Reserved	–	–	–
0x0040	Programmable Clock 0 Register	PMC_PCK0	Read/Write	0x0000_0000
0x0044	Programmable Clock 1 Register	PMC_PCK1	Read/Write	0x0000_0000
0x0048	Programmable Clock 2 Register	PMC_PCK2	Read/Write	0x0000_0000
0x004C - 0x005C	Reserved	–	–	–
0x0060	Interrupt Enable Register	PMC_IER	Write-only	–
0x0064	Interrupt Disable Register	PMC_IDR	Write-only	–
0x0068	Status Register	PMC_SR	Read-only	0x0001_0008
0x006C	Interrupt Mask Register	PMC_IMR	Read-only	0x0000_0000
0x0070	Fast Startup Mode Register	PMC_FSMR	Read/Write	0x0000_0000
0x0074	Fast Startup Polarity Register	PMC_FSPR	Read/Write	0x0000_0000
0x0078	Fault Output Clear Register	PMC_FOCR	Write-only	–
0x007C- 0x00E0	Reserved	–	–	–
0x00E4	Write Protection Mode Register	PMC_WPMR	Read/Write	0x0
0x00E8	Write Protection Status Register	PMC_WPSR	Read-only	0x0
0x00EC-0x00FC	Reserved	–	–	–
0x0100 - 0x0108	Reserved	–	–	–
0x010C	Reserved	–	–	–
0x0110	Oscillator Calibration Register	PMC_OCR	Read/Write	0x0040_4040
0114 - 0x120	Reserved	–	–	–
0x130	PLL Maximum Multiplier Value Register	PMC_PMMR	Read/Write	0x07FF07FF
0134 - 0x144	Reserved	–	–	–

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

23.16.1 PMC System Clock Enable Register

Name: PMC_SCER

Address: 0x400E0400

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

This register can only be written if the WPEN bit is cleared in “PMC Write Protection Mode Register” .

- **PCKx: Programmable Clock x Output Enable**

0: No effect.

1: Enables the corresponding Programmable Clock output.

23.16.2 PMC System Clock Disable Register

Name: PMC_SCDR

Address: 0x400E0404

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

This register can only be written if the WPEN bit is cleared in “PMC Write Protection Mode Register” .

- **PCKx: Programmable Clock x Output Disable**

0: No effect.

1: Disables the corresponding Programmable Clock output.

23.16.3 PMC System Clock Status Register

Name: PMC_SCSR

Address: 0x400E0408

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

- **PCKx: Programmable Clock x Output Status**

0: The corresponding Programmable Clock output is disabled.

1: The corresponding Programmable Clock output is enabled.

23.16.4 PMC Peripheral Clock Enable Register 0

Name: PMC_PCER0

Address: 0x400E0410

Access: Write-only

31	30	29	28	27	26	25	24
–	–	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 “PMC Write Protection Mode Register” .

- **PIDx: Peripheral Clock x Enable**

0: No effect.

1: Enables the corresponding peripheral clock.

Note: To get PIDx, refer to identifiers as defined in the section “Peripheral Identifiers” in the product datasheet.

Note: Programming the control bits of the Peripheral ID that are not implemented has no effect on the behavior of the PMC.

23.16.5 PMC Peripheral Clock Disable Register 0

Name: PMC_PCDR0

Address: 0x400E0414

Access: Write-only

31	30	29	28	27	26	25	24
–	–	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 “PMC Write Protection Mode Register” .

- **PIDx: Peripheral Clock x Disable**

0: No effect.

1: Disables the corresponding peripheral clock.

Note: To get PIDx, refer to identifiers as defined in the section “Peripheral Identifiers” in the product datasheet.

23.16.6 PMC Peripheral Clock Status Register 0

Name: PMC_PCSR0

Address: 0x400E0418

Access: Read-only

31	30	29	28	27	26	25	24
–	–	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: To get PIDx, refer to identifiers as defined in the section “Peripheral Identifiers” in the product datasheet.

23.16.7 PMC Clock Generator Main Oscillator Register

Name: CKGR_MOR

Address: 0x400E0420

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	CFDEN	MOSCSEL

23	22	21	20	19	18	17	16
KEY							

15	14	13	12	11	10	9	8
MOSCXTST							

7	6	5	4	3	2	1	0
–	MOSCRCF			MOSCRCEN	WAITMODE	MOSCXTBY	MOSCXTEN

This register can only be written if the WPEN bit is cleared in “PMC Write Protection Mode Register” .

- **MOSCXTEN: Main Crystal Oscillator Enable**

A crystal must be connected between XIN and XOUT.

0: The Main Crystal Oscillator is disabled.

1: The Main Crystal Oscillator is enabled. MOSCXTBY must be set to 0.

When MOSCXTEN is set, the MOSCXTS flag is set once the Main Crystal Oscillator start-up time is achieved.

- **MOSCXTBY: Main Crystal Oscillator Bypass**

0: No effect.

1: The Main Crystal Oscillator is bypassed. MOSCXTEN must be set to 0. An external clock must be connected on XIN.

When MOSCXTBY is set, the MOSCXTS flag in PMC_SR is automatically set.

Clearing MOSCXTEN and MOSCXTBY bits allows resetting the MOSCXTS flag.

- **WAITMODE: Wait Mode Command**

0: No effect.

1: Enters the device in Wait Mode.

Note: The WAITMODE bit is write-only.

- **MOSCRCEN: Main On-Chip RC Oscillator Enable**

0: The Main On-Chip RC Oscillator is disabled.

1: The Main On-Chip RC Oscillator is enabled.

When MOSCRCEN is set, the MOSCRCS flag is set once the Main On-Chip RC Oscillator start-up time is achieved.

- **MOSCRCF: Main On-Chip RC Oscillator Frequency Selection**

At startup, the main on-chip RC Oscillator frequency is 8 MHz.

Value	Name	Description
0x0	8_MHz	The Fast RC Oscillator Frequency is at 8 MHz (default)
0x1	16_MHz	The Fast RC Oscillator Frequency is at 16 MHz
0x2	24_MHz	The Fast RC Oscillator Frequency is at 24 MHz

Note: MOSCRCF must be changed only if MOSCRCS is set in the PMC_SR register. Therefore MOSCRCF and MOSCRCEN cannot be changed at the same time.

- **MOSCXTST: Main Crystal Oscillator Start-up Time**

Specifies the number of slow clock cycles multiplied by 8 for the main crystal oscillator start-up time.

- **KEY: Write Access Password**

Value	Name	Description
0x37	PASSWD	Writing any other value in this field aborts the write operation. Always reads as 0.

- **MOSCSEL: Main Oscillator Selection**

0: The main on-chip RC oscillator is selected.

1: The main crystal oscillator is selected.

- **CFDEN: Clock Failure Detector Enable**

0: The clock failure detector is disabled.

1: The clock failure detector is enabled.

Note: 1. The slow RC oscillator must be enabled when the CFDEN is enabled.

23.16.8 PMC Clock Generator Main Clock Frequency Register

Name: CKGR_MCFR

Address: 0x400E0424

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	RCMEAS	–	–	–	MAINFRDY

15	14	13	12	11	10	9	8
MAINF							

7	6	5	4	3	2	1	0
MAINF							

This register can only be written if the WPEN bit is cleared in “PMC Write Protection Mode Register” .

- **MAINF: Main Clock Frequency**

Gives the number of main clock cycles within 16 slow clock periods in order to determine the main clock frequency:

$f_{MCK} = (MAINF \times f_{SLCK}) / 16$ where frequency is in MHz.

- **MAINFRDY: Main Clock Ready**

0: MAINF value is not valid or the main oscillator is disabled or a measure has just been started by means of RCMEAS.

1: The main oscillator has been enabled previously and MAINF value is available.

Note: To ensure that a correct value is read on the MAINF 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: RC Oscillator Frequency Measure (write-only)**

0: No effect.

1: Restarts measuring of the main RC frequency. MAINF will carry the new frequency as soon as a low to high transition occurs on the MAINFRDY flag.

The measure is performed on the main frequency (i.e. not limited to RC oscillator only), but if the main clock frequency source is the fast crystal oscillator, the restart of measuring is not needed because of the well known stability of crystal oscillators.

23.16.9 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 “PMC Write Protection Mode Register” .

- **PLLAEN: PLLA Control**

0: PLLA is disabled.

1: PLLA is enabled

2 up to 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 deactivated (PLLA also disabled if DIVA = 0).

750 up to 1500 = The PLLA Clock frequency is the PLLA input frequency multiplied by MULA + 1.

To change the PLLA frequency, please read Section 22.6.1 “Phase Lock Loop Programming” on page 430.

23.16.10PMC Master Clock Register

Name: PMC_MCKR

Address: 0x400E0430

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	PLLADIV2	–	–	–	–
7	6	5	4	3	2	1	0
–	PRES			–	–	CSS	

This register can only be written if the WPEN bit is cleared in "PMC Write Protection Mode Register" .

- **CSS: Master Clock Source Selection**

Value	Name	Description
0	SLOW_CLK	Slow Clock is selected
1	MAIN_CLK	Main Clock is selected
2	PLLA_CLK	PLLA Clock is selected

- **PRES: Processor Clock Prescaler**

Value	Name	Description
0	CLK_1	Selected clock
1	CLK_2	Selected clock divided by 2
2	CLK_4	Selected clock divided by 4
3	CLK_8	Selected clock divided by 8
4	CLK_16	Selected clock divided by 16
5	CLK_32	Selected clock divided by 32
6	CLK_64	Selected clock divided by 64
7	CLK_3	Selected clock divided by 3

- **PLLADIV2: PLLA Divisor by 2**

PLLADIV2	PLLA Clock Division
0	PLLA clock frequency is divided by 1.
1	PLLA clock frequency is divided by 2.

23.16.11PMC Programmable Clock Register

Name: PMC_PCKx

Address: 0x400E0440

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in “PMC Write Protection Mode Register” .

- **CSS: Master Clock Source Selection**

Value	Name	Description
0	SLOW_CLK	Slow Clock is selected
1	MAIN_CLK	Main Clock is selected
2	PLLA_CLK	PLLA Clock is selected
4	MCK	Master Clock is selected

- **PRES: Programmable Clock Prescaler**

Value	Name	Description
0	CLK_1	Selected clock
1	CLK_2	Selected clock divided by 2
2	CLK_4	Selected clock divided by 4
3	CLK_8	Selected clock divided by 8
4	CLK_16	Selected clock divided by 16
5	CLK_32	Selected clock divided by 32
6	CLK_64	Selected clock divided by 64

23.16.12PMC Interrupt Enable Register

Name: PMC_IER

Address: 0x400E0460

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	CFDEV	MOSCRCS	MOSCSELS
15	14	13	12	11	10	9	8
–	–	–	–	–	PCKRDY2	PCKRDY1	PCKRDY0
7	6	5	4	3	2	1	0
–	–	–	–	MCKRDY	–	LOCKA	MOSCXTS

- **MOSCXTS: Main Crystal Oscillator Status Interrupt Enable**
- **LOCKA: PLLA Lock Interrupt Enable**
- **MCKRDY: Master Clock Ready Interrupt Enable**
- **PCKRDYx: Programmable Clock Ready x Interrupt Enable**
- **MOSCSELS: Main Oscillator Selection Status Interrupt Enable**
- **MOSCRCS: Main On-Chip RC Status Interrupt Enable**
- **CFDEV: Clock Failure Detector Event Interrupt Enable**

23.16.13PMC Interrupt Disable Register

Name: PMC_IDR

Address: 0x400E0464

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	CFDEV	MOSCRCS	MOSCSELS
15	14	13	12	11	10	9	8
–	–	–	–	–	PCKRDY2	PCKRDY1	PCKRDY0
7	6	5	4	3	2	1	0
–	–	–	–	MCKRDY	–	LOCKA	MOSCXTS

- **MOSCXTS: Main Crystal Oscillator Status Interrupt Disable**
- **LOCKA: PLLA Lock Interrupt Disable**
- **MCKRDY: Master Clock Ready Interrupt Disable**
- **PCKRDYx: Programmable Clock Ready x Interrupt Disable**
- **MOSCSELS: Main Oscillator Selection Status Interrupt Disable**
- **MOSCRCS: Main On-Chip RC Status Interrupt Disable**
- **CFDEV: Clock Failure Detector Event Interrupt Disable**

23.16.14PMC Status Register

Name: PMC_SR

Address: 0x400E0468

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	FOS	CFDS	CFDEV	MOSCRCS	MOSCSELS

15	14	13	12	11	10	9	8
–	–	–	–	–	PCKRDY2	PCKRDY1	PCKRDY0

7	6	5	4	3	2	1	0
OSCSELS	–	–	–	MCKRDY	–	LOCKA	MOSCXTS

- **MOSCXTS: Main XTAL Oscillator Status**

0: Main XTAL oscillator is not stabilized.

1: Main XTAL oscillator is stabilized.

- **LOCKA: PLLA Lock Status**

0: PLLA is not locked

1: PLLA is locked.

- **MCKRDY: Master Clock Status**

0: Master Clock is not ready.

1: Master Clock is ready.

- **OSCSELS: Slow Clock Oscillator Selection**

0: Internal slow clock RC oscillator is selected.

1: External slow clock 32 kHz oscillator is selected.

- **PCKRDYx: Programmable Clock Ready Status**

0: Programmable Clock x is not ready.

1: Programmable Clock x is ready.

- **MOSCSELS: Main Oscillator Selection Status**

0: Selection is in progress.

1: Selection is done.

- **MOSCRCS: Main On-Chip RC Oscillator Status**

0: Main on-chip RC oscillator is not stabilized.

1: Main on-chip RC oscillator is stabilized.

- **CFDEV: Clock Failure Detector Event**

0: No clock failure detection of the fast crystal oscillator clock has occurred since the last read of PMC_SR.

1: At least one clock failure detection of the fast crystal oscillator clock has occurred since the last read of PMC_SR.

- **CFDS: Clock Failure Detector Status**

0: A clock failure of the fast crystal oscillator clock is not detected.

1: A clock failure of the fast crystal oscillator clock is detected.

- **FOS: Clock Failure Detector Fault Output Status**

0: The fault output of the clock failure detector is inactive.

1: The fault output of the clock failure detector is active.

23.16.15PMC Interrupt Mask Register

Name: PMC_IMR

Address: 0x400E046C

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	CFDEV	MOSCRCS	MOSCSELS
15	14	13	12	11	10	9	8
–	–	–	–	–	PCKRDY2	PCKRDY1	PCKRDY0
7	6	5	4	3	2	1	0
–	–	–	–	MCKRDY	–	LOCKA	MOSCXTS

- **MOSCXTS: Main Crystal Oscillator Status Interrupt Mask**
- **LOCKA: PLLA Lock Interrupt Mask**
- **MCKRDY: Master Clock Ready Interrupt Mask**
- **PCKRDYx: Programmable Clock Ready x Interrupt Mask**
- **MOSCSELS: Main Oscillator Selection Status Interrupt Mask**
- **MOSCRCS: Main On-Chip RC Status Interrupt Mask**
- **CFDEV: Clock Failure Detector Event Interrupt Mask**

23.16.16PMC 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 “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 Power Management Controller.

1: The corresponding wake-up input enables a fast restart signal to the Power Management Controller.

- **RTTAL: RTT Alarm Enable**

0: The RTT alarm has no effect on the Power Management Controller.

1: The RTT alarm enables a fast restart signal to the Power Management Controller.

- **RTCAL: RTC Alarm Enable**

0: The RTC alarm has no effect on the Power Management Controller.

1: The RTC alarm enables a fast restart signal to the Power Management Controller.

- **LPM: Low-power Mode**

0: The WaitForInterrupt (WFI) or the WaitForEvent (WFE) instruction of the processor makes the processor enter Sleep Mode.

1: The WaitForEvent (WFE) instruction of the processor makes the system to enter in Wait Mode.

- **FLPM: Flash Low-power Mode**

Value	Name	Description
0	FLASH_STANDBY	Flash is in Standby Mode when system enters Wait Mode
1	FLASH_DEEP_POWERDOWN	Flash is in deep-power-down mode when system enters Wait Mode
2	FLASH_IDLE	idle mode

23.16.17PMC 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 “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.

23.16.18PMC 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.

23.16.19PMC Write Protection Mode Register

Name: PMC_WPMR

Address: 0x400E04E4

Access: Read/Write

Reset: See Table 23-2

31	30	29	28	27	26	25	24
WPKEY							

23	22	21	20	19	18	17	16
WPKEY							

15	14	13	12	11	10	9	8
WPKEY							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WPEN

- **WPEN: Write 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 23.15 ”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.

23.16.20PMC Write Protection Status Register

Name: PMC_WPSR

Address: 0x400E04E8

Access: Read-only

Reset: See Table 23-2

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

- **WPVS: Write 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.

23.16.21PMC 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
SEL24	CAL24						

15	14	13	12	11	10	9	8
SEL16	CAL16						

7	6	5	4	3	2	1	0
SEL8	CAL8						

This register can only be written if the WPEN bit is cleared in “PMC Write Protection Mode 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: Default value stored in Flash memory.

1: Value written by user in CAL8 field of this register.

- **CAL16: RC Oscillator Calibration bits for 16 MHz**

Calibration bits applied to the RC Oscillator when SEL16 is set.

- **SEL16: Selection of RC Oscillator Calibration bits for 16 MHz**

0: Factory determined value stored in Flash memory.

1: Value written by user in CAL16 field of this register.

- **CAL24: RC Oscillator Calibration bits for 24 MHz**

Calibration bits applied to the RC Oscillator when SEL24 is set.

- **SEL24: Selection of RC Oscillator Calibration bits for 24 MHz**

0: Factory determined value stored in Flash memory.

1: Value written by user in CAL24 field of this register.

23.16.22PLL Maximum Multiplier Value Register

Name: PMC_PMMR

Address: 0x400E0530

Access: Read/Write

Reset: 0x00000000

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

This register can only be written if the WPEN bit is cleared in "PMC Write Protection Mode Register" .

- **PLLA_MMAX: PLLA Maximum Allowed Multiplier Value**

Defines the maximum value of multiplication factor that can be sent to PLLA. Any value of the MULA field (see "PMC Clock Generator PLLA Register") above PLLA_MMAX is saturated to PLLA_MMAX. PLLA_MMAX write operation is cancelled in the following cases:

- The value of MULA is currently saturated by PLLA_MMAX
- The user is trying to write a value of PLLA_MMAX that is smaller than the current value of MULA

24. Chip Identifier (CHIPID)

24.1 Description

Chip Identifier (CHIPID) registers permit recognition of the device and its revision. These registers provide the sizes and types of the on-chip memories, as well as the set of embedded peripherals.

Two chip identifier registers are embedded: CHIPID_CIDR (Chip ID Register) and CHIPID_EXID (Extension ID). Both registers contain a hard-wired value that is read-only. The first register contains the following fields:

- EXT - shows the use of the extension identifier register
- NVPTYP and NVPSIZ - identifies the type of embedded non-volatile memory and its size
- ARCH - identifies the set of embedded peripherals
- SRAMSIZ - indicates the size of the embedded SRAM
- EPROC - indicates the embedded ARM processor
- VERSION - gives the revision of the silicon

The second register is device-dependent and reads 0 if the bit EXT is 0.

24.2 Embedded Characteristics

- Chip ID Registers
  - Identification of the Device Revision, Sizes of the Embedded Memories, Set of Peripherals, Embedded Processor

Table 24-1. SAM G51 Chip IDs Register

Chip Name	CHIPID_CIDR	CHIPID_EXID
SAM G51G18	0x243B_09E0	0x0
SAM G51N18	0x243B_09E8	0x0

24.3 Chip Identifier (CHIPID) User Interface

Table 24-2. Register Mapping

Offset	Register	Name	Access	Reset
0x0	Chip ID Register	CHIPID_CIDR	Read-only	–
0x4	Chip ID Extension Register	CHIPID_EXID	Read-only	–

24.3.1 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	–	Reserved
9	256K	256 Kbytes
10	512K	512 Kbytes
11	–	Reserved
12	1024K	1024 Kbytes

Value	Name	Description
13	–	Reserved
14	2048K	2048 Kbytes
15	–	Reserved

- **NVPSIZ2: Second Nonvolatile Program Memory Size**

Value	Name	Description
0	NONE	None
1	8K	8 Kbytes
2	16K	16 Kbytes
3	32K	32 Kbytes
4	–	Reserved
5	64K	64 Kbytes
6	–	Reserved
7	128K	128 Kbytes
8	–	Reserved
9	256K	256 Kbytes
10	512K	512 Kbytes
11	–	Reserved
12	1024K	1024 Kbytes
13	–	Reserved
14	2048K	2048 Kbytes
15	–	Reserved

- **SRAMSIZ: Internal SRAM Size**

Value	Name	Description
0	48K	48 Kbytes
1	192K	192 Kbytes
2	2K	2 Kbytes
3	6K	6 Kbytes
4	24K	24 Kbytes
5	4K	4 Kbytes
6	80K	80 Kbytes
7	160K	160 Kbytes
8	8K	8 Kbytes
9	16K	16 Kbytes
10	32K	32 Kbytes
11	64K	64 Kbytes
12	128K	128 Kbytes

Value	Name	Description
13	256K	256 Kbytes
14	96K	96 Kbytes
15	512K	512 Kbytes

- **ARCH: Architecture Identifier**

Value	Name	Description
0x43	SAM G51	SAM G51

- **NVPTYP: Nonvolatile Program Memory Type**

Value	Name	Description
0	ROM	ROM
1	ROMLESS	ROMless or on-chip Flash
4	SRAM	SRAM emulating ROM
2	FLASH	Embedded Flash Memory
3	ROM_FLASH	ROM and Embedded Flash Memory • NVPSIZ is ROM size • NVPSIZ2 is Flash size

- **EXT: Extension Flag**

0 = Chip ID has a single register definition without extension.

1 = An extended Chip ID exists.

24.3.2 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**

Reads 0 if the EXT bit in CHIPID_CIDR is 0.

25. Parallel Input/Output Controller (PIO)

25.1 Description

The Parallel Input/Output Controller (PIO) manages up to 32 fully programmable input/output lines. Each I/O line may be dedicated as a general-purpose I/O or be assigned to a function of an embedded peripheral. This assures effective optimization of the pins of the product.

Each I/O line is associated with a bit number in all of the 32-bit registers of the 32-bit wide user interface.

Each I/O line of the PIO Controller features:

- An input change interrupt enabling level change detection on any I/O line.
- Additional Interrupt modes enabling rising edge, falling edge, low-level or high-level detection on any I/O line.
- A glitch filter providing rejection of glitches lower than one-half of PIO clock cycle.
- A debouncing filter providing rejection of unwanted pulses from key or push button operations.
- Multi-drive capability similar to an open drain I/O line.
- Control of the pull-up and pull-down of the I/O line.
- Input visibility and output control.

The PIO Controller also features a synchronous output providing up to 32 bits of data output in a single write operation.

25.2 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
- Write Protect Registers
- Programmable Schmitt Trigger Inputs

25.3 Block Diagram

Figure 25-1. Block Diagram

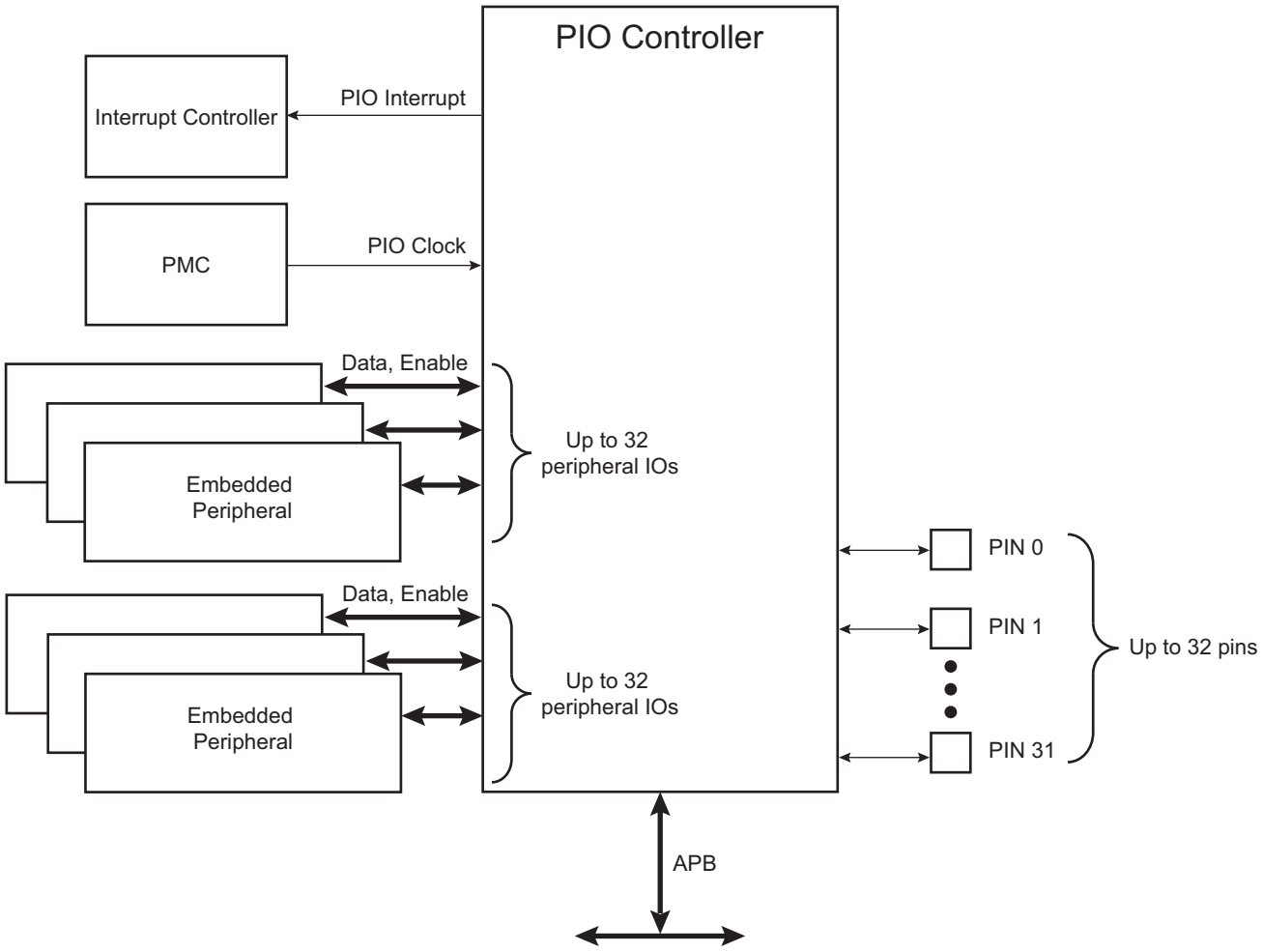
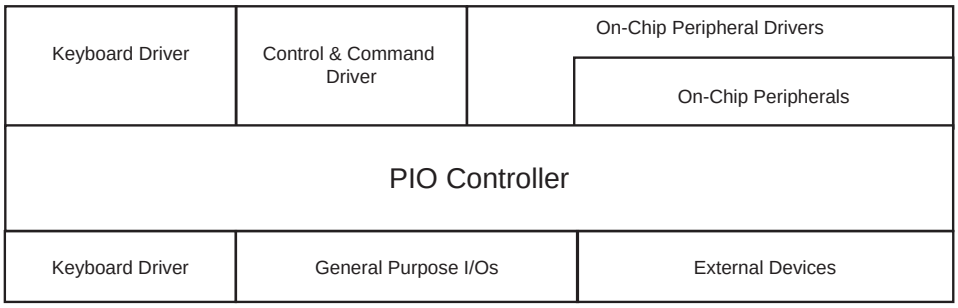


Figure 25-2. Application Block Diagram



25.4 Product Dependencies

25.4.1 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.

25.4.2 External Interrupt Lines

The interrupt signals FIQ and IRQ0 to IRQn are generally multiplexed through the PIO Controllers. However, it is not necessary to assign the I/O line to the interrupt function as the PIO Controller has no effect on inputs and the interrupt lines (FIQ or IRQs) are used only as inputs.

25.4.3 Power Management

The Power Management Controller controls the PIO Controller clock in order to save power. Writing any of the registers of the user interface does not require the PIO Controller clock to be enabled. This means that the configuration of the I/O lines does not require the PIO Controller clock to be enabled.

However, when the clock is disabled, not all of the features of the PIO Controller are available, including glitch filtering. Note that the input change interrupt, 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 PIO clock is disabled by default.

The user must configure the Power Management Controller before any access to the input line information.

25.4.4 Interrupt Generation

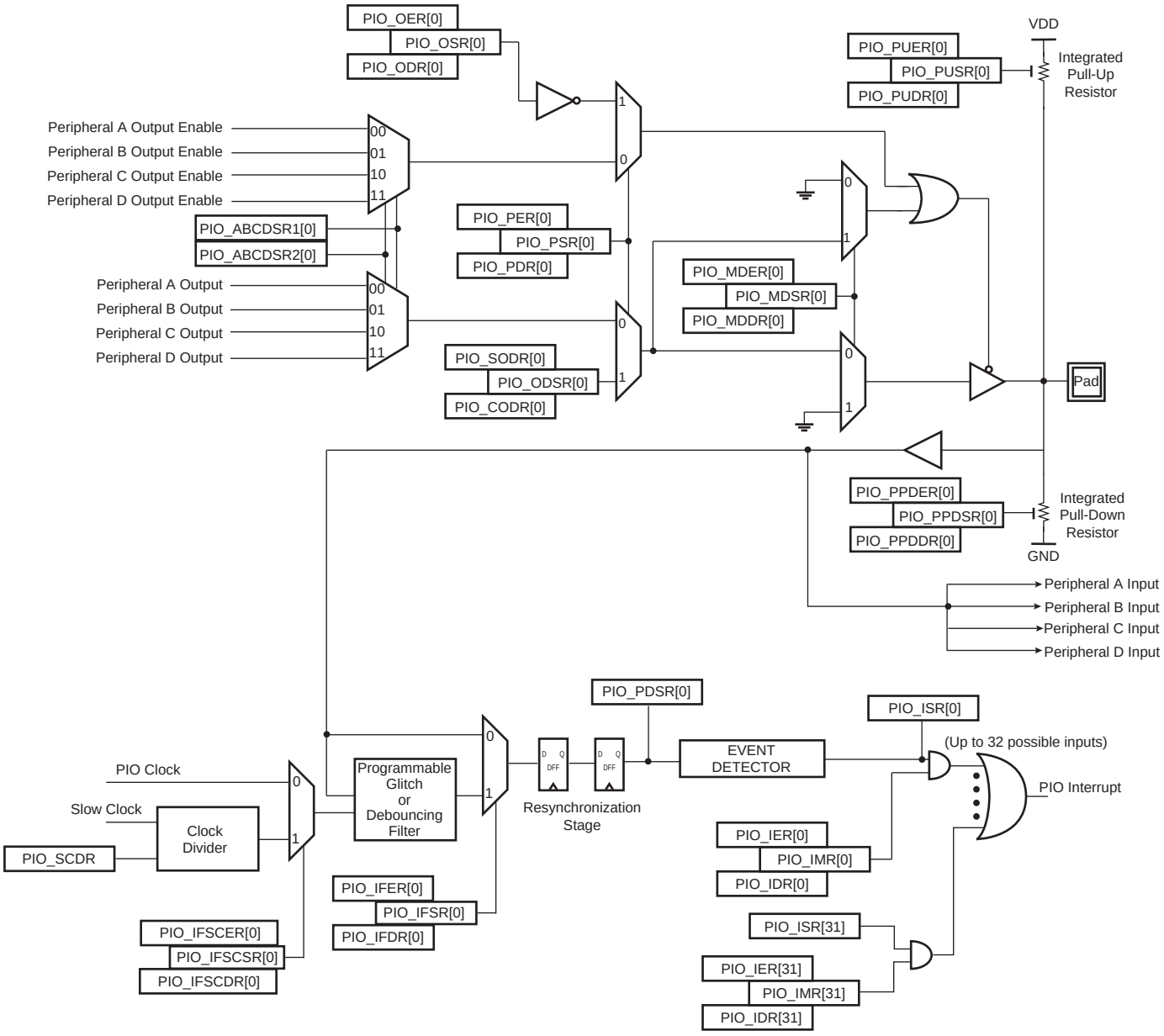
For interrupt handling, the PIO Controllers are considered as user peripherals. This means that the PIO Controller interrupt lines are connected among the interrupt sources. Refer to the PIO Controller peripheral identifier in the product description to identify the interrupt sources dedicated to the PIO Controllers. Using the PIO Controller requires the Interrupt Controller to be programmed first.

The PIO Controller interrupt can be generated only if the PIO Controller clock is enabled.

25.5 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 25-3. In this description each signal shown represents one of up to 32 possible indexes.

Figure 25-3. I/O Line Control Logic



25.5.1 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, all of the pull-ups are enabled, i.e. PIO_PUSR resets at the value 0x0, and all the pull-downs are disabled, i.e. PIO_PPDSR resets at the value 0xFFFFFFFF.

25.5.2 I/O Line or Peripheral Function Selection

When a pin is multiplexed with one or two peripheral functions, the selection is controlled with the Enable register (PIO_PER) and the Disable register (PIO_PDR). The Status register (PIO_PSR) is the result of the set and clear registers and indicates whether the pin is controlled by the corresponding peripheral or by the PIO Controller. A value of zero indicates that the pin is controlled by the corresponding on-chip peripheral selected in the ABCD Select registers (PIO_ABCDSR1 and PIO_ABCDSR2). A value of one indicates the pin is controlled by the PIO Controller.

If a pin is used as a general-purpose I/O line (not multiplexed with an on-chip peripheral), PIO_PER and PIO_PDR have no effect and PIO_PSR returns a one for the corresponding bit.

After reset, the I/O lines are controlled by the PIO Controller, i.e. PIO_PSR resets at one. However, in some events, it is important that PIO lines are controlled by the peripheral (as in the case of memory chip select lines that must be driven inactive after reset, or for address lines that must be driven low for booting out of an external memory). Thus, the reset value of PIO_PSR is defined at the product level and depends on the multiplexing of the device.

25.5.3 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 zero in PIO_ABCDSR2 means peripheral D is selected.

Note that multiplexing of peripheral lines A, B, C and D only affects the output line. The peripheral input lines are always connected to the pin input.

Writing in PIO_ABCDSR1 and PIO_ABCDSR2 manages the multiplexing regardless of the configuration of the pin. However, assignment of a pin to a peripheral function requires a write in 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.

25.5.4 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.

25.5.5 Synchronous Data Output

Clearing one or more PIO line(s) and setting another one or more PIO line(s) synchronously cannot be done by using PIO_SODR and PIO_CODR registers. It requires two successive write operations into two different registers. To overcome this, the PIO Controller offers a direct control of PIO outputs by single write access to PIO_ODSR. 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.

25.5.6 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.

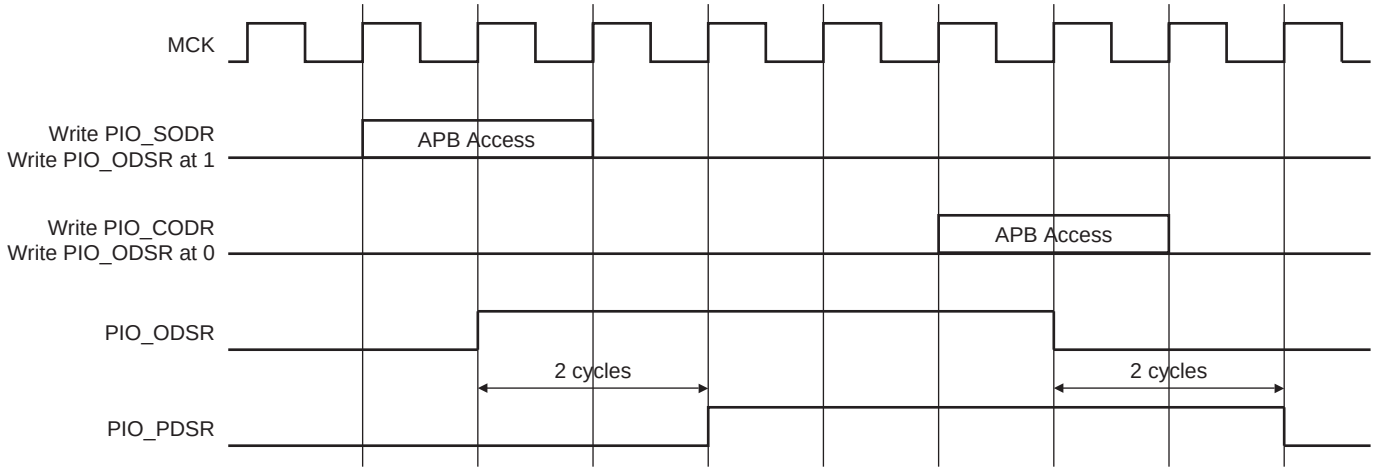
The multi-drive feature is controlled by the Multi-driver Enable register (PIO_MDER) and the Multi-driver Disable register (PIO_MDDR). The multi-drive can be selected whether the I/O line is controlled by the PIO Controller or assigned to a peripheral function. The Multi-driver Status register (PIO_MDSR) 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.

25.5.7 Output Line Timings

Figure 25-4 shows how the outputs are driven either by writing PIO_SODR or PIO_CODR, or by directly writing PIO_ODSR. This last case is valid only if the corresponding bit in PIO_OWSR is set. Figure 25-4 also shows when the feedback in the Pin Data Status register (PIO_PDSR) is available.

Figure 25-4. Output Line Timings



25.5.8 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.

25.5.9 Input Glitch and Debouncing Filters

Optional input glitch and debouncing filters are independently programmable on each I/O line.

The glitch filter can filter a glitch with a duration of less than 1/2 master clock (MCK) and the debouncing filter can filter a pulse of less than 1/2 period of a programmable divided slow clock.

The selection between glitch filtering or debounce filtering is done by writing in the 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 register 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 performed by writing in the DIV field of the Slow Clock Divider register (PIO_SCDR.

Tdiv_slclk = ((DIV+1)*2).Tslow_clock

When the glitch or debouncing filter is enabled, a glitch or pulse with a duration of less than 1/2 selected clock cycle (selected clock represents MCK or divided slow clock depending on PIO_IFSCDR and PIO_IFSCER programming) is automatically rejected, while a pulse with a duration of one selected clock (MCK 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 25-5 and Figure 25-6.

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 PIO Controller clock is enabled.

Figure 25-5. Input Glitch Filter Timing

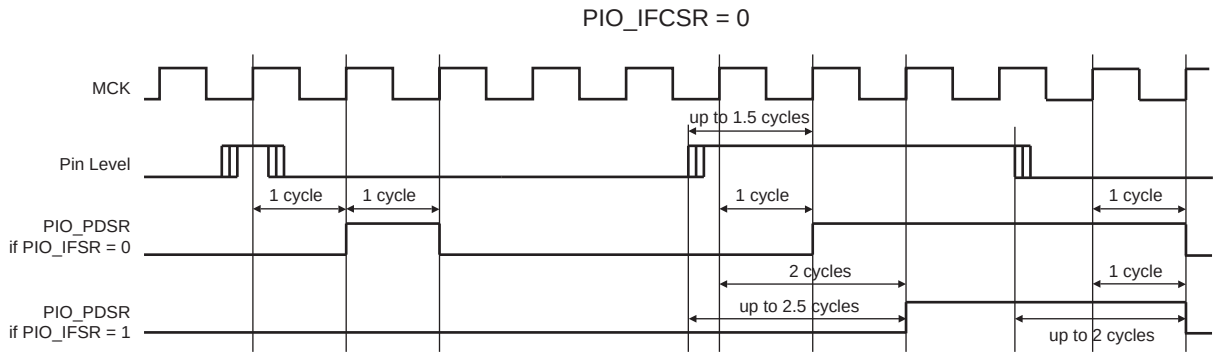
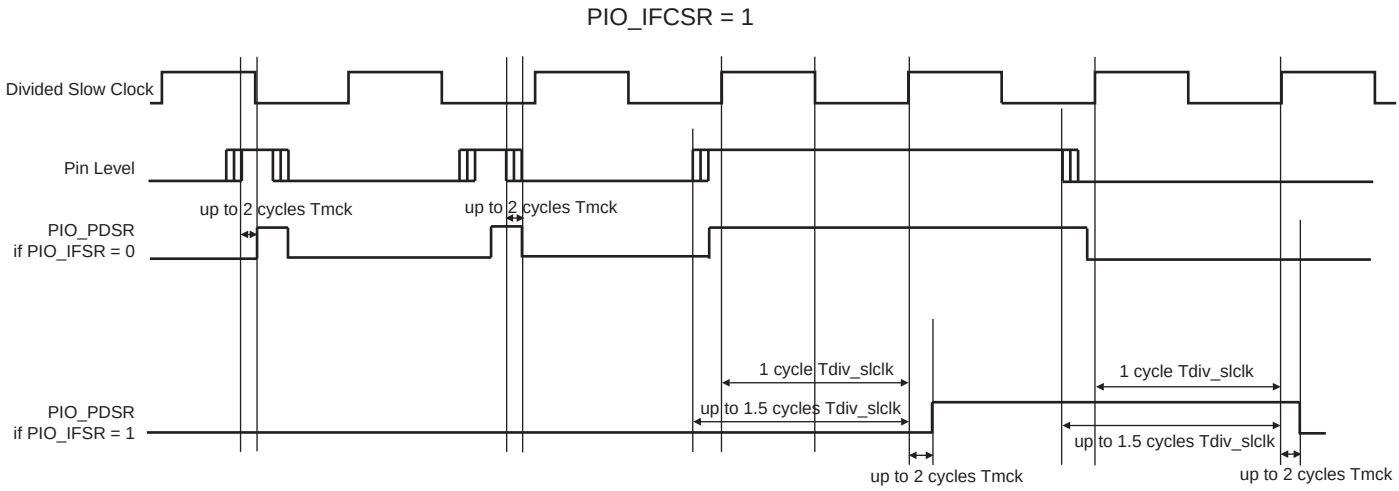


Figure 25-6. Input Debouncing Filter Timing



25.5.10 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 PIO Controller clock must be enabled. The Input Change interrupt is available regardless of the configuration of the I/O line, i.e. configured as an input only, controlled by the PIO Controller or assigned to a peripheral function.

By default, the interrupt can be generated at any time an edge is detected on the input.

Some additional interrupt modes can be enabled/disabled by writing in the 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

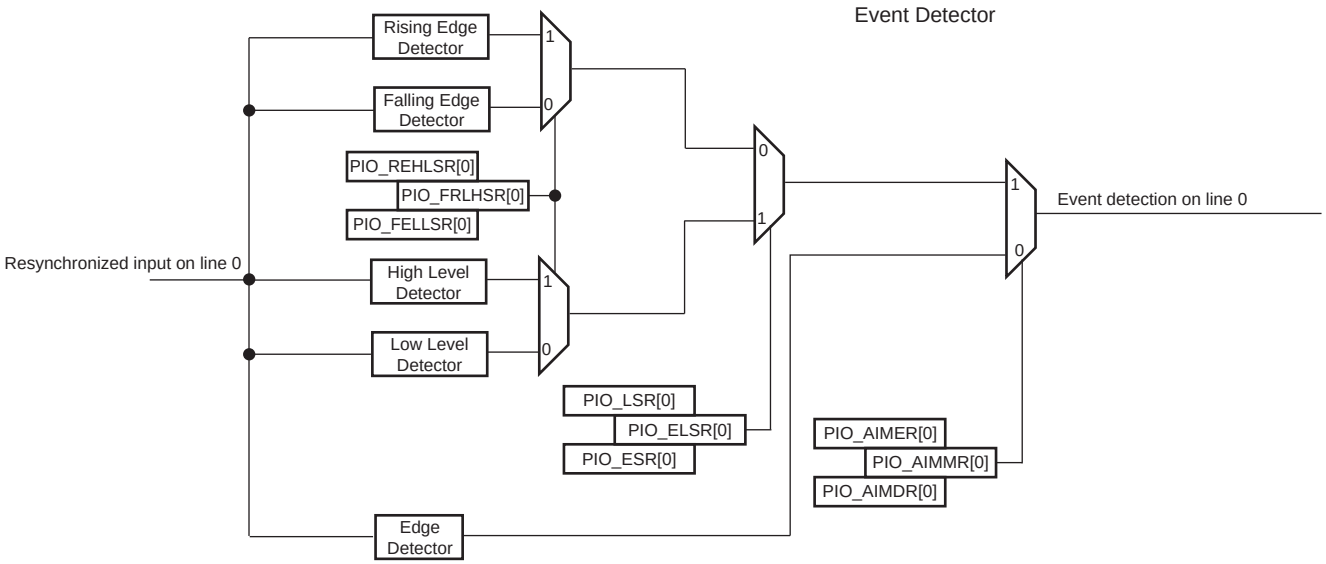
In order to select an additional interrupt mode:

- 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 , 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 25-7. Event Detector on Input Lines (Figure Represents Line 0)



25.5.10.1 Example

If generating an interrupt is required on the lines below, the configuration required is described in Section 25.5.10.2 "Interrupt Mode Configuration", Section 25.5.10.3 "Edge or Level Detection Configuration" and Section 25.5.10.4 "Falling/Rising Edge or Low/High-Level Detection Configuration":

- Rising edge on PIO line 0
- Falling edge on PIO line 1
- Rising edge on PIO line 2
- Low-level on PIO line 3
- High-level on PIO line 4
- High-level on PIO line 5
- Falling edge on PIO line 6
- Rising edge on PIO line 7
- Any edge on the other lines

the configuration required is described below.

25.5.10.2 Interrupt Mode Configuration

All the interrupt sources are enabled by writing 32'hFFFF_FFFF in PIO_IER.

Then the additional interrupt mode is enabled for lines 0 to 7 by writing 32'h0000_00FF in PIO_AIMER.

25.5.10.3 Edge or Level Detection Configuration

Lines 3, 4 and 5 are configured in level detection by writing 32'h0000_0038 in PIO_LSR.

The other lines are configured in edge detection by default, if they have not been previously configured. Otherwise, lines 0, 1, 2, 6 and 7 must be configured in edge detection by writing 32'h0000_00C7 in PIO_ESR.

25.5.10.4 Falling/Rising Edge or Low/High-Level Detection Configuration

Lines 0, 2, 4, 5 and 7 are configured in rising edge or high-level detection by writing 32'h0000_00B5 in PIO_REHLSR.

The other lines are configured in falling edge or low-level detection by default if they have not been previously configured. Otherwise, lines 1, 3 and 6 must be configured in falling edge/low-level detection by writing 32'h0000_004A in PIO_FELLSR.

Figure 25-8. Input Change Interrupt Timings When No Additional Interrupt Modes

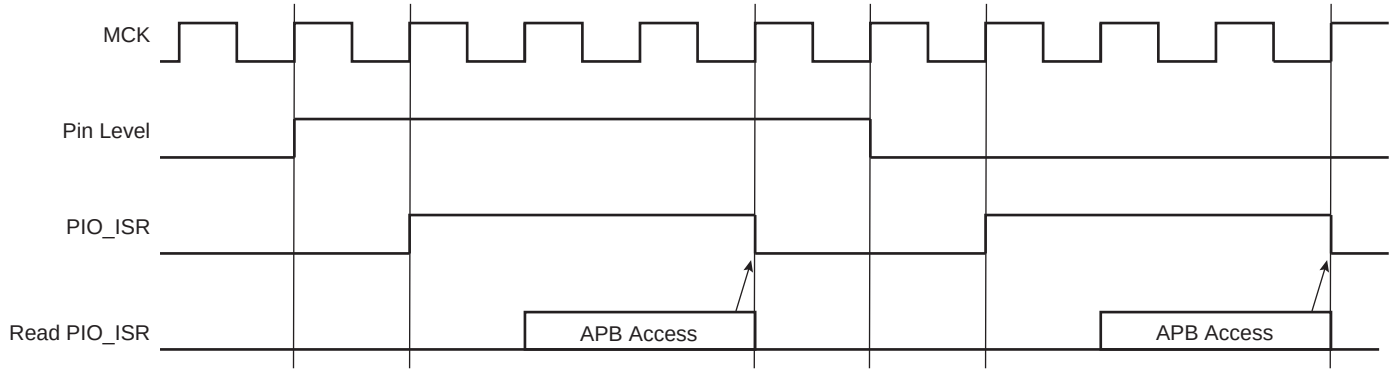


Figure 25-9.

25.5.11 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.

25.5.12 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" on page 488
- "PIO Disable Register" on page 488
- "PIO Output Enable Register" on page 490
- "PIO Output Disable Register" on page 490
- "PIO Input Filter Enable Register" on page 492
- "PIO Input Filter Disable Register" on page 492
- "PIO Multi-driver Enable Register" on page 500
- "PIO Multi-driver Disable Register" on page 500
- "PIO Pull-Up Disable Register" on page 502
- "PIO Pull-Up Enable Register" on page 502
- "PIO Peripheral ABCD Select Register 1" on page 504
- "PIO Peripheral ABCD Select Register 2" on page 505
- "PIO Output Write Enable Register" on page 511
- "PIO Output Write Disable Register" on page 511
- "PIO Pad Pull-Down Disable Register" on page 509
- "PIO Pad Pull-Down Status Register" on page 510

25.6 I/O Lines Programming Example

The programming example shown in Table 25-1 is used to obtain the following configuration.

- 4-bit output port on I/O lines 0 to 3, (should be written in a single write operation), open-drain, with pull-up resistor
- Four output signals on I/O lines 4 to 7 (to drive LEDs for example), driven high and low, no pull-up resistor, no pull-down resistor
- Four input signals on I/O lines 8 to 11 (to read push-button states for example), with pull-up resistors, glitch filters and input change interrupts
- Four input signals on I/O line 12 to 15 to read an external device status (polled, thus no input change interrupt), no pull-up resistor, no glitch filter
- I/O lines 16 to 19 assigned to peripheral A functions with pull-up resistor
- I/O lines 20 to 23 assigned to peripheral B functions with pull-down resistor
- I/O line 24 to 27 assigned to peripheral C with Input Change Interrupt, no pull-up resistor and no pull-down resistor
- I/O line 28 to 31 assigned to peripheral D, no pull-up resistor and no pull-down resistor

Table 25-1. Programming Example

Register	Value to be Written
PIO_PER	0x0000_FFFF
PIO_PDR	0xFFFF_0000
PIO_OER	0x0000_00FF
PIO_ODR	0xFFFF_FF00
PIO_IFER	0x0000_0F00
PIO_IFDR	0xFFFF_F0FF
PIO_SODR	0x0000_0000
PIO_CODR	0x0FFF_FFFF
PIO_IER	0x0F00_0F00
PIO_IDR	0xF0FF_F0FF
PIO_MDER	0x0000_000F
PIO_MDDR	0xFFFF_FFF0
PIO_PUDR	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

25.7 Parallel Input/Output Controller (PIO) User Interface

Each I/O line controlled by the PIO Controller is associated with a bit in each of the PIO Controller User Interface registers. Each register is 32 bits wide. If a parallel I/O line is not defined, writing to the corresponding bits has no effect. Undefined bits read zero. If the I/O line is notmultiplexed with any peripheral, the I/O line is controlled by the PIO Controller and PIO_PSR returns one systematically.

Table 25-2. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	PIO Enable Register	PIO_PER	Write-only	–
0x0004	PIO Disable Register	PIO_PDR	Write-only	–
0x0008	PIO Status Register	PIO_PSR	Read-only	(1)
0x000C	Reserved	–	–	–
0x0010	Output Enable Register	PIO_OER	Write-only	–
0x0014	Output Disable Register	PIO_ODR	Write-only	–
0x0018	Output Status Register	PIO_OSR	Read-only	0x0000 0000
0x001C	Reserved	–	–	–
0x0020	Glitch Input Filter Enable Register	PIO_IFER	Write-only	–
0x0024	Glitch Input Filter Disable Register	PIO_IFDR	Write-only	–
0x0028	Glitch Input Filter Status Register	PIO_IFSR	Read-only	0x0000 0000
0x002C	Reserved	–	–	–
0x0030	Set Output Data Register	PIO_SODR	Write-only	–
0x0034	Clear Output Data Register	PIO_CODR	Write-only	
0x0038	Output Data Status Register	PIO_ODSR	Read-only or(2) Read/Write	–
0x003C	Pin Data Status Register	PIO_PDSR	Read-only	(3)
0x0040	Interrupt Enable Register	PIO_IER	Write-only	–
0x0044	Interrupt Disable Register	PIO_IDR	Write-only	–
0x0048	Interrupt Mask Register	PIO_IMR	Read-only	0x00000000
0x004C	Interrupt Status Register(4)	PIO_ISR	Read-only	0x00000000
0x0050	Multi-driver Enable Register	PIO_MDER	Write-only	–
0x0054	Multi-driver Disable Register	PIO_MDDR	Write-only	–
0x0058	Multi-driver Status Register	PIO_MDSR	Read-only	0x00000000
0x005C	Reserved	–	–	–
0x0060	Pull-up Disable Register	PIO_PUDR	Write-only	–
0x0064	Pull-up Enable Register	PIO_PUER	Write-only	–
0x0068	Pad Pull-up Status Register	PIO_PUSR	Read-only	(1)
0x006C	Reserved	–	–	–

Table 25-2. Register Mapping (Continued)

Offset	Register	Name	Access	Reset
0x0070	Peripheral Select Register 1	PIO_ABCDSR1	Read/Write	0x00000000
0x0074	Peripheral Select Register 2	PIO_ABCDSR2	Read/Write	0x00000000
0x0078 to 0x007C	Reserved	–	–	–
0x0080	Input Filter Slow Clock Disable Register	PIO_IFSCDR	Write-only	–
0x0084	Input Filter Slow Clock Enable Register	PIO_IFSCER	Write-only	–
0x0088	Input Filter Slow Clock Status Register	PIO_IFSCSR	Read-only	0x00000000
0x008C	Slow Clock Divider Debouncing Register	PIO_SCDR	Read/Write	0x00000000
0x0090	Pad Pull-down Disable Register	PIO_PPDDR	Write-only	–
0x0094	Pad Pull-down Enable Register	PIO_PPDER	Write-only	–
0x0098	Pad Pull-down Status Register	PIO_PPDSR	Read-only	(1)
0x009C	Reserved	–	–	–
0x00A0	Output Write Enable	PIO_OWER	Write-only	–
0x00A4	Output Write Disable	PIO_OWDR	Write-only	–
0x00A8	Output Write Status Register	PIO_OWSR	Read-only	0x00000000
0x00AC	Reserved	–	–	–
0x00B0	Additional Interrupt Modes Enable Register	PIO_AIMER	Write-only	–
0x00B4	Additional Interrupt Modes Disable Register	PIO_AIMDR	Write-only	–
0x00B8	Additional Interrupt Modes Mask Register	PIO_AIMMR	Read-only	0x00000000
0x00BC	Reserved	–	–	–
0x00C0	Edge Select Register	PIO_ESR	Write-only	–
0x00C4	Level Select Register	PIO_LSR	Write-only	–
0x00C8	Edge/Level Status Register	PIO_ELSR	Read-only	0x00000000
0x00CC	Reserved	–	–	–
0x00D0	Falling Edge/Low-Level Select Register	PIO_FELLSR	Write-only	–
0x00D4	Rising Edge/ High-Level Select Register	PIO_REHLSR	Write-only	–
0x00D8	Fall/Rise - Low/High Status Register	PIO_FRLHSR	Read-only	0x00000000
0x00DC	Reserved	–	–	–
0x00E0	Reserved	–	–	–
0x00E4	Write Protection Mode Register	PIO_WPMR	Read/Write	0x0
0x00E8	Write Protection Status Register	PIO_WPSR	Read-only	0x0
0x00EC to 0x00F8	Reserved	–	–	–
0x0100	Schmitt Trigger Register	PIO_SCHMITT	Read/Write	0x00000000
0x0104-0x010C	Reserved	–	–	–

Table 25-2. Register Mapping (Continued)

Offset	Register	Name	Access	Reset
0x0110	Reserved	–	–	–
0x0114-0x011C	Reserved	–	–	–
0x0120 to 0x014C	Reserved	–	–	–

Notes:
1. Reset value depends on the product implementation.
2. PIO_ODSR is Read-only or Read/Write depending on PIO_OWSR I/O lines.
3. Reset value of PIO_PDSR depends on the level of the I/O lines. Reading the I/O line levels requires the clock of the PIO Controller to be enabled, otherwise PIO_PDSR reads the levels present on the I/O line at the time the clock was disabled.
4. PIO_ISR is reset at 0x0. However, the first read of the register may read a different value as input changes may have occurred.
5. If an offset is not listed in the table it must be considered as reserved.

25.7.1 PIO Enable Register

Name: PIO_PER

Address: 0x400E0E00 (PIOA), 0x400E1000 (PIOB)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “PIO Write 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).

25.7.2 PIO Disable Register

Name: PIO_PDR

Address: 0x400E0E04 (PIOA), 0x400E1004 (PIOB)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “PIO Write 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).

25.7.3 PIO Status Register

Name: PIO_PSR

Address: 0x400E0E08 (PIOA), 0x400E1008 (PIOB)

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).

25.7.4 PIO Output Enable Register

Name: PIO_OER

Address: 0x400E0E10 (PIOA), 0x400E1010 (PIOB)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “PIO Write Protection Mode Register” .

- **P0-P31: Output Enable**

0: No effect.

1: Enables the output on the I/O line.

25.7.5 PIO Output Disable Register

Name: PIO_ODR

Address: 0x400E0E14 (PIOA), 0x400E1014 (PIOB)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “PIO Write Protection Mode Register” .

- **P0-P31: Output Disable**

0: No effect.

1: Disables the output on the I/O line.

25.7.6 PIO Output Status Register

Name: PIO_OSR

Address: 0x400E0E18 (PIOA), 0x400E1018 (PIOB)

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.

25.7.7 PIO Input Filter Enable Register

Name: PIO_IFER

Address: 0x400E0E20 (PIOA), 0x400E1020 (PIOB)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “PIO Write Protection Mode Register” .

- **P0-P31: Input Filter Enable**

0: No effect.

1: Enables the input glitch filter on the I/O line.

25.7.8 PIO Input Filter Disable Register

Name: PIO_IFDR

Address: 0x400E0E24 (PIOA), 0x400E1024 (PIOB)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “PIO Write Protection Mode Register” .

- **P0-P31: Input Filter Disable**

0: No effect.

1: Disables the input glitch filter on the I/O line.

25.7.9 PIO Input Filter Status Register

Name: PIO_IFSR

Address: 0x400E0E28 (PIOA), 0x400E1028 (PIOB)

Access: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Input Filer Status**

0: The input glitch filter is disabled on the I/O line.

1: The input glitch filter is enabled on the I/O line.

25.7.10 PIO Set Output Data Register

Name: PIO_SODR

Address: 0x400E0E30 (PIOA), 0x400E1030 (PIOB)

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.

25.7.11 PIO Clear Output Data Register

Name: PIO_CODR

Address: 0x400E0E34 (PIOA), 0x400E1034 (PIOB)

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.

25.7.12 PIO Output Data Status Register

Name: PIO_ODSR

Address: 0x400E0E38 (PIOA), 0x400E1038 (PIOB)

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.

25.7.13 PIO Pin Data Status Register

Name: PIO_PDSR

Address: 0x400E0E3C (PIOA), 0x400E103C (PIOB)

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.

25.7.14 PIO Interrupt Enable Register

Name: PIO_IER

Address: 0x400E0E40 (PIOA), 0x400E1040 (PIOB)

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.

25.7.15 PIO Interrupt Disable Register

Name: PIO_IDR

Address: 0x400E0E44 (PIOA), 0x400E1044 (PIOB)

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.

25.7.16 PIO Interrupt Mask Register

Name: PIO_IMR

Address: 0x400E0E48 (PIOA), 0x400E1048 (PIOB)

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.

25.7.17 PIO Interrupt Status Register

Name: PIO_ISR

Address: 0x400E0E4C (PIOA), 0x400E104C (PIOB)

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.

25.7.18 PIO Multi-driver Enable Register

Name: PIO_MDER

Address: 0x400E0E50 (PIOA), 0x400E1050 (PIOB)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “PIO Write Protection Mode Register” .

- **P0-P31: Multi-Drive Enable**

0: No effect.

1: Enables multi-drive on the I/O line.

25.7.19 PIO Multi-driver Disable Register

Name: PIO_MDDR

Address: 0x400E0E54 (PIOA), 0x400E1054 (PIOB)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “PIO Write Protection Mode Register” .

- **P0-P31: Multi-Drive Disable**

0: No effect.

1: Disables multi-drive on the I/O line.

25.7.20 PIO Multi-driver Status Register

Name: PIO_MDSR

Address: 0x400E0E58 (PIOA), 0x400E1058 (PIOB)

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.

25.7.21 PIO Pull-Up Disable Register

Name: PIO_PUDR

Address: 0x400E0E60 (PIOA), 0x400E1060 (PIOB)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “PIO Write Protection Mode Register” .

- **P0-P31: Pull-Up Disable**

0: No effect.

1: Disables the pull-up resistor on the I/O line.

25.7.22 PIO Pull-Up Enable Register

Name: PIO_PUER

Address: 0x400E0E64 (PIOA), 0x400E1064 (PIOB)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “PIO Write Protection Mode Register” .

- **P0-P31: Pull-Up Enable**

0: No effect.

1: Enables the pull-up resistor on the I/O line.

25.7.23 PIO Pull-Up Status Register

Name: PIO_PUSR

Address: 0x400E0E68 (PIOA), 0x400E1068 (PIOB)

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.

25.7.24 PIO Peripheral ABCD Select Register 1

Name: PIO_ABCDSR1

Access: Read/Write

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “PIO Write 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.

25.7.25 PIO Peripheral ABCD Select Register 2

Name: PIO_ABCDSR2

Access: Read/Write

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “PIO Write 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.

25.7.26 PIO Input Filter Slow Clock Disable Register

Name: PIO_IFSCDR

Address: 0x400E0E80 (PIOA), 0x400E1080 (PIOB)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: PIO Clock Glitch Filtering Select**

0: No effect.

1: The glitch filter is able to filter glitches with a duration < Tmck/2.

25.7.27 PIO Input Filter Slow Clock Enable Register

Name: PIO_IFSCER

Address: 0x400E0E84 (PIOA), 0x400E1084 (PIOB)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Debouncing Filtering Select**

0: No effect.

1: The debouncing filter is able to filter pulses with a duration < Tdiv_slclk/2.

25.7.28 PIO Input Filter Slow Clock Status Register

Name: PIO_IFSCSR

Address: 0x400E0E88 (PIOA), 0x400E1088 (PIOB)

Access: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Glitch or Debouncing Filter Selection Status**

0: The glitch filter is able to filter glitches with a duration < Tmck2.

1: The debouncing filter is able to filter pulses with a duration < Tdiv_slclk/2.

25.7.29 PIO Slow Clock Divider Debouncing Register

Name: PIO_SCDR

Address: 0x400E0E8C (PIOA), 0x400E108C (PIOB)

Access: Read/Write

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

- **DIV: Slow Clock Divider Selection for Debouncing**

Tdiv_slclk = 2*(DIV+1)*Tslow_clock.

25.7.30 PIO Pad Pull-Down Disable Register

Name: PIO_PPDDR

Address: 0x400E0E90 (PIOA), 0x400E1090 (PIOB)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “PIO Write Protection Mode Register” .

- **P0-P31: Pull-Down Disable**

0: No effect.

1: Disables the pull-down resistor on the I/O line.

25.7.31 PIO Pad Pull-Down Enable Register

Name: PIO_PPDER

Address: 0x400E0E94 (PIOA), 0x400E1094 (PIOB)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “PIO Write Protection Mode Register” .

- **P0-P31: Pull-Down Enable**

0: No effect.

1: Enables the pull-down resistor on the I/O line.

25.7.32 PIO Pad Pull-Down Status Register

Name: PIO_PPDSR

Address: 0x400E0E98 (PIOA), 0x400E1098 (PIOB)

Access: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “PIO Write Protection Mode Register” .

- **P0-P31: Pull-Down Status**

0: Pull-down resistor is enabled on the I/O line.

1: Pull-down resistor is disabled on the I/O line.

25.7.33 PIO Output Write Enable Register

Name: PIO_OWER

Address: 0x400E0EA0 (PIOA), 0x400E10A0 (PIOB)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “PIO Write Protection Mode Register” .

- **P0-P31: Output Write Enable**

0: No effect.

1: Enables writing PIO_ODSR for the I/O line.

25.7.34 PIO Output Write Disable Register

Name: PIO_OWDR

Address: 0x400E0EA4 (PIOA), 0x400E10A4 (PIOB)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in “PIO Write Protection Mode Register” .

- **P0-P31: Output Write Disable**

0: No effect.

1: Disables writing PIO_ODSR for the I/O line.

25.7.35 PIO Output Write Status Register

Name: PIO_OWSR

Address: 0x400E0EA8 (PIOA), 0x400E10A8 (PIOB)

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.

25.7.36 PIO Additional Interrupt Modes Enable Register

Name: PIO_AIMER

Address: 0x400E0EB0 (PIOA), 0x400E10B0 (PIOB)

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.

25.7.37 PIO Additional Interrupt Modes Disable Register

Name: PIO_AIMDR

Address: 0x400E0EB4 (PIOA), 0x400E10B4 (PIOB)

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).

25.7.38 PIO Additional Interrupt Modes Mask Register

Name: PIO_AIMMR

Address: 0x400E0EB8 (PIOA), 0x400E10B8 (PIOB)

Access: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Peripheral CD Status**

0: The interrupt source is a both-edge detection event.

1: The interrupt source is described by the registers PIO_ELSR and PIO_FRLHSR.

25.7.39 PIO Edge Select Register

Name: PIO_ESR

Address: 0x400E0EC0 (PIOA), 0x400E10C0 (PIOB)

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.

25.7.40 PIO Level Select Register

Name: PIO_LSR

Address: 0x400E0EC4 (PIOA), 0x400E10C4 (PIOB)

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.

25.7.41 PIO Edge/Level Status Register

Name: PIO_ELSR

Address: 0x400E0EC8 (PIOA), 0x400E10C8 (PIOB)

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.

25.7.42 PIO Falling Edge/Low-Level Select Register

Name: PIO_FELLSR

Address: 0x400E0ED0 (PIOA), 0x400E10D0 (PIOB)

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.

25.7.43 PIO Rising Edge/High-Level Select Register

Name: PIO_REHLSR

Address: 0x400E0ED4 (PIOA), 0x400E10D4 (PIOB)

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.

25.7.44 PIO Fall/Rise - Low/High Status Register

Name: PIO_FRLHSR

Address: 0x400E0ED8 (PIOA), 0x400E10D8 (PIOB)

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).

25.7.45 PIO Write Protection Mode Register

Name: PIO_WPMR

Address: 0x400E0EE4 (PIOA), 0x400E10E4 (PIOB)

Access: Read/Write

Reset: See Table 25-2

31	30	29	28	27	26	25	24
WPKEY							

23	22	21	20	19	18	17	16
WPKEY							

15	14	13	12	11	10	9	8
WPKEY							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WPEN

For more information on write-protecting registers, refer to Section 25.5.12 ”Register Write Protection”.

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x50494F (“PIO” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x50494F (“PIO” in ASCII).

See Section 25.5.12 ”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.

25.7.46 PIO Write Protection Status Register

Name: PIO_WPSR

Address: 0x400E0EE8 (PIOA), 0x400E10E8 (PIOB)

Access: Read-only

Reset: See Table 25-2

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

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of the PIO_WPSR register.

1: A write protection violation has occurred since the last read of the PIO_WPSR register. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protection Violation Source**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

25.7.47 PIO Schmitt Trigger Register

Name: PIO_SCHMITT

Address: 0x400E0F00 (PIOA), 0x400E1100 (PIOB)

Access: Read/Write

Reset: See Table 25-2

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.

26. Serial Peripheral Interface (SPI)

26.1 Description

The Serial Peripheral Interface (SPI) circuit is a synchronous serial data link that provides communication with external devices in Master or Slave Mode. It also enables communication between processors if an external processor is connected to the system.

The Serial Peripheral Interface is essentially a shift register that serially transmits data bits to other SPIs. During a data transfer, one SPI system acts as the “master”' which controls the data flow, while the other devices act as “slaves" which have data shifted into and out by the master. Different CPUs can take turn being masters (Multiple Master Protocol opposite to Single Master Protocol where one CPU is always the master while all of the others are always slaves) and one master may simultaneously shift data into multiple slaves. However, only one slave may drive its output to write data back to the master at any given time.

A slave device is selected when the master asserts its NSS signal. If multiple slave devices exist, the master generates a separate slave select signal for each slave (NPCS).

The SPI system consists of two data lines and two control lines:

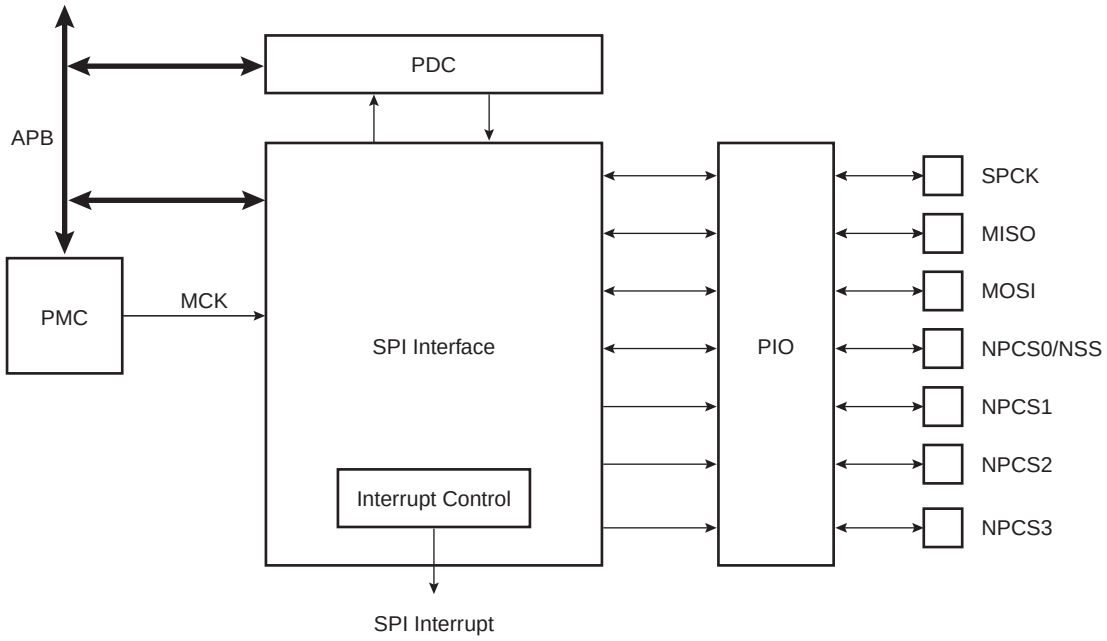
- Master Out Slave In (MOSI): This data line supplies the output data from the master shifted into the input(s) of the slave(s).
- Master In Slave Out (MISO): This data line supplies the output data from a slave to the input of the master. There may be no more than one slave transmitting data during any particular transfer.
- Serial Clock (SPCK): This control line is driven by the master and regulates the flow of the data bits. The master may transmit data at a variety of baud rates; the SPCK line cycles once for each bit that is transmitted.
- Slave Select (NSS): This control line allows slaves to be turned on and off by hardware.

26.2 Embedded Characteristics

- Supports Communication with Serial External Devices
  - Master Mode can drive SPCK up to peripheral clock (bounded by maximum bus clock divided by 2)
  - Slave Mode operates on SPCK, asynchronously to Core and Bus Clock
  - Four Chip Selects with External Decoder Support Allow Communication with Up to 15 Peripherals
  - Serial Memories, such as DataFlash and 3-wire EEPROMs
  - Serial Peripherals, such as ADCs, DACs, LCD Controllers, CAN Controllers and Sensors
  - External Coprocessors
- Master or Slave Serial Peripheral Bus Interface
  - 8-bit to 16-bit Programmable Data Length Per Chip Select
  - Programmable Phase and Polarity Per Chip Select
  - Programmable Transfer Delay Between Consecutive Transfers and Delay before SPI Clock per Chip Select
  - Programmable Delay Between Chip Selects
  - Selectable Mode Fault Detection
- Connection to PDC Channel Capabilities Optimizes Data Transfers
  - One Channel for the Receiver, One Channel for the Transmitter

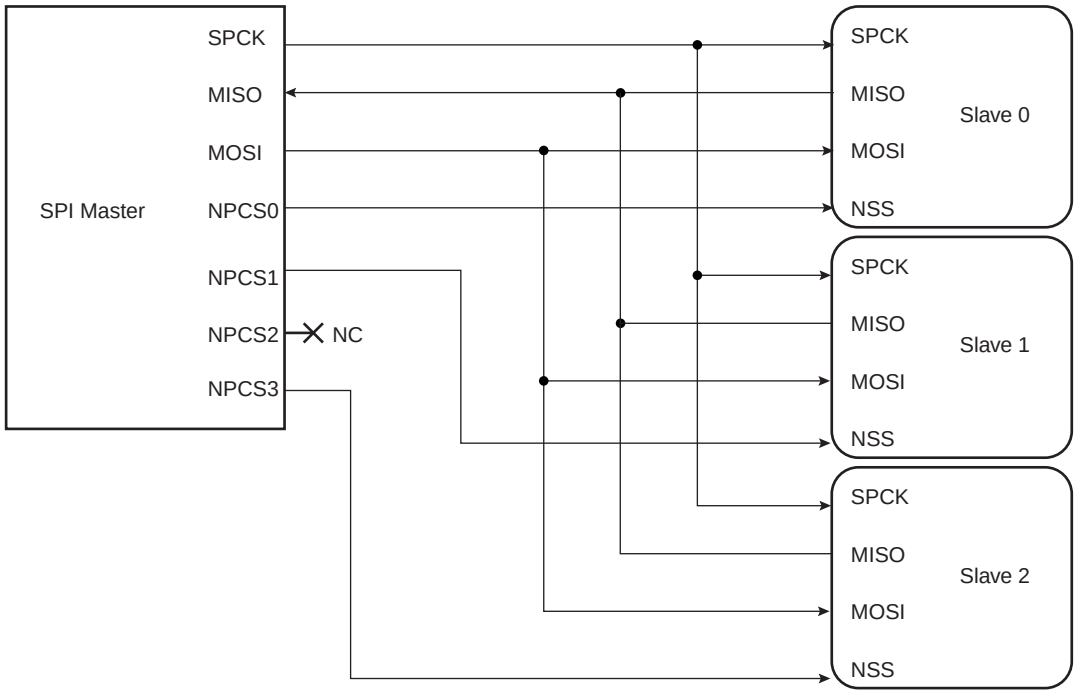
26.3 Block Diagram

Figure 26-1. Block Diagram



26.4 Application Block Diagram

Figure 26-2. Application Block Diagram: Single Master/Multiple Slave Implementation



26.5 Signal Description

Table 26-1. 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

26.6 Product Dependencies

26.6.1 I/O Lines

The pins used for interfacing the compliant external devices may be multiplexed with PIO lines. The programmer must first program the PIO controllers to assign the SPI pins to their peripheral functions.

Table 26-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
SPI	MISO	PA12	A
SPI	MOSI	PA13	A
SPI	NPCS0	PA11	A
SPI	NPCS1	PA9	B
SPI	NPCS1	PB2	B
SPI	SPCK	PA14	A

26.6.2 Power Management

The SPI may be clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the SPI clock.

26.6.3 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 26-3. Peripheral IDs

Instance	ID
SPI	21

26.6.4 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 the corresponding section in the full datasheet.

26.7 Functional Description

26.7.1 Modes of Operation

The SPI operates in Master Mode or in Slave Mode.

Operation in Master Mode is programmed by writing at 1 the MSTR bit in the Mode Register. The pins NPCS0 to NPCS3 are all configured as outputs, the SPCK pin is driven, the MISO line is wired on the receiver input and the MOSI line driven as an output by the transmitter.

If the MSTR bit is written at 0, the SPI operates in Slave Mode. The MISO line is driven by the transmitter output, the MOSI line is wired on the receiver input, the SPCK pin is driven by the transmitter to synchronize the receiver. The NPCS0 pin becomes an input, and is used as a Slave Select signal (NSS). The pins NPCS1 to NPCS3 are not driven and can be used for other purposes.

The data transfers are identically programmable for both modes of operations. The baud rate generator is activated only in Master Mode.

26.7.2 Data Transfer

Four combinations of polarity and phase are available for data transfers. The clock polarity is programmed with the CPOL bit in the Chip Select Register. The clock phase is programmed with the NCPHA bit. These two parameters determine the edges of the clock signal on which data is driven and sampled. Each of the two parameters has two possible states, resulting in four possible combinations that are incompatible with one another. Thus, a master/slave pair must use the same parameter pair values to communicate. If multiple slaves are used and fixed in different configurations, the master must reconfigure itself each time it needs to communicate with a different slave.

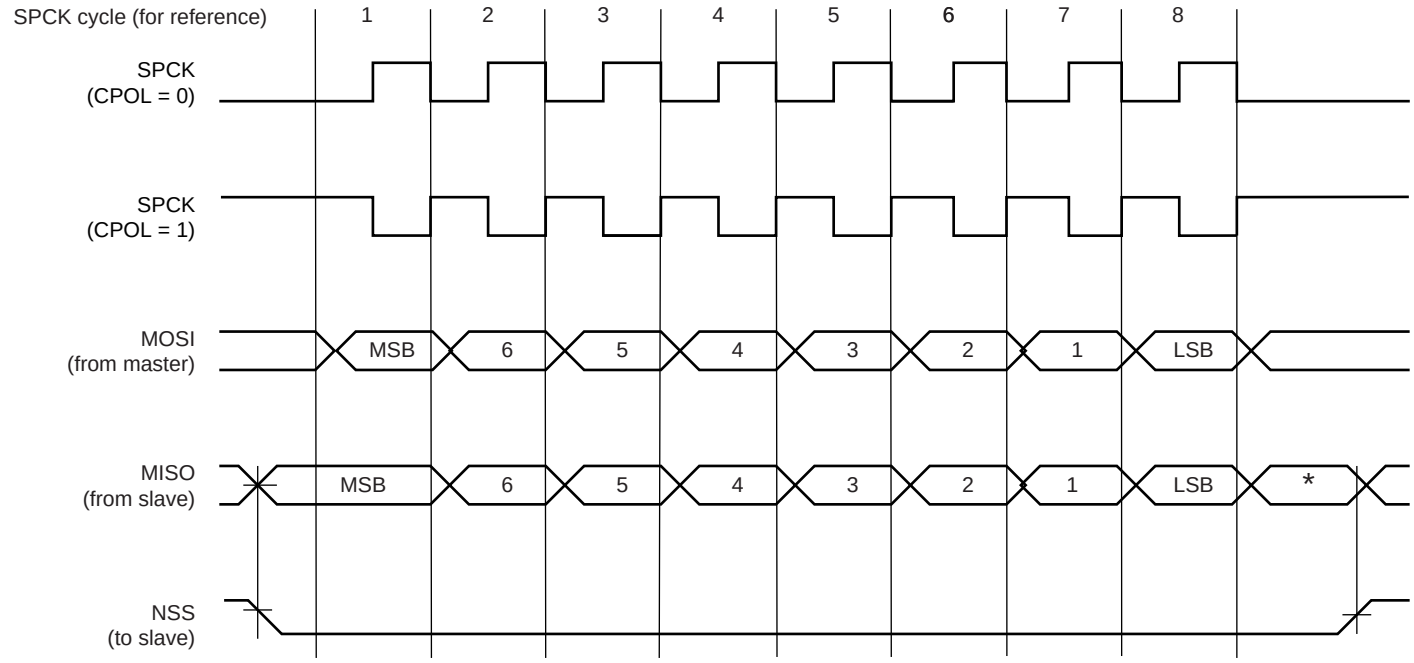
Table 26-4 shows the four modes and corresponding parameter settings.

Table 26-4. SPI Bus Protocol Mode

SPI Mode	CPOL	NCPHA	Shift SPCK Edge	Capture SPCK Edge	SPCK Inactive Level
0	0	1	Falling	Rising	Low
1	0	0	Rising	Falling	Low
2	1	1	Rising	Falling	High
3	1	0	Falling	Rising	High

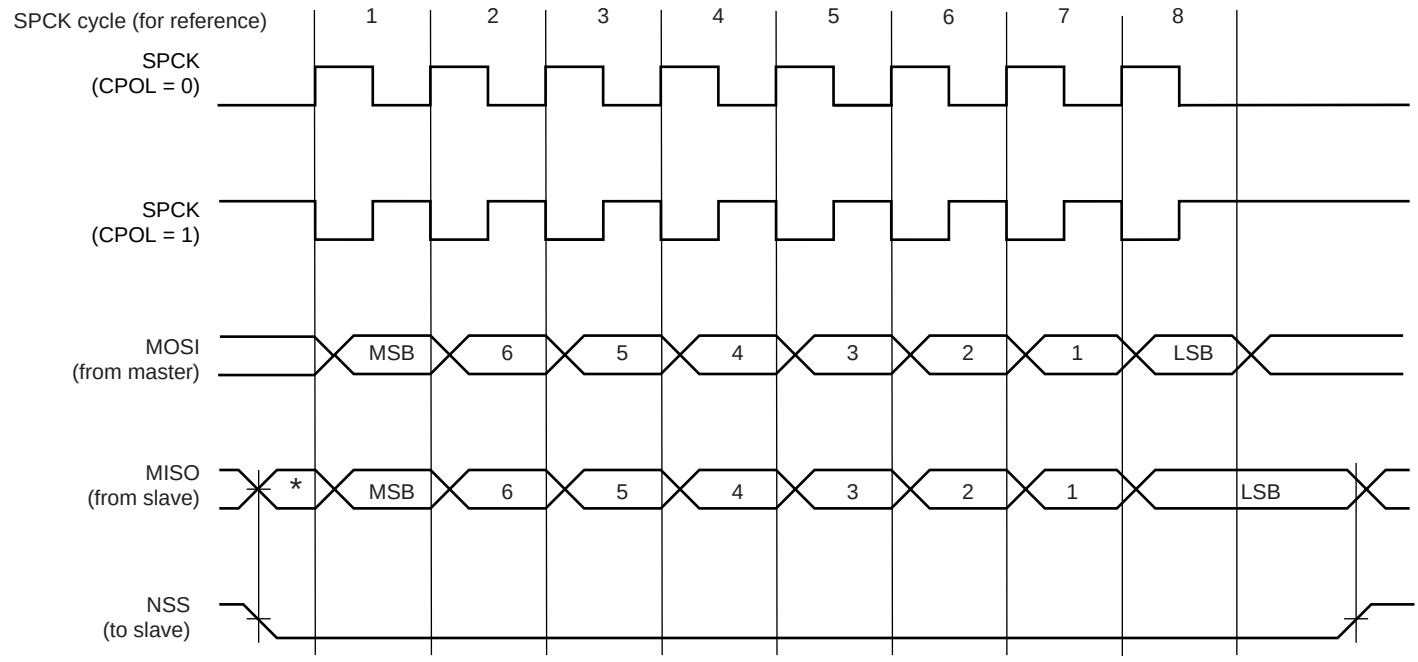
Figure 26-3 and Figure 26-4 show examples of data transfers.

Figure 26-3. SPI Transfer Format (NCPHA = 1, 8 bits per transfer)



* Not defined, but normally MSB of previous character received.

Figure 26-4. SPI Transfer Format (NCPHA = 0, 8 bits per transfer)



* Not defined but normally LSB of previous character transmitted.

26.7.3 Master Mode Operations

When configured in Master Mode, the SPI operates on the clock generated by the internal programmable baud rate generator. It fully controls the data transfers to and from the slave(s) connected to the SPI bus. The SPI drives the chip select line to the slave and the serial clock signal (SPCK).

The SPI features two holding registers, the Transmit Data Register and the Receive Data Register, and a single Shift Register. The holding registers maintain the data flow at a constant rate.

After enabling the SPI, a data transfer begins when the processor writes to the SPI_TDR (Transmit Data Register). The written data is immediately transferred in the Shift Register and transfer on the SPI bus starts. While the data in the Shift Register is shifted on the MOSI line, the MISO line is sampled and shifted in the Shift Register. Receiving data cannot occur without transmitting data. If receiving mode is not needed, for example when communicating with a slave receiver only (such as an LCD), the receive status flags in the status register can be discarded.

Before writing the TDR, the PCS field in the SPI_MR register must be set in order to select a slave.

After enabling the SPI, a data transfer begins when the processor writes to the SPI_TDR (Transmit Data Register). The written data is immediately transferred in the Shift Register and transfer on the SPI bus starts. While the data in the Shift Register is shifted on the MOSI line, the MISO line is sampled and shifted in the Shift Register. Transmission cannot occur without reception.

Before writing the TDR, the PCS field must be set in order to select a slave.

If new data is written in SPI_TDR during the transfer, it stays in it until the current transfer is completed. Then, the received data is transferred from the Shift Register to SPI_RDR, the data in SPI_TDR is loaded in the Shift Register and a new transfer starts.

The transfer of a data written in SPI_TDR in the Shift Register is indicated by the TDRE bit (Transmit Data Register Empty) in the Status Register (SPI_SR). When new data is written in SPI_TDR, this bit is cleared. The TDRE bit is used to trigger the Transmit PDC channel.

The end of transfer is indicated by the TXEMPTY flag in the SPI_SR register. If a transfer delay (DLYBCT) is greater than 0 for the last transfer, TXEMPTY is set after the completion of said delay. The master clock (MCK) can be switched off at this time.

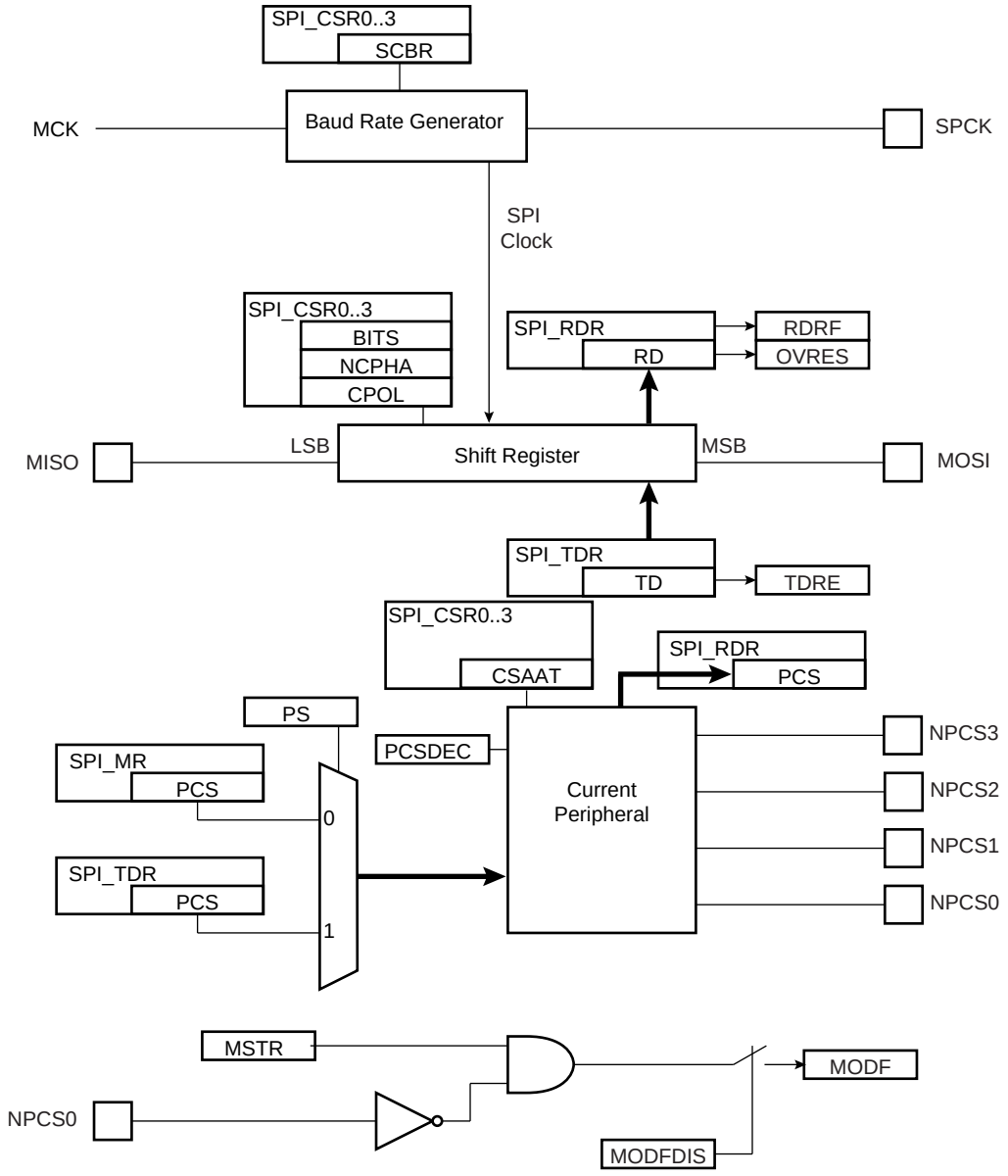
The transfer of received data from the Shift Register in SPI_RDR is indicated by the RDRF bit (Receive Data Register Full) in the Status Register (SPI_SR). When the received data is read, the RDRF bit is cleared.

If the SPI_RDR (Receive Data Register) has not been read before new data is received, the Overrun Error bit (OVRES) in SPI_SR is set. As long as this flag is set, data is loaded in SPI_RDR. The user has to read the status register to clear the OVRES bit.

Figure 26-5, shows a block diagram of the SPI when operating in Master Mode. Figure 26-6 on page 529 shows a flow chart describing how transfers are handled.

26.7.3.1 Master Mode Block Diagram

Figure 26-5. Master Mode Block Diagram



26.7.3.2 Master Mode Flow Diagram

Figure 26-6. Master Mode Flow Diagram

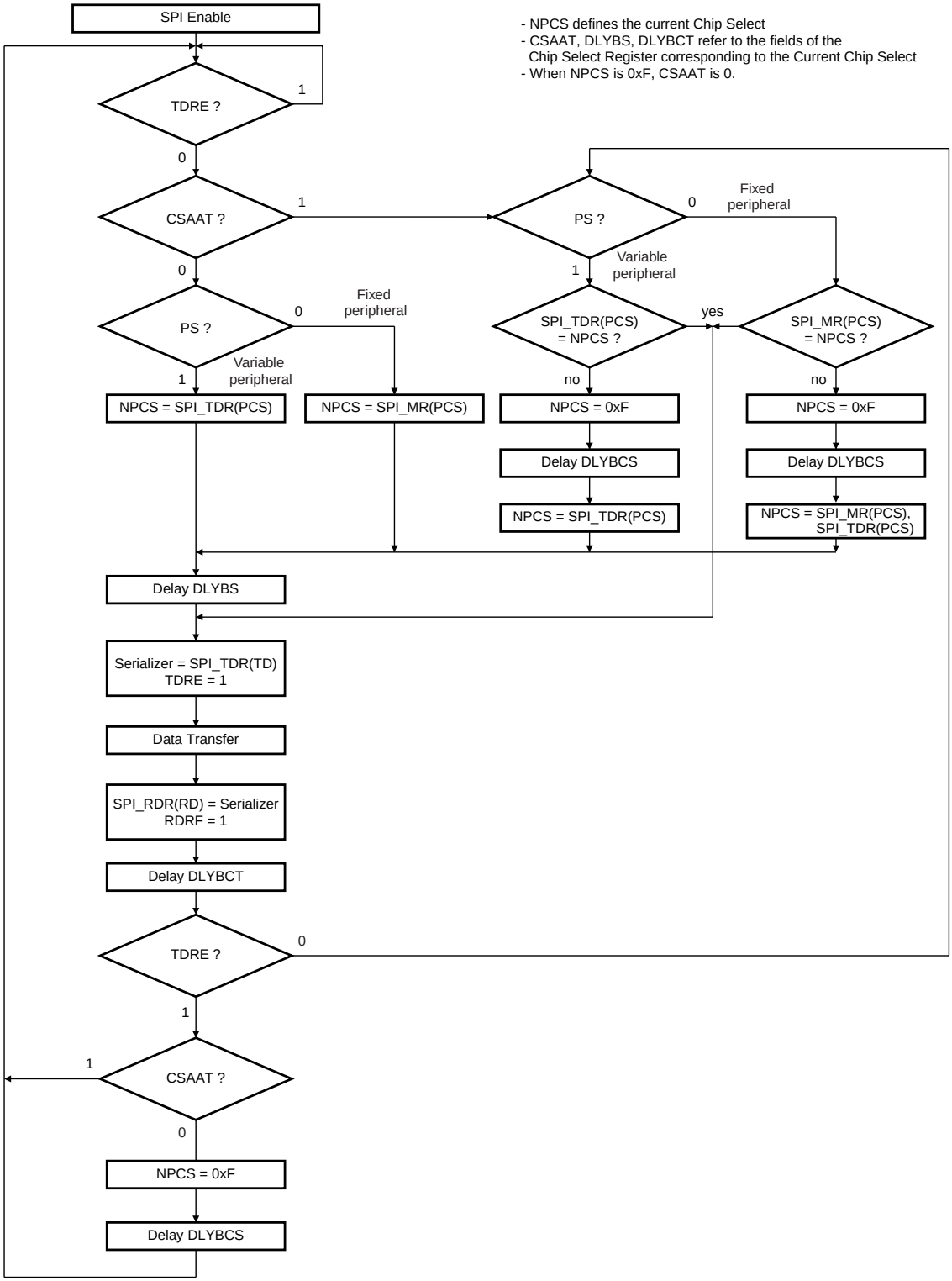


Figure 26-7 shows Transmit Data Register Empty (TDRE), Receive Data Register (RDRF) and Transmission Register Empty (TXEMPTY) status flags behavior within the SPI_SR (Status Register) during an 8-bit data transfer in fixed mode and no Peripheral Data Controller involved.

Figure 26-7. Status Register Flags Behavior

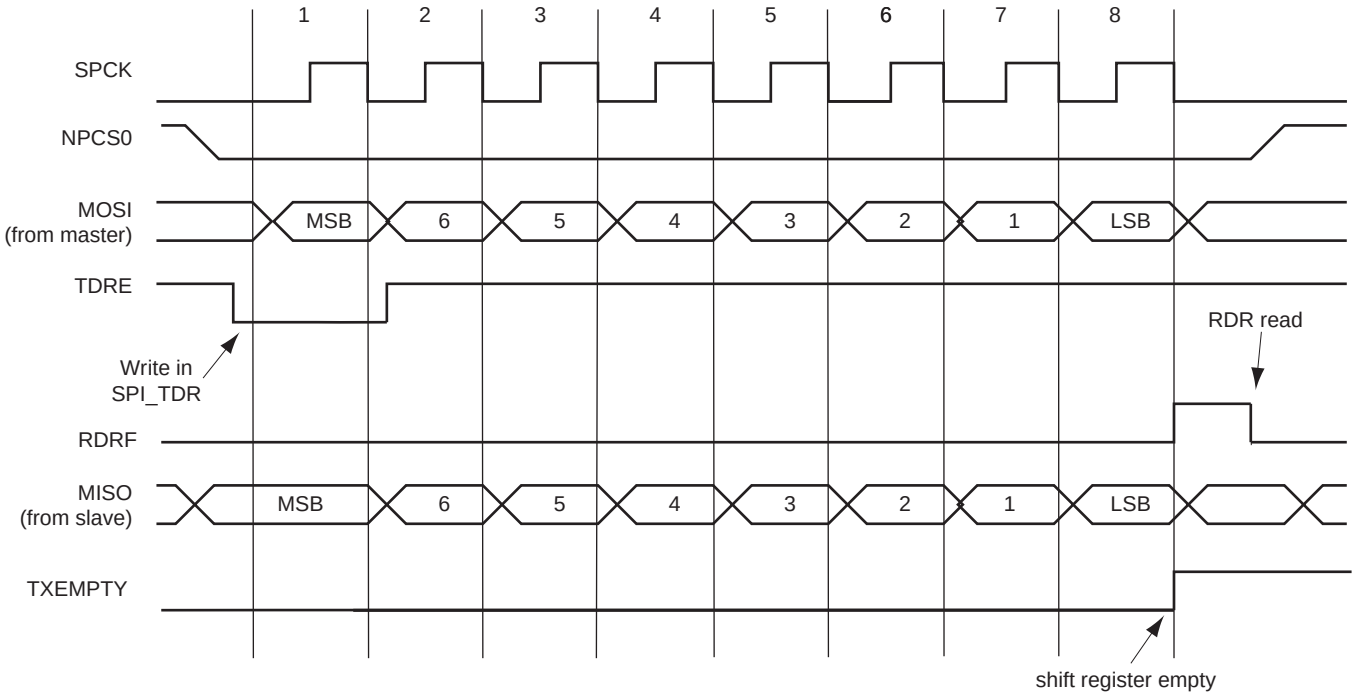
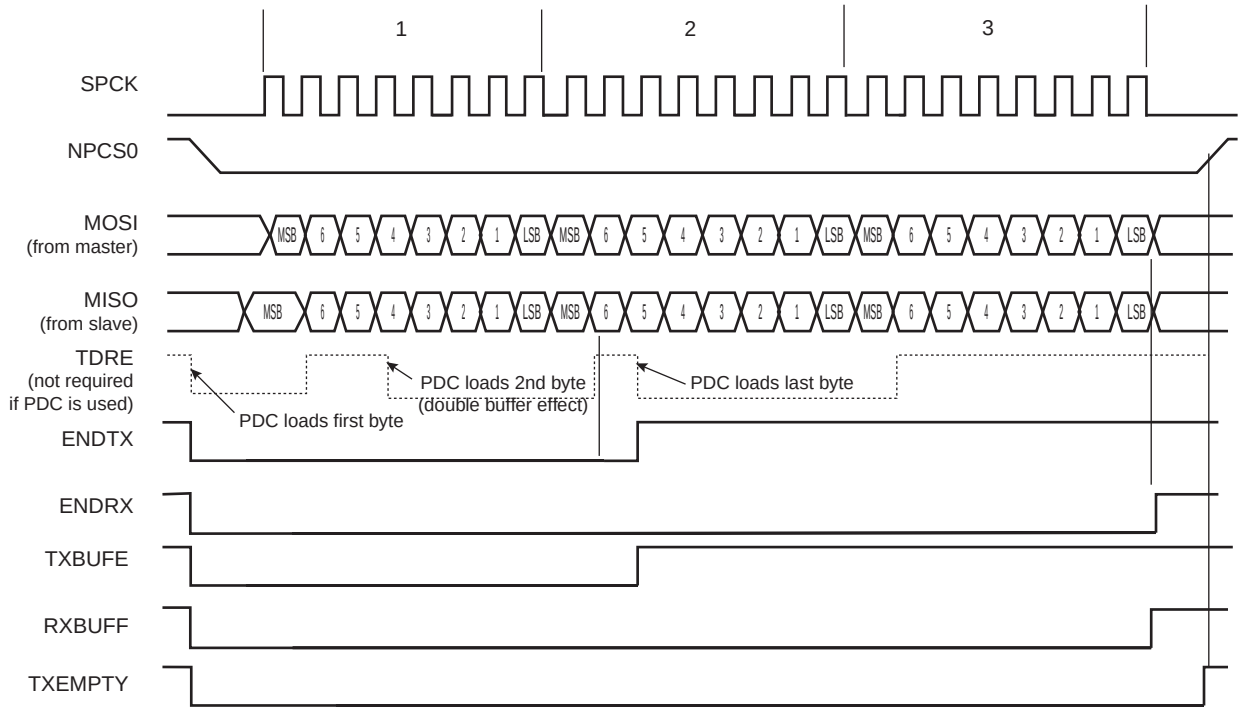


Figure 26-8 shows Transmission Register Empty (TXEMPTY), End of RX buffer (ENDRX), End of TX buffer (ENDTX), RX Buffer Full (RXBUFF) and TX Buffer Empty (TXBUFE) status flags behavior within the SPI_SR (Status Register) during an 8-bit data transfer in fixed mode with the Peripheral Data Controller involved. The PDC is programmed to transfer and receive three data. The next pointer and counter are not used. The RDRF and TDRE are not shown because these flags are managed by the PDC when using the PDC.

Figure 26-8. PDC Status Register Flags Behavior



26.7.3.3 Clock Generation

The SPI Baud rate clock is generated by dividing the Master Clock (MCK), by a value between 1 and 255.

This allows a maximum operating baud rate at up to Master Clock and a minimum operating baud rate of MCK divided by 255.

Programming the SCBR field at 0 is forbidden. Triggering a transfer while SCBR is at 0 can lead to unpredictable results.

At reset, SCBR is 0 and the user has to program it at a valid value before performing the first transfer.

The divisor can be defined independently for each chip select, as it has to be programmed in the SCBR field of the Chip Select Registers. This allows the SPI to automatically adapt the baud rate for each interfaced peripheral without reprogramming.

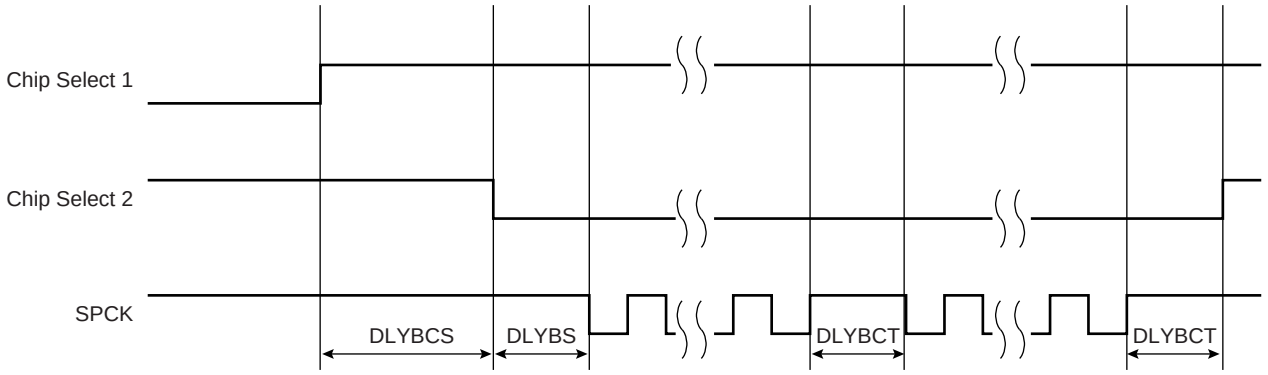
26.7.3.4 Transfer Delays

Figure 26-9 shows a chip select transfer change and consecutive transfers on the same chip select. Three delays can be programmed to modify the transfer waveforms:

- The delay between chip selects, programmable only once for all the chip selects by writing the DLYBCS field in the Mode Register. Allows insertion of a delay between release of one chip select and before assertion of a new one.
- The delay before SPCK, independently programmable for each chip select by writing the field DLYBS. Allows the start of SPCK to be delayed after the chip select has been asserted.
- The delay between consecutive transfers, independently programmable for each chip select by writing the DLYBCT field. Allows insertion of a delay between two transfers occurring on the same chip select

These delays allow the SPI to be adapted to the interfaced peripherals and their speed and bus release time.

Figure 26-9. Programmable Delays



26.7.3.5 Peripheral Selection

The serial peripherals are selected through the assertion of the NPCS0 to NPCS3 signals. By default, all the NPCS signals are high before and after each transfer.

- Fixed Peripheral Select: SPI exchanges data with only one peripheral

Fixed Peripheral Select is activated by writing the PS bit to zero in SPI_MR (Mode Register). In this case, the current peripheral is defined by the PCS field in SPI_MR and the PCS field in the SPI_TDR has no effect.

- Variable Peripheral Select: Data can be exchanged with more than one peripheral without having to reprogram the NPCS field in the SPI_MR register.

Variable Peripheral Select is activated by setting PS bit to one. The PCS field in SPI_TDR is used to select the current peripheral. This means that the peripheral selection can be defined for each new data. The value to write in the SPI_TDR register as the following format.

[xxxxxxx(7-bit) + LASTXFER(1-bit)[(1)]+ xxxx(4-bit) + PCS (4-bit) + DATA (8 to 16-bit)] with PCS equals to the chip select to assert as defined in Section 26.8.4 (SPI Transmit Data Register) and LASTXFER bit at 0 or 1 depending on CSAAT bit.

Note: 1. Optional.

CSAAT, LASTXFER and CSNAAT bits are discussed in Section 26.7.3.9 ”Peripheral Deselection with PDC” .

If LASTXFER is used, the command must be issued before writing the last character. Instead of LASTXFER, the user can use the SPIDIS command. After the end of the PDC transfer, wait for the TXEMPTY flag, then write SPIDIS into the SPI_CR register (this will not change the configuration register values); the NPCS will be deactivated after the last character transfer. Then, another PDC transfer can be started if the SPIEN was previously written in the SPI_CR register.

26.7.3.6 SPI Peripheral DMA Controller (PDC)

In both fixed and variable mode the Peripheral DMA Controller (PDC) can be used to reduce processor overhead.

The Fixed Peripheral Selection allows buffer transfers with a single peripheral. Using the PDC is an optimal means, as the size of the data transfer between the memory and the SPI is either 8 bits or 16 bits. However, changing the peripheral selection requires the Mode Register to be reprogrammed.

The Variable Peripheral Selection allows buffer transfers with multiple peripherals without reprogramming the Mode Register. Data written in SPI_TDR is 32 bits wide and defines the real data to be transmitted and the peripheral it is destined to. Using the PDC in this mode requires 32-bit wide buffers, with the data in the LSBs and the PCS and LASTXFER fields in the MSBs, however the SPI still controls the number of bits (8 to16) to be transferred through MISO and MOSI lines with the chip select configuration registers. This is not the optimal means in term of memory size for the buffers, but it provides a very effective means to exchange data with several peripherals without any intervention of the processor.

Transfer Size

Depending on the data size to transmit, from 8 to 16 bits, the PDC manages automatically the type of pointer's size it has to point to. The PDC will perform the following transfer size depending on the mode and number of bits per data.

Fixed Mode:

- 8-bit Data:
  Byte transfer, PDC Pointer Address = Address + 1 byte,
  PDC Counter = Counter - 1
- 8-bit to 16-bit Data:
  2 bytes transfer. n-bit data transfer with don't care data (MSB) filled with 0's,
  PDC Pointer Address = Address + 2 bytes,
  PDC Counter = Counter - 1

Variable Mode:

In variable mode, PDC Pointer Address = Address +4 bytes and PDC Counter = Counter - 1 for 8 to 16-bit transfer size. When using the PDC, the TDRE and RDRF flags are handled by the PDC, thus the user's application does not have to check those bits. Only End of RX Buffer (ENDRX), End of TX Buffer (ENDTX), Buffer Full (RXBUFF), TX Buffer Empty (TXBUFE) are significant. For further details about the Peripheral DMA Controller and user interface, refer to the PDC section of the product datasheet.

26.7.3.7 Peripheral Chip Select Decoding

The user can program the SPI to operate with up to 15 peripherals by decoding the four Chip Select lines, NPCS0 to NPCS3 with 1 of up to 16 decoder/demultiplexer. This can be enabled by writing the PCSDEC bit at 1 in the Mode Register (SPI_MR).

When operating without decoding, the SPI makes sure that in any case only one chip select line is activated, i.e., one NPCS line driven low at a time. If two bits are defined low in a PCS field, only the lowest numbered chip select is driven low.

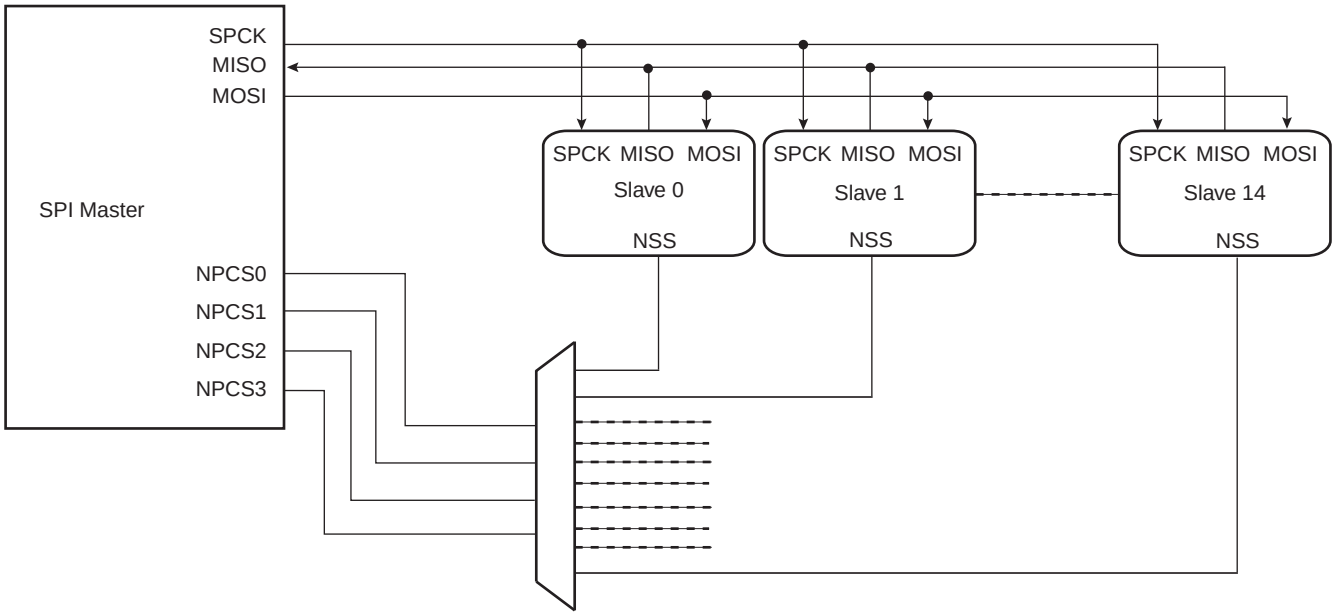
When operating with decoding, the SPI directly outputs the value defined by the PCS field on NPCS lines of either the Mode Register or the Transmit Data Register (depending on PS).

As the SPI sets a default value of 0xF on the chip select lines (i.e. all chip select lines at 1) when not processing any transfer, only 15 peripherals can be decoded.

The SPI has only four Chip Select Registers, not 15. As a result, when decoding is activated, each chip select defines the characteristics of up to four peripherals. As an example, SPI_CRS0 defines the characteristics of the externally decoded peripherals 0 to 3, corresponding to the PCS values 0x0 to 0x3. Thus, the user has to make sure to connect compatible peripherals on the decoded chip select lines 0 to 3, 4 to 7, 8 to 11 and 12 to 14. Figure 26-10 below shows such an implementation.

If the CSAAT bit is used, with or without the PDC, the Mode Fault detection for NPCS0 line must be disabled. This is not needed for all other chip select lines since Mode Fault Detection is only on NPCS0.

Figure 26-10. Chip Select Decoding Application Block Diagram: Single Master/Multiple Slave Implementation



26.7.3.8 Peripheral Deselection without PDC

During a transfer of more than one data on a Chip Select without the PDC, the SPI_TDR is loaded by the processor, the flag TDRE rises as soon as the content of the SPI_TDR is transferred into the internal shift register. When this flag is detected high, the SPI_TDR can be reloaded. If this reload by the processor occurs before the end of the current transfer and if the next transfer is performed on the same chip select as the current transfer, the Chip Select is not de-asserted between the two transfers. But depending on the application software handling the SPI status register flags (by interrupt or polling method) or servicing other interrupts or other tasks, the processor may not reload the SPI_TDR in time to keep the chip select active (low). A null Delay Between Consecutive Transfer (DLYBCT) value in the SPI_CSR register, will give even less time for the processor to reload the SPI_TDR. With some SPI slave peripherals, requiring the chip select line to remain active (low) during a full set of transfers might lead to communication errors.

To facilitate interfacing with such devices, the Chip Select Register [CSR0...CSR3] can be programmed with the CSAAT bit (Chip Select Active After Transfer) at 1. This allows the chip select lines to remain in their current state (low = active) until transfer to another chip select is required. Even if the SPI_TDR is not reloaded the chip select will remain active. To have the chip select line to raise at the end of the transfer the Last transfer Bit (LASTXFER) in the SPI_MR register must be set at 1 before writing the last data to transmit into the SPI_TDR.

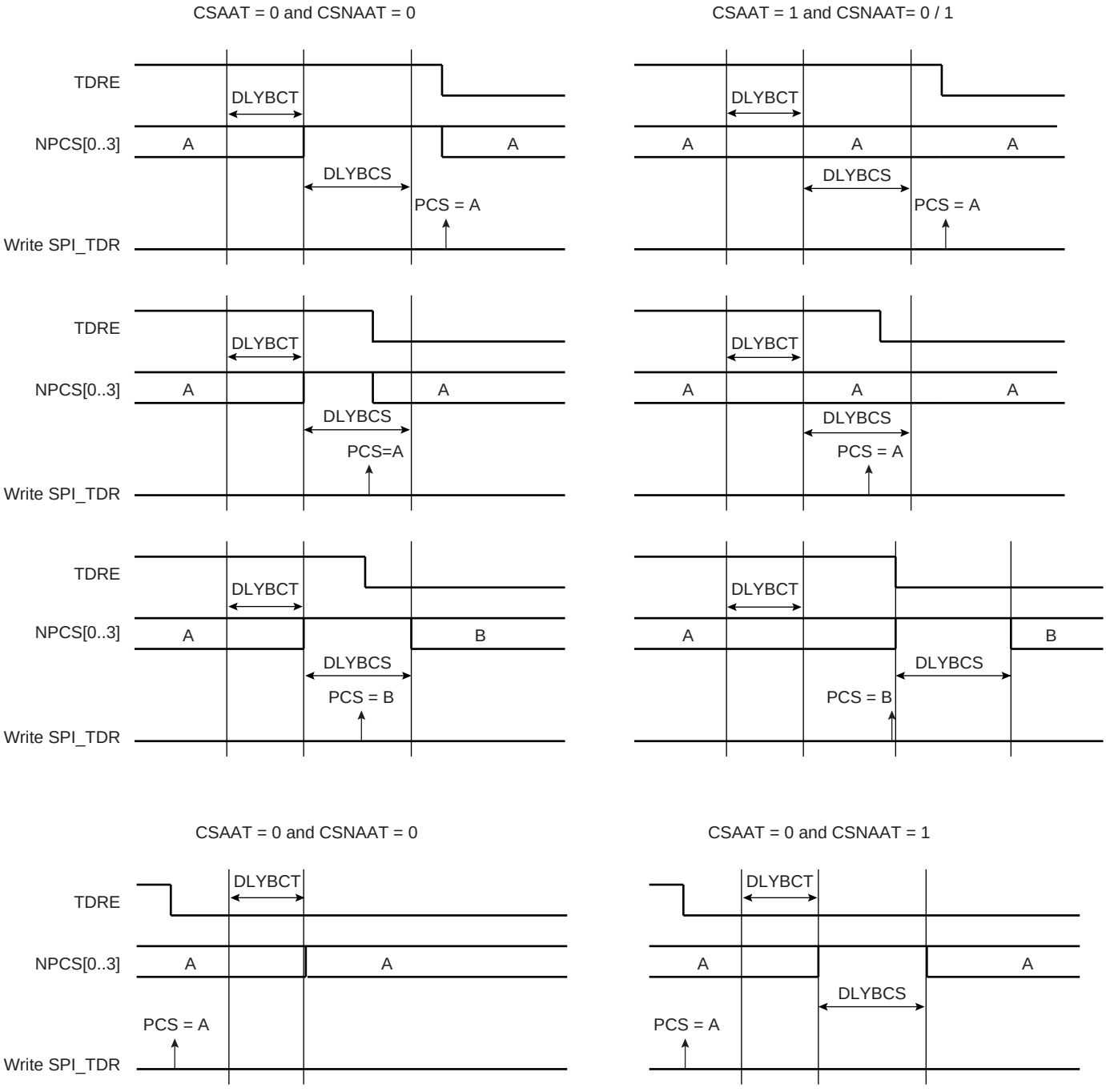
26.7.3.9 Peripheral Deselection with PDC

When the Peripheral DMA Controller is used, the chip select line will remain low during the whole transfer since the TDRE flag is managed by the PDC itself. The reloading of the SPI_TDR by the PDC is done as soon as TDRE flag is set to one. In this case the use of CSAAT bit might not be needed. However, it may happen that when other PDC channels connected to other peripherals are in use as well, the SPI PDC might be delayed by another (PDC with a higher priority on the bus). Having PDC buffers in slower memories like flash memory or SDRAM compared to fast internal SRAM, may lengthen the reload time of the SPI_TDR by the PDC as well. This means that the SPI_TDR might not be reloaded in time to keep the chip select line low. In this case the chip select line may toggle between data transfer and according to some SPI Slave devices, the communication might get lost. The use of the CSAAT bit might be needed. When the CSAAT bit is set at 0, the NPCS does not rise in all cases between two transfers on the same peripheral. During a transfer on a Chip Select, the flag TDRE rises as soon as the content of the SPI_TDR is transferred into the internal shifter. When this flag is detected the SPI_TDR can be reloaded. If this reload occurs before the end of the current transfer and if the next transfer is performed on the same chip select as the current transfer, the Chip Select is not de-asserted between the two transfers. This might lead to difficulties for interfacing with some serial peripherals requiring the

chip select to be de-asserted after each transfer. To facilitate interfacing with such devices, the Chip Select Register can be programmed with the CSNAAT bit (Chip Select Not Active After Transfer) at 1. This allows to de-assert systematically the chip select lines during a time DLYBCS. (The value of the CSNAAT bit is taken into account only if the CSAAT bit is set at 0 for the same Chip Select).

Figure 26-11 shows different peripheral deselection cases and the effect of the CSAAT and CSNAAT bits.

Figure 26-11. Peripheral Deselection



26.7.3.10 Mode Fault Detection

A mode fault is detected when the SPI is programmed in Master Mode and a low level is driven by an external master on the NPCS0/NSS signal. In this case, multi-master configuration, NPCS0, MOSI, MISO and SPCK pins must be configured in open drain (through the PIO controller). When a mode fault is detected, the MODF bit in the SPI_SR is set until the SPI_SR is read and the SPI is automatically disabled until re-enabled by writing the SPIEN bit in the SPI_CR (Control Register) at 1.

By default, the Mode Fault detection circuitry is enabled. The user can disable Mode Fault detection by setting the MODFDIS bit in the SPI Mode Register (SPI_MR).

26.7.4 SPI Slave Mode

When operating in Slave Mode, the SPI processes data bits on the clock provided on the SPI clock pin (SPCK).

The SPI waits for NSS to go active before receiving the serial clock from an external master. When NSS falls, the clock is validated on the serializer, which processes the number of bits defined by the BITS field of the Chip Select Register 0 (SPI_CSR0). These bits are processed following a phase and a polarity defined respectively by the NCPHA and CPOL bits of the SPI_CSR0. Note that BITS, CPOL and NCPHA of the other Chip Select Registers have no effect when the SPI is programmed in Slave Mode.

The bits are shifted out on the MISO line and sampled on the MOSI line.

(For more information on BITS field, see also, the (Note:) below the register table; Section 26.8.9 “SPI Chip Select Register” on page 550.)

When all the bits are processed, the received data is transferred in the Receive Data Register and the RDRF bit rises. If the SPI_RDR (Receive Data Register) has not been read before new data is received, the Overrun Error bit (OVRES) in SPI_SR is set. As long as this flag is set, data is loaded in SPI_RDR. The user has to read the status register to clear the OVRES bit.

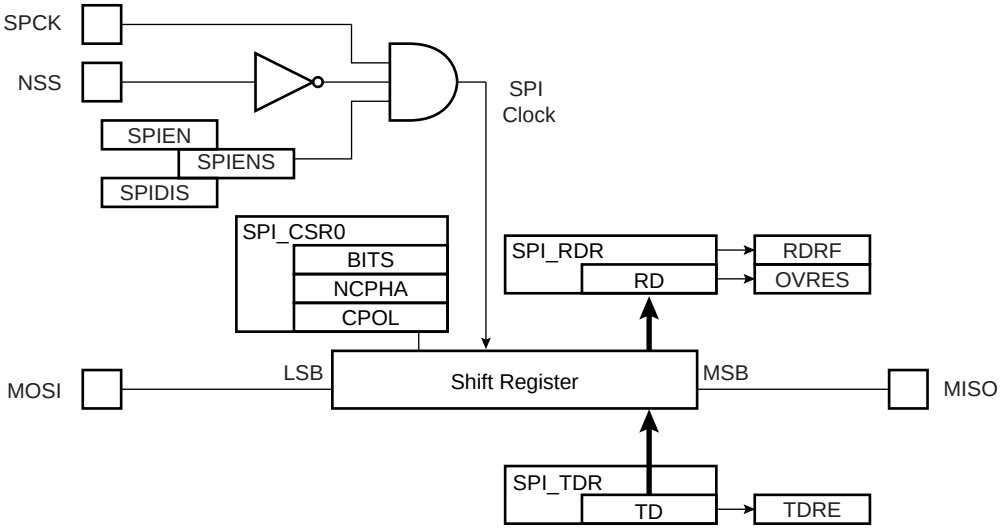
When a transfer starts, the data shifted out is the data present in the Shift Register. If no data has been written in the Transmit Data Register (SPI_TDR), the last data received is transferred. If no data has been received since the last reset, all bits are transmitted low, as the Shift Register resets at 0.

When a first data is written in SPI_TDR, it is transferred immediately in the Shift Register and the TDRE bit rises. If new data is written, it remains in SPI_TDR until a transfer occurs, i.e. NSS falls and there is a valid clock on the SPCK pin. When the transfer occurs, the last data written in SPI_TDR is transferred in the Shift Register and the TDRE bit rises. This enables frequent updates of critical variables with single transfers.

Then, a new data is loaded in the Shift Register from the Transmit Data Register. In case no character is ready to be transmitted, i.e. no character has been written in SPI_TDR since the last load from SPI_TDR to the Shift Register, the SPI_TDR is retransmitted. In this case the Underrun Error Status Flag (UNDES) is set in the SPI_SR.

Figure 26-12 shows a block diagram of the SPI when operating in Slave Mode.

Figure 26-12. Slave Mode Functional Block Diagram



26.7.5 Write Protected Registers

To prevent any single software error that may corrupt SPI behavior, the registers listed below can be write-protected by setting the WPEN bit in the SPI Write Protection Mode Register (SPI_WPMR).

If a write access in a write-protected register is detected, then the WPVS flag in the SPI Write Protection Status Register (SPI_WPSR) is set and the field WPVSRC indicates in which register the write access has been attempted.

The WPVS flag is automatically reset after reading the SPI Write Protection Status Register (SPI_WPSR).

List of the write-protected registers:

Section 26.8.2 ”SPI Mode Register”

Section 26.8.9 ”SPI Chip Select Register”

26.8 Serial Peripheral Interface (SPI) User Interface

Table 26-5. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	SPI_CR	Write-only	---
0x04	Mode Register	SPI_MR	Read-write	0x0
0x08	Receive Data Register	SPI_RDR	Read-only	0x0
0x0C	Transmit Data Register	SPI_TDR	Write-only	---
0x10	Status Register	SPI_SR	Read-only	0x000000F0
0x14	Interrupt Enable Register	SPI_IER	Write-only	---
0x18	Interrupt Disable Register	SPI_IDR	Write-only	---
0x1C	Interrupt Mask Register	SPI_IMR	Read-only	0x0
0x20 - 0x2C	Reserved			
0x30	Chip Select Register 0	SPI_CSR0	Read-write	0x0
0x34	Chip Select Register 1	SPI_CSR1	Read-write	0x0
0x38	Chip Select Register 2	SPI_CSR2	Read-write	0x0
0x3C	Chip Select Register 3	SPI_CSR3	Read-write	0x0
0x40 - 0xE0	Reserved	–	–	–
0xE4	Write Protection Control Register	SPI_WPMR	Read-write	0x0
0xE8	Write Protection Status Register	SPI_WPSR	Read-only	0x0
0x00EC - 0x00F8	Reserved	–	–	–
0x00FC	Reserved	–	–	–
0x100 - 0x124	Reserved for PDC Registers	–	–	–

26.8.1 SPI Control Register

Name: SPI_CR

Address: 0x40008000

Access: Write-only

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

- **SPIEN: SPI Enable**

0 = No effect.

1 = Enables the SPI to transfer and receive data.

- **SPIDIS: SPI Disable**

0 = No effect.

1 = Disables the SPI.

As soon as SPIDIS is set, SPI finishes its transfer.

All pins are set in input mode and no data is received or transmitted.

If a transfer is in progress, the transfer is finished before the SPI is disabled.

If both SPIEN and SPIDIS are equal to one when the control register is written, the SPI is disabled.

- **SWRST: SPI Software Reset**

0 = No effect.

1 = Reset the SPI. A software-triggered hardware reset of the SPI interface is performed.

The SPI is in slave mode after software reset.

PDC channels are not affected by software reset.

- **LASTXFER: Last Transfer**

0 = No effect.

1 = The current NPCS will be deasserted after the character written in TD has been transferred. When CSAAT is set, this allows to close the communication with the current serial peripheral by raising the corresponding NPCS line as soon as TD transfer has completed.

Refer to Section 26.7.3.5 ”Peripheral Selection” for more details.

26.8.2 SPI Mode Register

Name: SPI_MR

Address: 0x40008004

Access: Read-write

31	30	29	28	27	26	25	24
DLYBCS							
23	22	21	20	19	18	17	16
–	–	–	–	PCS			
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
LLB	–	WDRBT	MODFDIS	–	PCSDEC	PS	MSTR

This register can only be written if the WPEN bit is cleared in "SPI Write Protection Mode Register".

- **MSTR: Master/Slave Mode**

0 = SPI is in Slave mode.

1 = SPI is in Master mode.

- **PS: Peripheral Select**

0 = Fixed Peripheral Select.

1 = Variable Peripheral Select.

- **PCSDEC: Chip Select Decode**

0 = The chip selects are directly connected to a peripheral device.

1 = The four chip select lines are connected to a 4- to 16-bit decoder.

When PCSDEC equals one, up to 15 Chip Select signals can be generated with the four lines using an external 4- to 16-bit decoder. The Chip Select Registers define the characteristics of the 15 chip selects according to the following rules:

SPI_CSR0 defines peripheral chip select signals 0 to 3.

SPI_CSR1 defines peripheral chip select signals 4 to 7.

SPI_CSR2 defines peripheral chip select signals 8 to 11.

SPI_CSR3 defines peripheral chip select signals 12 to 14.

- **MODFDIS: Mode Fault Detection**

0 = Mode fault detection is enabled.

1 = Mode fault detection is disabled.

- **WDRBT: Wait Data Read Before Transfer**

0 = No Effect. In master mode, a transfer can be initiated whatever the state of the Receive Data Register is.

1 = In Master Mode, a transfer can start only if the Receive Data Register is empty, i.e. does not contain any unread data. This mode prevents overrun error in reception.

- **LLB: Local Loopback Enable**

0 = Local loopback path disabled.

1 = Local loopback path enabled

LLB controls the local loopback on the data serializer for testing in Master Mode only. (MISO is internally connected on MOSI.)

- **PCS: Peripheral Chip Select**

This field is only used if Fixed Peripheral Select is active (PS = 0).

If PCSDEC = 0:

PCS = xxx0	NPCS[3:0] = 1110
PCS = xx01	NPCS[3:0] = 1101
PCS = x011	NPCS[3:0] = 1011
PCS = 0111	NPCS[3:0] = 0111
PCS = 1111	forbidden (no peripheral is selected)

(x = don’t care)

If PCSDEC = 1:

NPCS[3:0] output signals = PCS.

- **DLYBCS: Delay Between Chip Selects**

This field defines the delay from NPCS inactive to the activation of another NPCS. The DLYBCS time guarantees non-overlapping chip selects and solves bus contentions in case of peripherals having long data float times.

If DLYBCS is less than or equal to six, six MCK periods will be inserted by default.

Otherwise, the following equation determines the delay:

$$\text{Delay Between Chip Selects} = \frac{DLYBCS}{MCK}$$

26.8.3 SPI Receive Data Register

Name: SPI_RDR

Address: 0x40008008

Access: Read-only

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

- **RD: Receive Data**

Data received by the SPI Interface is stored in this register right-justified. Unused bits read zero.

- **PCS: Peripheral Chip Select**

In Master Mode only, these bits indicate the value on the NPCS pins at the end of a transfer. Otherwise, these bits read zero.

Note: When using variable peripheral select mode (PS = 1 in SPI_MR) it is mandatory to also set the WDRBT field to 1 if the SPI_RDR PCS field is to be processed.

26.8.4 SPI Transmit Data Register

Name: SPI_TDR

Address: 0x4000800C

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	LASTXFER

23	22	21	20	19	18	17	16
–	–	–	–	PCS			

15	14	13	12	11	10	9	8
TD							

7	6	5	4	3	2	1	0
TD							

- **TD: Transmit Data**

Data to be transmitted by the SPI Interface is stored in this register. Information to be transmitted must be written to the transmit data register in a right-justified format.

- **PCS: Peripheral Chip Select**

This field is only used if Variable Peripheral Select is active (PS = 1).

If PCSDEC = 0:

PCS = xxx0 NPCS[3:0] = 1110

PCS = xx01 NPCS[3:0] = 1101

PCS = x011 NPCS[3:0] = 1011

PCS = 0111 NPCS[3:0] = 0111

PCS = 1111 forbidden (no peripheral is selected)

(x = don’t care)

If PCSDEC = 1:

NPCS[3:0] output signals = PCS

- **LASTXFER: Last Transfer**

0 = No effect.

1 = The current NPCS will be deasserted after the character written in TD has been transferred. When CSAAT is set, this allows to close the communication with the current serial peripheral by raising the corresponding NPCS line as soon as TD transfer has completed.

This field is only used if Variable Peripheral Select is active (PS = 1).

26.8.5 SPI Status Register

Name: SPI_SR

Address: 0x40008010

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	SPIENS
15	14	13	12	11	10	9	8
–	–	–	–	–	UNDES	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

- **RDRF: Receive Data Register Full**

0 = No data has been received since the last read of SPI_RDR

1 = Data has been received and the received data has been transferred from the serializer to SPI_RDR since the last read of SPI_RDR.

- **TDRE: Transmit Data Register Empty**

0 = Data has been written to SPI_TDR and not yet transferred to the serializer.

1 = The last data written in the Transmit Data Register has been transferred to the serializer.

TDRE equals zero when the SPI is disabled or at reset. The SPI enable command sets this bit to one.

- **MODF: Mode Fault Error**

0 = No Mode Fault has been detected since the last read of SPI_SR.

1 = A Mode Fault occurred since the last read of the SPI_SR.

- **OVRES: Overrun Error Status**

0 = No overrun has been detected since the last read of SPI_SR.

1 = An overrun has occurred since the last read of SPI_SR.

An overrun occurs when SPI_RDR is loaded at least twice from the serializer since the last read of the SPI_RDR.

- **ENDRX: End of RX buffer**

0 = The Receive Counter Register has not reached 0 since the last write in SPI_RCR(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**

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**

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**

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**

0 = No rising edge detected on NSS pin since last read.

1 = A rising edge occurred on NSS pin since last read.

- **TXEMPTY: Transmission Registers Empty**

0 = As soon as data is written in SPI_TDR.

1 = SPI_TDR and internal shifter are empty. If a transfer delay has been defined, TXEMPTY is set after the completion of such delay.

- **UNDES: Underrun Error Status (Slave Mode Only)**

0 = No underrun has been detected since the last read of SPI_SR.

1 = A transfer begins whereas no data has been loaded in the Transmit Data Register.

- **SPIENS: SPI Enable Status**

0 = SPI is disabled.

1 = SPI is enabled.

Note: 1. SPI_RCR, SPI_RNCR, SPI_TCR, SPI_TNCR are physically located in the PDC.

26.8.6 SPI Interrupt Enable Register

Name: SPI_IER

Address: 0x40008014

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	UNDES	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

0 = No effect.

1 = Enables the corresponding interrupt.

- **RDRF: Receive Data Register Full Interrupt Enable**
- **TDRE: SPI Transmit Data Register Empty Interrupt Enable**
- **MODF: Mode Fault Error Interrupt Enable**
- **OVRES: Overrun Error Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable**
- **ENDTX: End of Transmit Buffer Interrupt Enable**
- **RXBUFF: Receive Buffer Full Interrupt Enable**
- **TXBUFE: Transmit Buffer Empty Interrupt Enable**
- **NSSR: NSS Rising Interrupt Enable**
- **TXEMPTY: Transmission Registers Empty Enable**
- **UNDES: Underrun Error Interrupt Enable**

26.8.7 SPI Interrupt Disable Register

Name: SPI_IDR

Address: 0x40008018

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	UNDES	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

0 = No effect.

1 = Disables the corresponding interrupt.

- **RDRF: Receive Data Register Full Interrupt Disable**
- **TDRE: SPI Transmit Data Register Empty Interrupt Disable**
- **MODF: Mode Fault Error Interrupt Disable**
- **OVRES: Overrun Error Interrupt Disable**
- **ENDRX: End of Receive Buffer Interrupt Disable**
- **ENDTX: End of Transmit Buffer Interrupt Disable**
- **RXBUFF: Receive Buffer Full Interrupt Disable**
- **TXBUFE: Transmit Buffer Empty Interrupt Disable**
- **NSSR: NSS Rising Interrupt Disable**
- **TXEMPTY: Transmission Registers Empty Disable**
- **UNDES: Underrun Error Interrupt Disable**

26.8.8 SPI Interrupt Mask Register

Name: SPI_IMR

Address: 0x4000801C

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	UNDES	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

0 = The corresponding interrupt is not enabled.

1 = The corresponding interrupt is enabled.

- **RDRF: Receive Data Register Full Interrupt Mask**
- **TDRE: SPI Transmit Data Register Empty Interrupt Mask**
- **MODF: Mode Fault Error Interrupt Mask**
- **OVRES: Overrun Error Interrupt Mask**
- **ENDRX: End of Receive Buffer Interrupt Mask**
- **ENDTX: End of Transmit Buffer Interrupt Mask**
- **RXBUFF: Receive Buffer Full Interrupt Mask**
- **TXBUFE: Transmit Buffer Empty Interrupt Mask**
- **NSSR: NSS Rising Interrupt Mask**
- **TXEMPTY: Transmission Registers Empty Mask**
- **UNDES: Underrun Error Interrupt Mask**

26.8.9 SPI Chip Select Register

Name: SPI_CSRx[x=0..3]

Address: 0x40008030

Access: Read/Write

31	30	29	28	27	26	25	24
DLYBCT							
23	22	21	20	19	18	17	16
DLYBS							
15	14	13	12	11	10	9	8
SCBR							
7	6	5	4	3	2	1	0
BITS				CSAAT	CSNAAT	NCPHA	CPOL

This register can only be written if the WPEN bit is cleared in "SPI Write Protection Mode Register".

Note: SPI_CSRx registers must be written even if the user wants to use the defaults. The BITS field will not be updated with the translated value unless the register is written.

- **CPOL: Clock Polarity**

0 = The inactive state value of SPCK is logic level zero.

1 = The inactive state value of SPCK is logic level one.

CPOL is used to determine the inactive state value of the serial clock (SPCK). It is used with NCPHA to produce the required clock/data relationship between master and slave devices.

- **NCPHA: Clock Phase**

0 = Data is changed on the leading edge of SPCK and captured on the following edge of SPCK.

1 = Data is captured on the leading edge of SPCK and changed on the following edge of SPCK.

NCPHA determines which edge of SPCK causes data to change and which edge causes data to be captured. NCPHA is used with CPOL to produce the required clock/data relationship between master and slave devices.

- **CSNAAT: Chip Select Not Active After Transfer (Ignored if CSAAT = 1)**

0 = The Peripheral Chip Select does not rise between two transfers if the SPI_TDR is reloaded before the end of the first transfer and if the two transfers occur on the same Chip Select.

1 = The Peripheral Chip Select rises systematically after each transfer performed on the same slave. It remains active after the end of transfer for a minimal duration of:

  – $\frac{DLYBCT}{MCK}$ (if DLYBCT field is different from 0)

  – $\frac{DLYBCT + 1}{MCK}$ (if DLYBCT field equals 0)

- **CSAAT: Chip Select Active After Transfer**

0 = The Peripheral Chip Select Line rises as soon as the last transfer is achieved.

1 = The Peripheral Chip Select does not rise after the last transfer is achieved. It remains active until a new transfer is requested on a different chip select.

- **BITS: Bits Per Transfer**

(See the (Note:) below the register table; Section 26.8.9 “SPI Chip Select Register” on page 550.)

The BITS field determines the number of data bits transferred. Reserved values should not be used.

Value	Name	Description
0	8_BIT	8 bits for transfer
1	9_BIT	9 bits for transfer
2	10_BIT	10 bits for transfer
3	11_BIT	11 bits for transfer
4	12_BIT	12 bits for transfer
5	13_BIT	13 bits for transfer
6	14_BIT	14 bits for transfer
7	15_BIT	15 bits for transfer
8	16_BIT	16 bits for transfer
9	–	Reserved
10	–	Reserved
11	–	Reserved
12	–	Reserved
13	–	Reserved
14	–	Reserved
15	–	Reserved

- **SCBR: Serial Clock Baud Rate**

In Master Mode, the SPI Interface uses a modulus counter to derive the SPCK baud rate from the Master Clock MCK. The Baud rate is selected by writing a value from 1 to 255 in the SCBR field. The following equations determine the SPCK baud rate:

$$\text{SPCK Baudrate} = \frac{MCK}{SCBR}$$

Programming the SCBR field at 0 is forbidden. Triggering a transfer while SCBR is at 0 can lead to unpredictable results.

At reset, SCBR is 0 and the user has to program it at a valid value before performing the first transfer.

Note: If one of the SCBR fields inSPI_CSRx is set to 1, the other SCBR fields in SPI_CSRx must be set to 1 as well, if they are required to process transfers. If they are not used to transfer data, they can be set at any value.

- **DLYBS: Delay Before SPCK**

This field defines the delay from NPCS valid to the first valid SPCK transition.

When DLYBS equals zero, the NPCS valid to SPCK transition is 1/2 the SPCK clock period.

Otherwise, the following equations determine the delay:

$$\text{Delay Before SPCK} = \frac{DLYBS}{MCK}$$

- **DLYBCT: Delay Between Consecutive Transfers**

This field defines the delay between two consecutive transfers with the same peripheral without removing the chip select. The delay is always inserted after each transfer and before removing the chip select if needed.

When DLYBCT equals zero, no delay between consecutive transfers is inserted and the clock keeps its duty cycle over the character transfers.

Otherwise, the following equation determines the delay:

$$\text{Delay Between Consecutive Transfers} = \frac{32 \times DLYBCT}{MCK}$$

26.8.10 SPI Write Protection Mode Register

Name: SPI_WPMR

Address: 0x400080E4

Access: Read-write

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

- **WPEN: Write Protect Enable**

0: Disables the Write Protect if WPKEY corresponds to 0x535049 (“SPI” in ASCII).

1: Enables the Write Protect if WPKEY corresponds to 0x535049 (“SPI” in ASCII).

Protects the registers:

- Section 26.8.2 ”SPI Mode Register”
- Section 26.8.9 ”SPI Chip Select Register”

- **WPKEY: Write Protect Key**

Value	Name	Description
0x535049	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

26.8.11 SPI Write Protection Status Register

Name: SPI_WPSR

Address: 0x400080E8

Access: Read-only

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

- **WPVS: Write Protection Violation Status**

0 = No Write Protect Violation has occurred since the last read of the SPI_WPSR register.

1 = A Write Protect Violation has occurred since the last read of the SPI_WPSR register. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protection Violation Source**

This Field indicates the APB Offset of the register concerned by the violation (SPI_MR or SPI_CSRx)

27. Two-Wire Interface (TWIHS)

27.1 Description

The Atmel Two-wire Interface (TWIHS) 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 in fast mode and up to 3.4 Mbits per second in high-speed slave mode only, based on a byte-oriented transfer format. It can be used with any Atmel Two-wire Interface bus Serial EEPROM and I²C-compatible devices, such as a Real-Time Clock (RTC), Dot Matrix/Graphic LCD Controller and temperature sensor. The TWI is programmable as a master or a slave with sequential or single-byte access. Multiple master capability is supported.

Arbitration of the bus is performed internally and puts the TWI in slave mode automatically if the bus arbitration is lost.

A configurable baud rate generator permits the output data rate to be adapted to a wide range of core clock frequencies.

Table 27-1 lists the compatibility level of the Atmel Two-wire Interface in Master mode and a full I^2C compatible device.

Table 27-1. Atmel TWI Compatibility with I^2C Standard

I^2C Standard	Atmel TWI
Standard Mode Speed (100 kHz)	Supported
Fast Mode Speed (400 kHz)	Supported
High-speed Mode (Slave only, 3.4 MHz)	Supported
7- or 10-bit(1) Slave Addressing	Supported
START BYTE(2)	Not Supported
Repeated Start (Sr) Condition	Supported
ACK and NACK Management	Supported
Input Filtering	Supported
Slope Control	Not Supported
Clock Stretching	Supported
Multi Master Capability	Supported

Notes: 1. 10-bit support in Master mode only
2. START + b000000001 + Ack + Sr

27.2 Embedded Characteristics

- 1 TWIHS
- 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 and Multi-Master Operation (Standard and Fast Mode Only)
- Slave Mode Operation (Standard, Fast and High-Speed Mode)
- Bit Rate: Up to 400 Kbit/s in Fast Mode and 3.4 Mbit/s in High-Speed Mode (Slave Only)
- 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 27-1 for details on compatibility with I²C Standard.

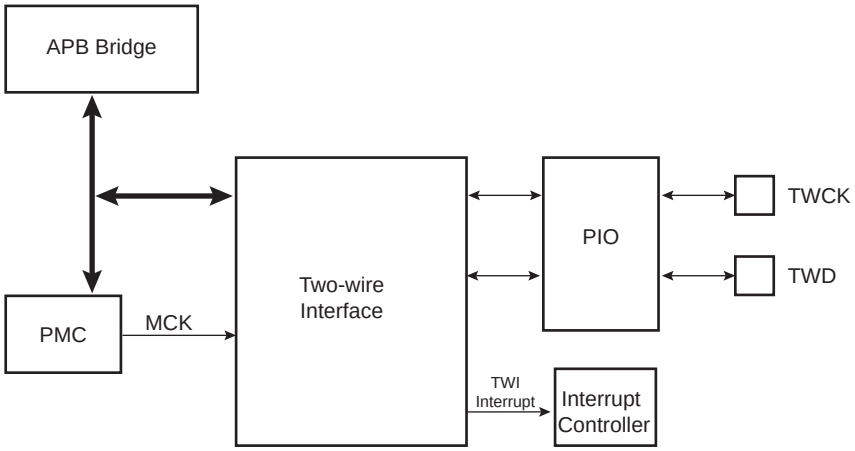
27.3 List of Abbreviations

Table 27-2. 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

27.4 Block Diagram

Figure 27-1. Block Diagram



27.5 Application Block Diagram

Figure 27-2. High-Speed Mode Application Block Diagram

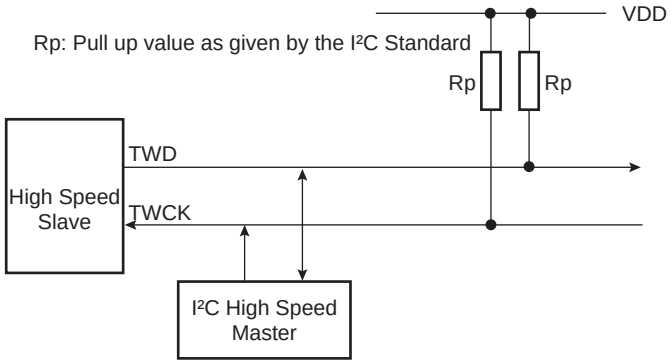
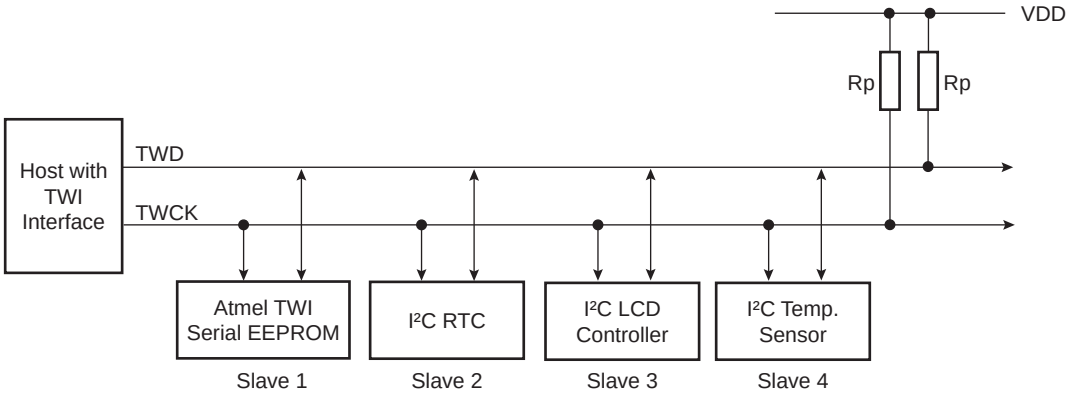


Figure 27-3. Standard and Fast Mode Application Block Diagram



Rp: Pull up value as given by the I²C Standard

27.5.1 I/O Lines Description

Table 27-3. I/O Lines Description

Pin Name	Pin Description	Type
TWD	Two-Wire Serial Data	Input/Output
TWCK	Two-Wire Serial Clock	Input/Output

27.6 Product Dependencies

27.6.1 I/O Lines

Both TWD and TWCK are bidirectional lines, connected to a positive supply voltage via a current source or pull-up resistor (see Figure 27-3 ”Standard and Fast Mode Application Block Diagram”). 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. When high-speed slave mode is enabled, the analog pad filter must be enabled.

The user must not program TWD and TWCK as open-drain. It is already done by the hardware.

Table 27-4. I/O Lines

Instance	Signal	I/O Line	Peripheral
TWI0	TWCK0	PA4	A
TWI0	TWD0	PA3	A

27.6.2 Power Management

Enable the peripheral clock.

The TWIHS interface may be clocked through the Power Management Controller (PMC), thus the user must first configure the PMC to enable the TWI clock.

27.6.3 Interrupt

The TWIHS interface has an interrupt line connected to the Interrupt Controller. In order to handle interrupts, the Interrupt Controller must be programmed before configuring the TWI.

Table 27-5. Peripheral IDs

Instance	ID
TWI0	19

27.7 Functional Description

27.7.1 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 27-5).

Each transfer begins with a START condition and terminates with a STOP condition (see Figure 27-4).

- A high-to-low transition on the TWD line while TWCK is high defines the START condition.
- A low-to-high transition on the TWD line while TWCK is high defines a STOP condition.

Figure 27-4. START and STOP Conditions

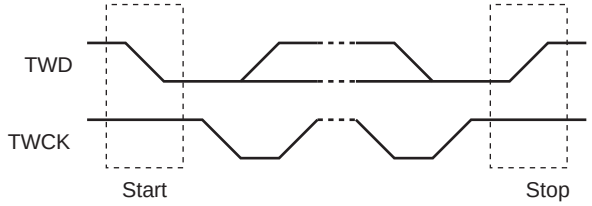
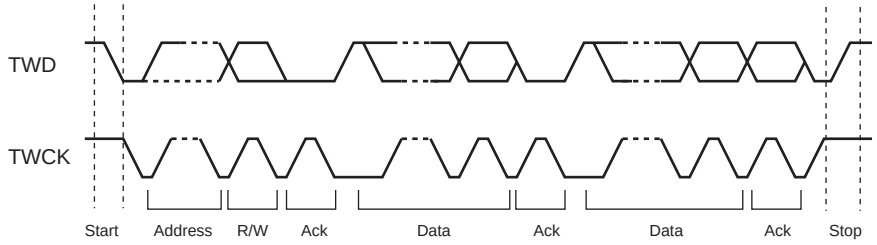


Figure 27-5. Transfer Format



27.7.2 Modes of Operation

The TWIHS has different modes of operation:

- Master transmitter mode (standard and fast mode only)
- Master receiver mode (standard and fast mode only)
- Multi-master transmitter mode (standard and fast mode only)
- Multi-master receiver mode (standard and fast mode only)
- Slave transmitter mode (standard, fast and high-speed mode)
- Slave receiver mode (standard, fast and high-speed mode)

These modes are described in the following sections.

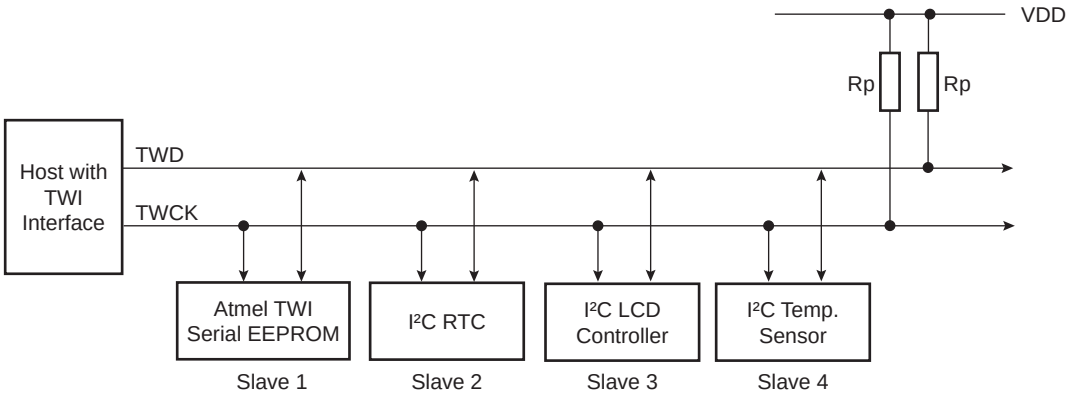
27.7.3 Master Mode

27.7.3.1 Definition

The master is the device that starts a transfer, generates a clock and stops it. This operating mode is not available if high-speed mode is selected.

27.7.3.2 Application Block Diagram

Figure 27-6. Master Mode Typical Application Block Diagram



Rp: Pull up value as given by the I²C Standard

27.7.3.3 Programming Master Mode

The following registers must be programmed before entering master mode:

1. DADR (+ IADRSZ + IADR if a 10-bit device is addressed): The device address is used to access slave devices in read or write mode.
2. CKDIV + CHDIV + CLDIV: Clock waveform.
3. SVDIS: Disables the slave mode.
4. 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

27.7.3.4 Master Transmitter Mode

This operating mode is not available if high-speed mode is selected.

After the master initiates a START condition when writing into the Transmit Holding register TWIHS_THR, it sends a 7-bit slave address, configured in the Master Mode register (DADR in TWIHS_MMR), to notify the slave device. The bit following the slave address indicates the transfer direction, 0 in this case (MREAD = 0 in TWIHS_MMR).

The TWI transfers require the slave to acknowledge each received byte. During the acknowledge clock pulse (ninth pulse), the master releases the data line (HIGH), enabling the slave to pull it down in order to generate the acknowledge. The master polls the data line during this clock pulse and sets the Not Acknowledge bit (NACK**)** in the status register if the slave does not acknowledge the byte. As with the other status bits, an interrupt can be generated if enabled in the interrupt enable register (TWIHS_IER). If the slave acknowledges the byte, the data written in the TWIHS_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 TWIHS_THR.

TXRDY is used as transmit ready for the PDC transmit channel.

While no new data is written in the TWIHS_THR, the serial clock line is tied low. When new data is written in the TWIHS_THR, the SCL is released and the data is sent. To generate a STOP event, the STOP command must be performed by writing in the STOP field of TWIHS_CR.

After a master write transfer, the Serial Clock line is stretched (tied low) while no new data is written in the TWIHS_THR or until a STOP command is performed.

See Figure 27-7, Figure 27-8, and Figure 27-9.

Figure 27-7. Master Write with One Data Byte

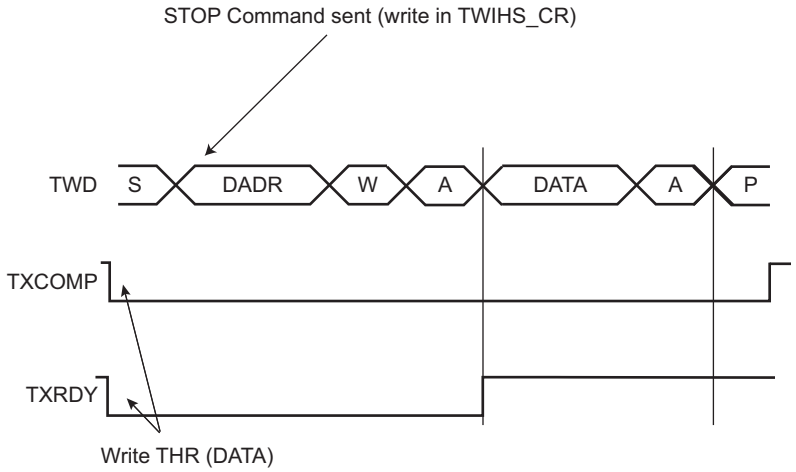


Figure 27-8. Master Write with Multiple Data Bytes

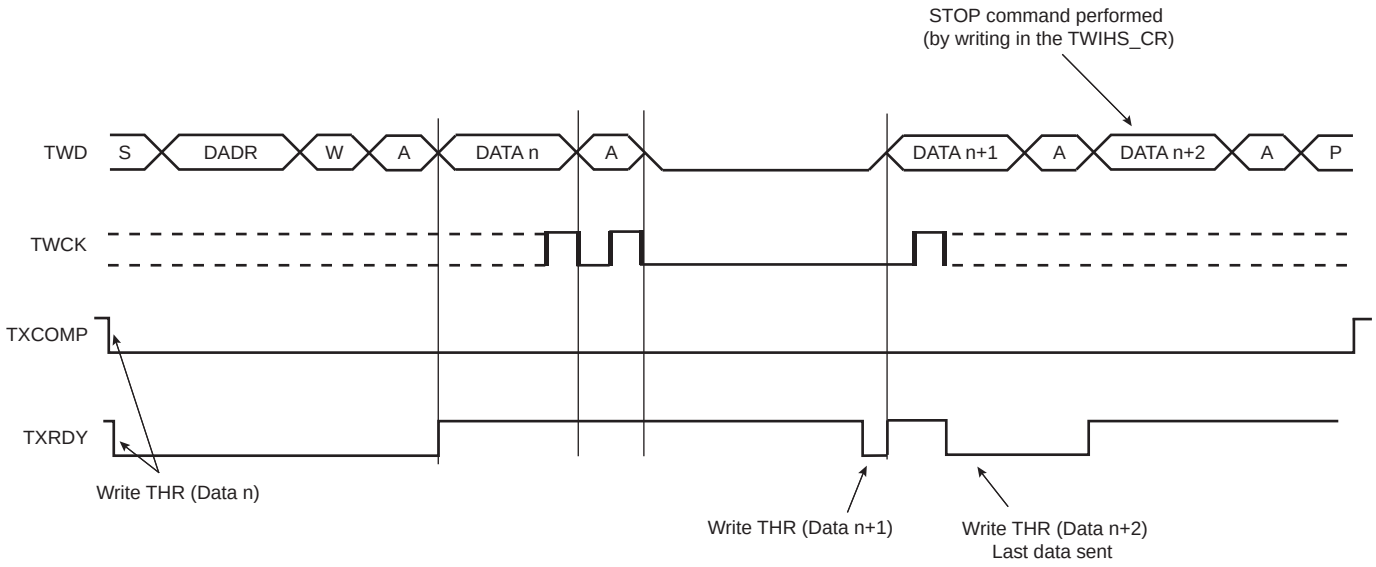
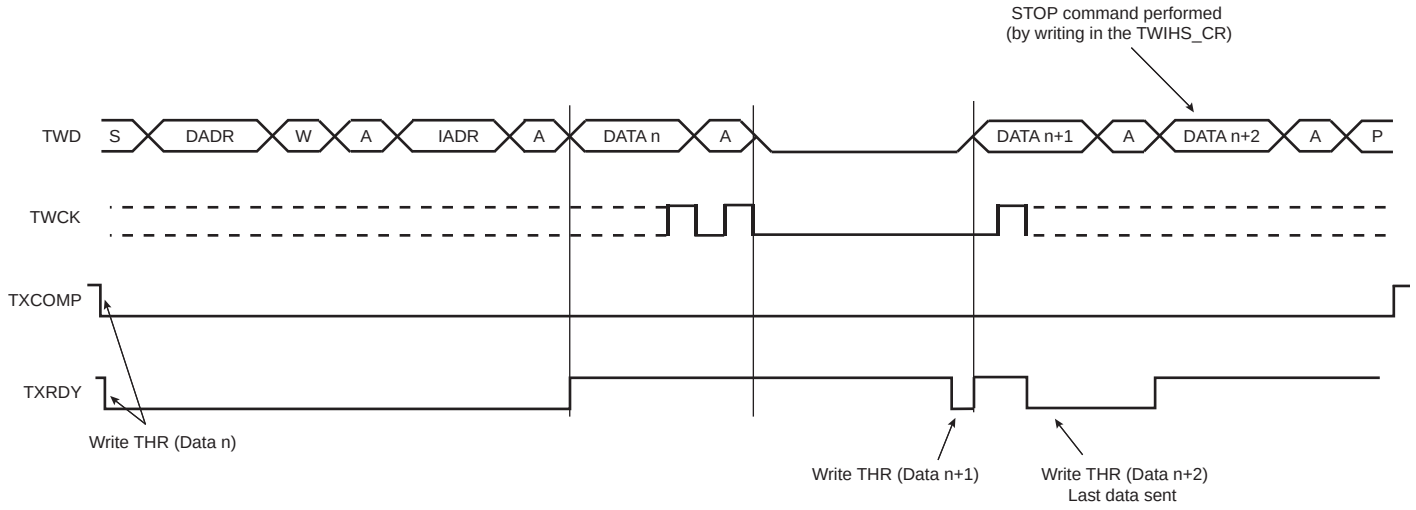


Figure 27-9. Master Write with One Byte Internal Address and Multiple Data Bytes



27.7.3.5 Master Receiver Mode

Master receiver mode is not available if high-speed mode is selected.

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 TWIHS_MMR). During the acknowledge clock pulse (9th pulse), the master releases the data line (HIGH), enabling the slave to pull it down in order to generate the acknowledge. The master polls the data line during this clock pulse and sets the NACK bit in the status register if the slave does not acknowledge the byte.

If an acknowledge is received, the master is then ready to receive data from the slave. After data has been received, the master sends an acknowledge condition to notify the slave that the data has been received except for the last data (see Figure 27-10). When the RXRDY bit is set in the status register, a character has been received in the receive-holding register (TWIHS_RHR). The RXRDY bit is reset when reading the TWIHS_RHR.

When a single data byte read is performed, with or without internal address **(**IADR), the START and STOP bits must be set at the same time. See Figure 27-10 ”Master Read with One Data Byte”. 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 (same condition applies for START bit to generate a repeated start). See Figure 27-11 ”Master Read with Multiple Data Bytes”. For internal address usage, see Section 27.7.3.6 ”Internal Address”.

If the receive holding register (TWIHS_RHR) is full (RXRDY high) and the master is receiving data, the serial clock line will be tied low before receiving the last bit of the data and until the TWIHS_RHR is read. Once the TWIHS_RHR is read, the master will stop stretching the serial clock line and end the data reception, see Figure 27-12.

Warning: When receiving multiple bytes in master read mode, if the next-to-last access is not read (the RXRDY flag remains high), the last access will not be completed until TWIHS_RHR is read. The last access stops on the next-to-last bit (clock stretching). When the TWIHS_RHR is read there is only half a bit period to send the STOP bit (or START bit) command, else another read access might occur (spurious access).

A possible workaround is to raise the STOP bit (or START bit) command before reading the TWIHS_RHR on the next-to-last access (within IT handler).

Figure 27-10. Master Read with One Data Byte

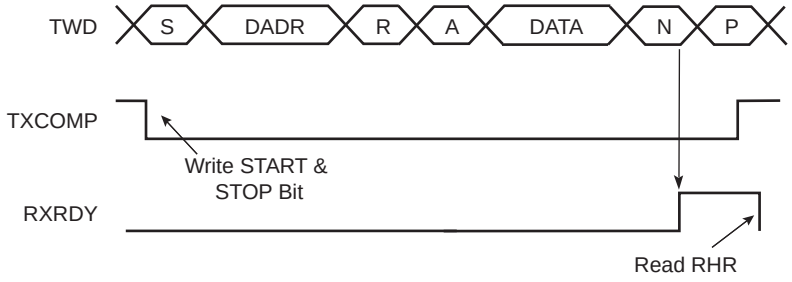


Figure 27-11. Master Read with Multiple Data Bytes

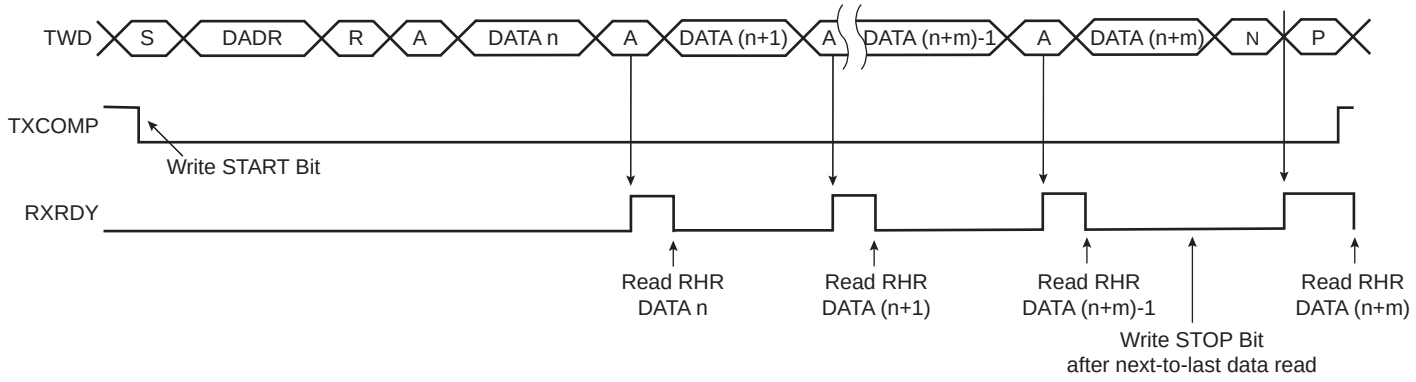
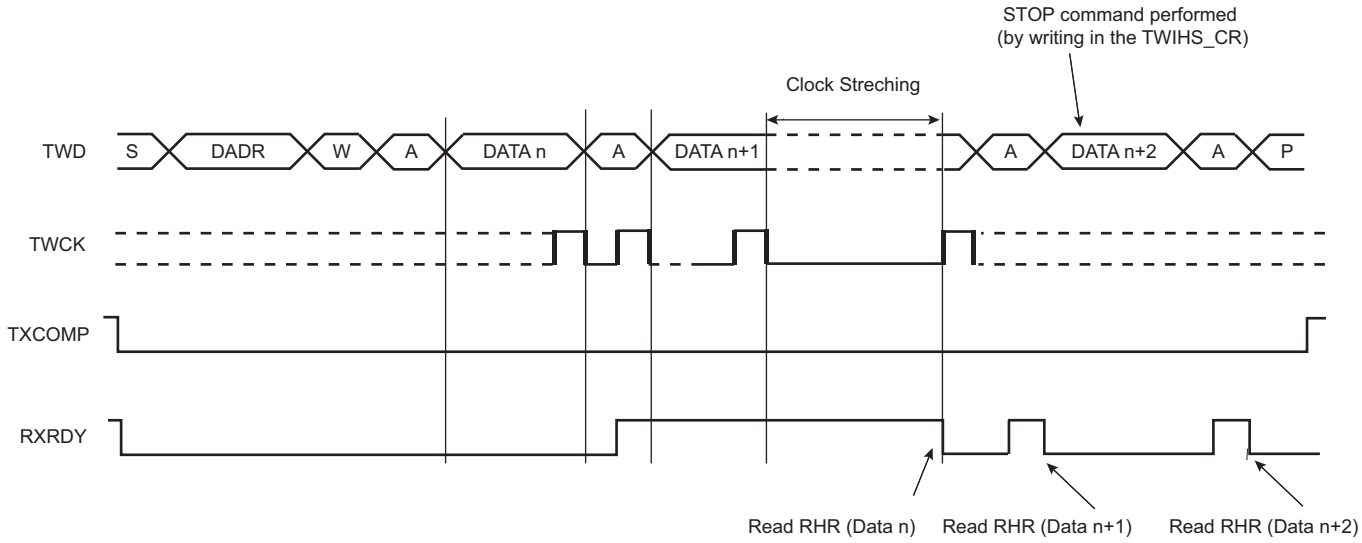


Figure 27-12. Master Read Clock Stretching with Multiple Data Bytes



RXRDY is used as receive ready for the PDC receive channel.

27.7.3.6 Internal Address

The TWIHS interface can perform transfers with 7-bit slave address devices and with 10-bit slave address devices.

7-bit Slave Addressing

When addressing 7-bit slave devices, the internal address bytes are used to perform random address (read or write) accesses to reach one or more data bytes, e.g. within a memory page location in a serial memory. When performing read operations with an internal address, the TWI performs a write operation to set the internal address into the slave device,

and then switch to master receiver mode. Note that the second start condition (after sending the IADR) is sometimes called “repeated start” (Sr) in I²C fully-compatible devices. See Figure 27-14 ”Master Read with One, Two or Three Bytes Internal Address and One Data Byte”.

See Figure 27-13 ”Master Write with One, Two or Three Bytes Internal Address and One Data Byte” and Figure 27-15 ”Internal Address Usage” for the master write operation with internal address.

The three internal address bytes are configurable through the Master Mode Register (TWIHS_MMR).

If the slave device supports only a 7-bit address, i.e. no internal address, IADRSZ must be set to 0.

In the figures below the following abbreviations are used:

S	Start
Sr	Repeated Start
P	Stop
W	Write
R	Read
A	Acknowledge
N	Not Acknowledge
DADR	Device Address
IADR	Internal Address

Figure 27-13. Master Write with One, Two or Three Bytes Internal Address and One Data Byte

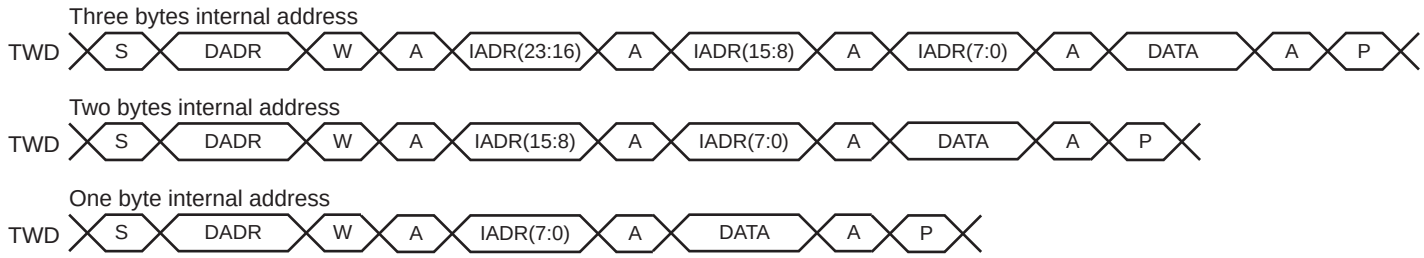
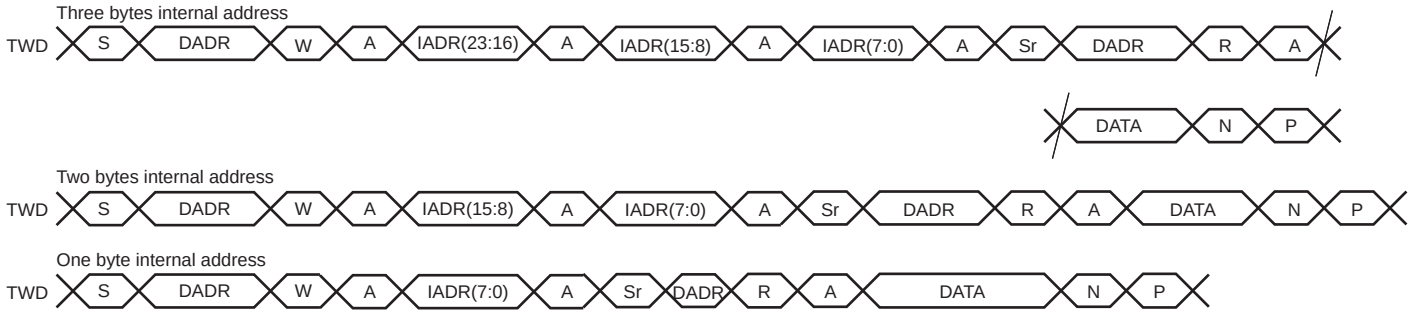


Figure 27-14. Master Read with One, Two or Three Bytes Internal Address and One Data Byte



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 (TWIHS_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.

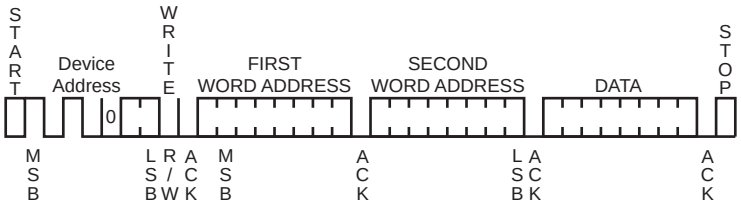
Example: Address a 10-bit device (10-bit device address is b1 b2 b3 b4 b5 b6 b7 b8 b9 b10)

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 TWIHS_IADR with b3 b4 b5 b6 b7 b8 b9 b10 (b10 is the LSB of the 10-bit address)

Figure 27-15 shows a byte write to an Atmel AT24LC512 EEPROM. This demonstrates the use of internal addresses to access the device.

Figure 27-15. Internal Address Usage



27.7.3.7 Repeated Start

In addition to internal address mode, repeated start (Sr) can be generated manually by writing the START bit at the end of a transfer instead of the STOP bit. In such case the parameters of the next transfer (direction, SADR, etc.) will need to be set before writing the START bit at the end of the previous transfer.

See Section 27.7.3.12 for detailed flowcharts.

Note that generating a repeated start after a single data read is not supported.

27.7.3.8 Bus Clear Command

The TWI interface can perform a Bus Clear Command:

1. Configure the master mode (DADR, CKDIV, etc).
2. Start the transfer by setting the CLEAR bit in the TWIHS_CR.

27.7.3.9 Using the Peripheral DMA Controller (PDC) in Master Mode

The use of the PDC significantly reduces the CPU load.

To assure correct implementation, respect the following programming sequences:

Data Transmit with the PDC in Master Mode

The PDC transfer size must be defined with the buffer size minus 1. The remaining character must be managed without PDC to ensure that the exact number of bytes are transmitted regardless of system bus latency conditions during the end of the buffer transfer period.

1. Initialize the transmit PDC (memory pointers, transfer size - 1).
2. Configure the master mode (DADR, CKDIV, 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 TWIHS_SR.
7. Set the STOP command in TWIHS_CR.
8. Write the last character in TWIHS_THR.
9. (Optional) Wait for the TXCOMP flag in TWIHS_SR before disabling the peripheral clock if required.

Data Receive with the PDC in Master Mode

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 during the end of the buffer transfer period.

1. Initialize the receive PDC (memory pointers, transfer size - 2).
2. Configure the master mode (DADR, CKDIV, etc.).
3. Set the PDC RXTEN bit.

4. (Master Only) Write the START bit in the TWIHS_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 TWIHS_SR.
8. Set the STOP command in TWIHS_CR to end the transfer.
9. Read the penultimate character in TWIHS_RHR.
10. Wait for the RXRDY flag in TWIHS_SR.
11. Read the last character in TWIHS_RHR.
12. (Optional) Wait for the TXCOMP flag in TWIHS_SR before disabling the peripheral clock if required.

27.7.3.10 SMBus Mode

SMBus mode is enabled when the SMEN bit is written to one in the TWIHS_CR. SMBus mode operation is similar to I²C operation with the following exceptions:

1. Only 7-bit addressing can be used.
2. The SMBus standard describes a set of timeout values to ensure progress and throughput on the bus. These time-out values must be programmed into TWIHS_SMBTR.
3. Transmissions can optionally include a CRC byte, called Packet Error Check (PEC).
4. A dedicated bus line, SMBALERT, allows a slave to get a master attention.
5. A set of addresses has been reserved for protocol handling, such as alert response address (ARA) and host header (HH) address. Address matching on these addresses can be enabled by configuring the TWIHS_CR appropriately.

Packet Error Checking

Each SMBus transfer can optionally end with a CRC byte, called the PEC byte. Writing the PECEN bit in TWIHS_CR to one enables automatic PEC handling in the current transfer. Transfers with and without PEC can freely be intermixed in the same system, since some slaves may not support PEC. The PEC LFSR is always updated on every bit transmitted or received, so that PEC handling on combined transfers will be correct.

In master transmitter mode, the master calculates a PEC value and transmits it to the slave after all data bytes have been transmitted. Upon reception of this PEC byte, the slave will compare it to the PEC value it has computed itself. If the values match, the data was received correctly, and the slave will return an ACK to the master. If the PEC values differ, data was corrupted, and the slave will return a NACK value. Some slaves may not be able to check the received PEC in time to return a NACK if an error occurred. In this case, the slave should always return an ACK after the PEC byte, and some other mechanism must be implemented to verify that the transmission was received correctly.

In master receiver mode, the slave calculates a PEC value and transmits it to the master after all data bytes have been transmitted. Upon reception of this PEC byte, the master will compare it to the PEC value it has computed itself. If the values match, the data was received correctly. If the PEC values differ, data was corrupted, and the PECERR bit in TWIHS_SR is set. In master receiver mode, the PEC byte is always followed by a NACK transmitted by the master, since it is the last byte in the transfer.

In combined transfers, the PECRQ bit should only be set in the last of the combined transfers. .

Consider the following transfer:

S, ADR+W, COMMAND_BYTE, ACK, SR, ADR+R, DATA_BYTE, ACK, PEC_BYTE, NACK, P

See Section 27.7.3.12 for detailed flowcharts.

Timeouts

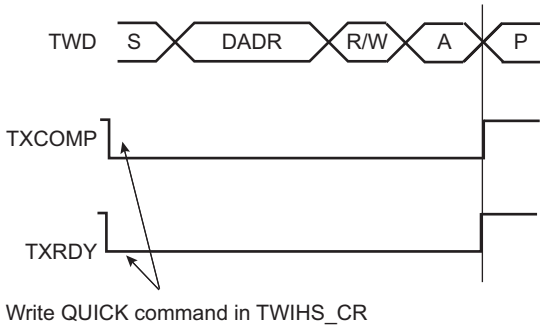
The TLOWS and TLOWM fields in TWIHS_SMBTR configure the SMBus timeout values. If a timeout occurs, the master will transmit a STOP condition and leave the bus. The TOUT bit is also set in TWIHS_SR.

27.7.3.11 SMBUS Quick Command (Master Mode Only)

The TWI interface can perform a quick command:

1. Configure the master mode (DADR, CKDIV, etc).
2. Write the MREAD bit in the TWIHS_MMR at the value of the one-bit command to be sent.
3. Start the transfer by setting the QUICK bit in the TWIHS_CR.

Figure 27-16. SMBUS Quick Command



27.7.3.12 Read/Write Flowcharts

The following flowcharts shown in Figure 27-18, Figure 27-19, Figure 27-24, Figure 27-25 and Figure 27-26 give examples for read and write operations. A polling or interrupt method can be used to check the status bits. The interrupt method requires that the interrupt enable register (TWIHS_IER) be configured first.

Figure 27-17. TWI Write Operation with Single Data Byte without Internal Address

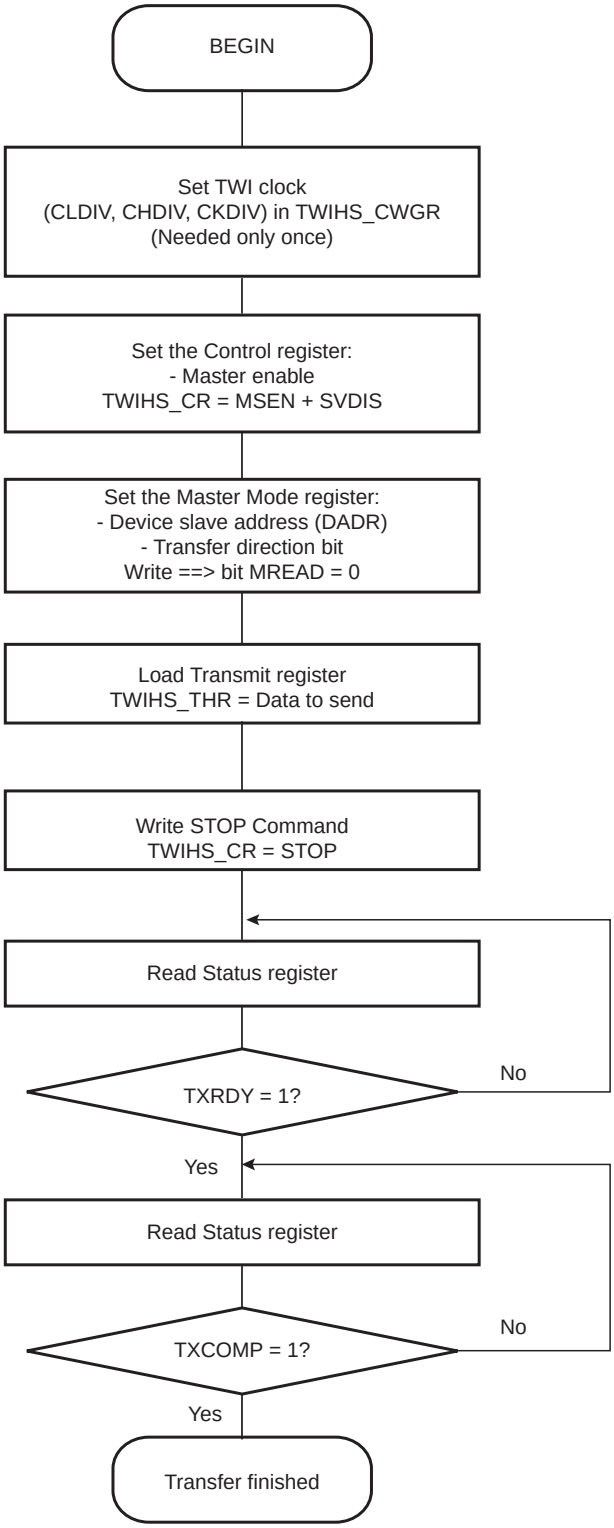


Figure 27-18. TWI Write Operation with Single Data Byte and Internal Address

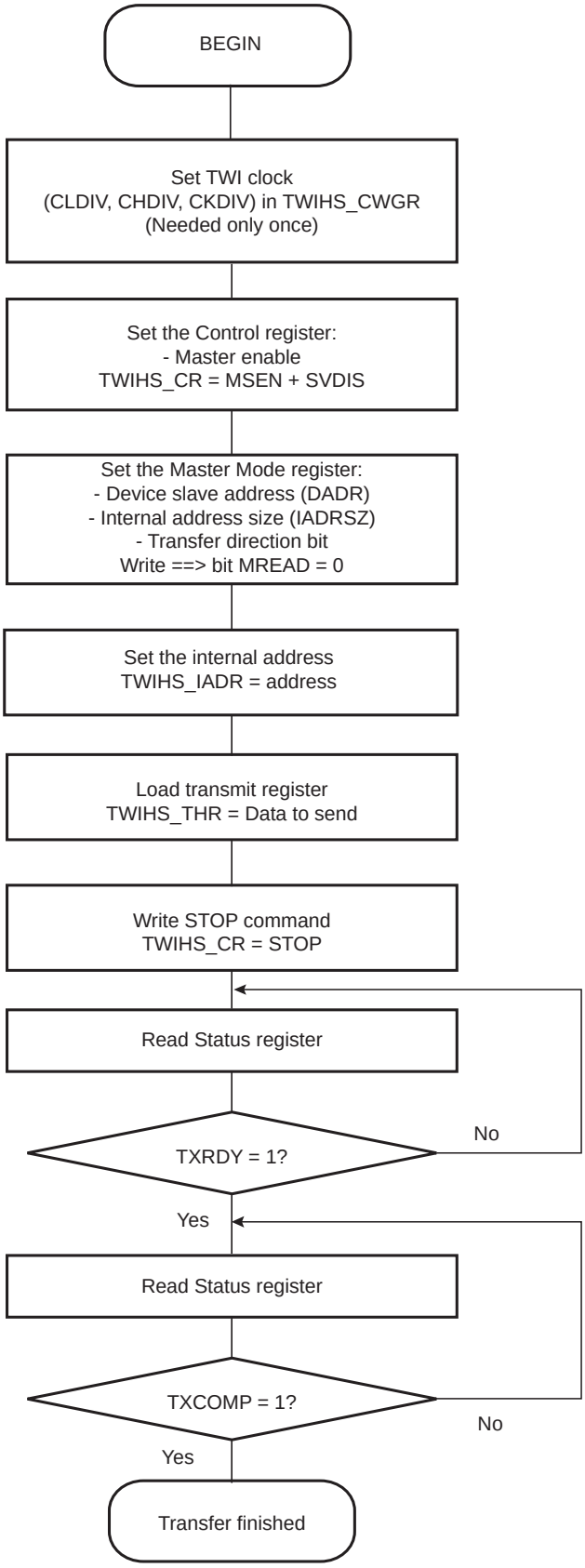


Figure 27-19. TWI Write Operation with Multiple Data Bytes with or without Internal Address

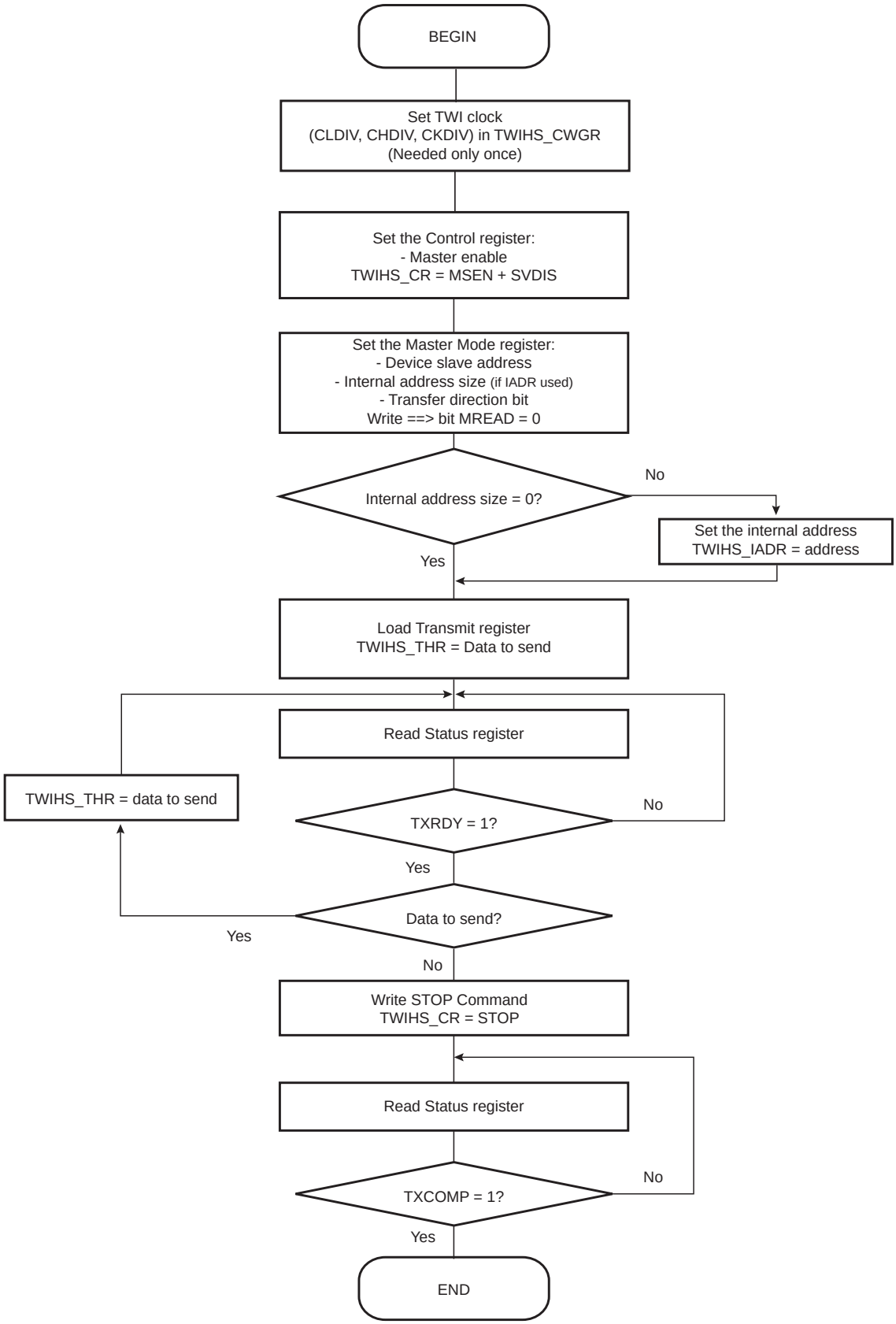


Figure 27-20. SMBus Write Operation with Multiple Data Bytes with or without Internal Address and PEC Sending

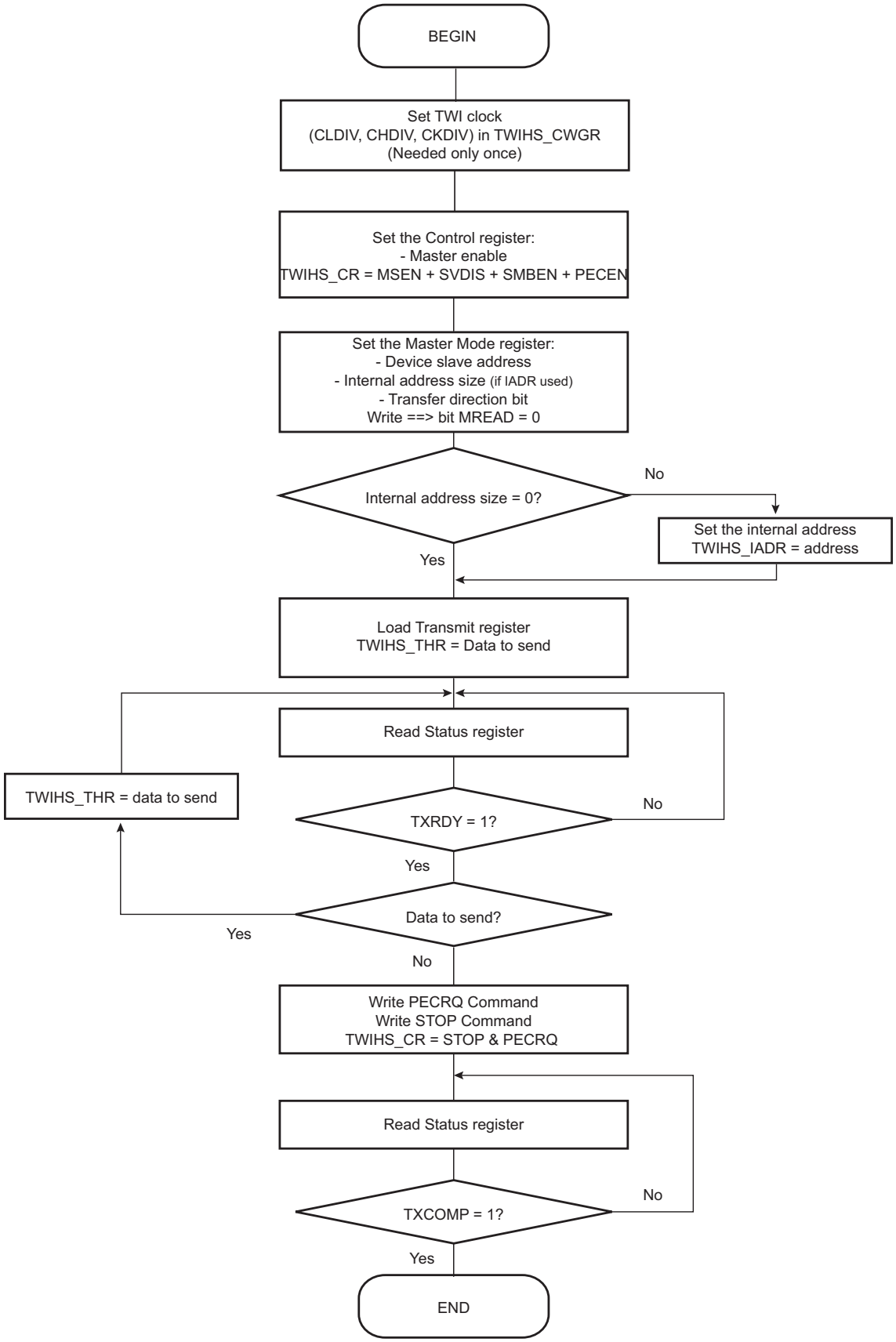


Figure 27-21. SMBus Write Operation with Multiple Data Bytes with PEC and Alternative Command Mode

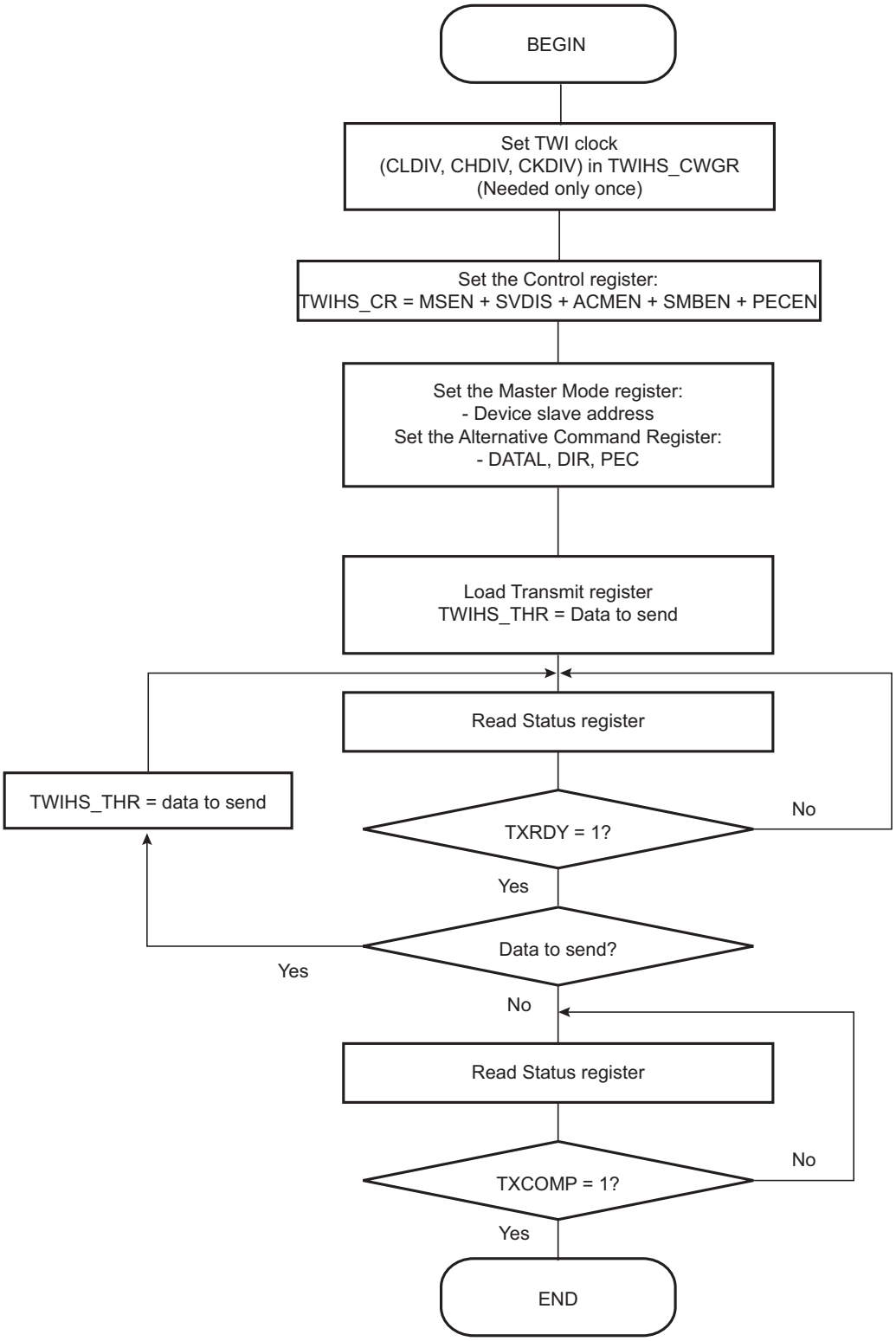


Figure 27-22. TWI Write Operation with Multiple Data Bytes and Read Operation with Multiple Data Bytes (Sr)

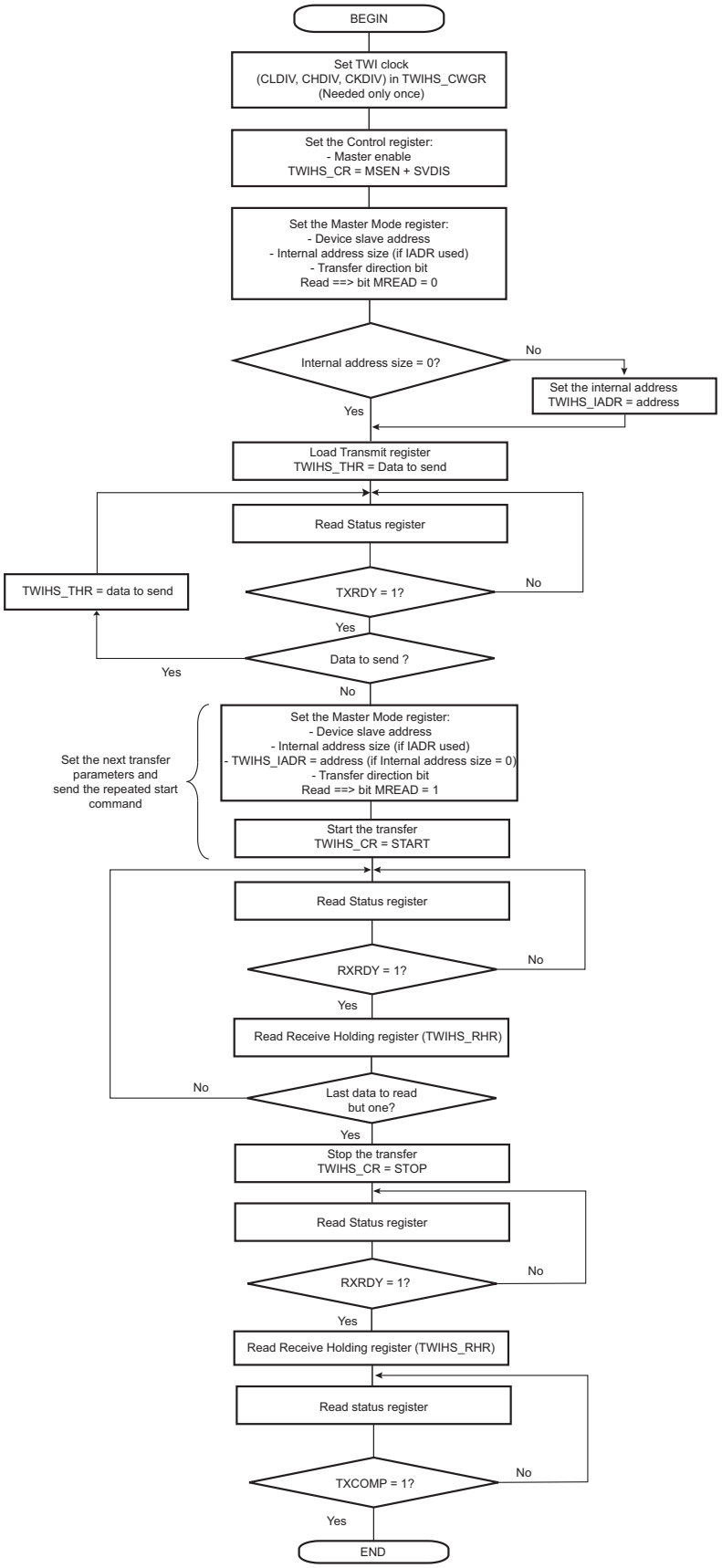


Figure 27-23. TWI Write Operation with Multiple Data Bytes + Read Operation and Alternative Command Mode + PEC

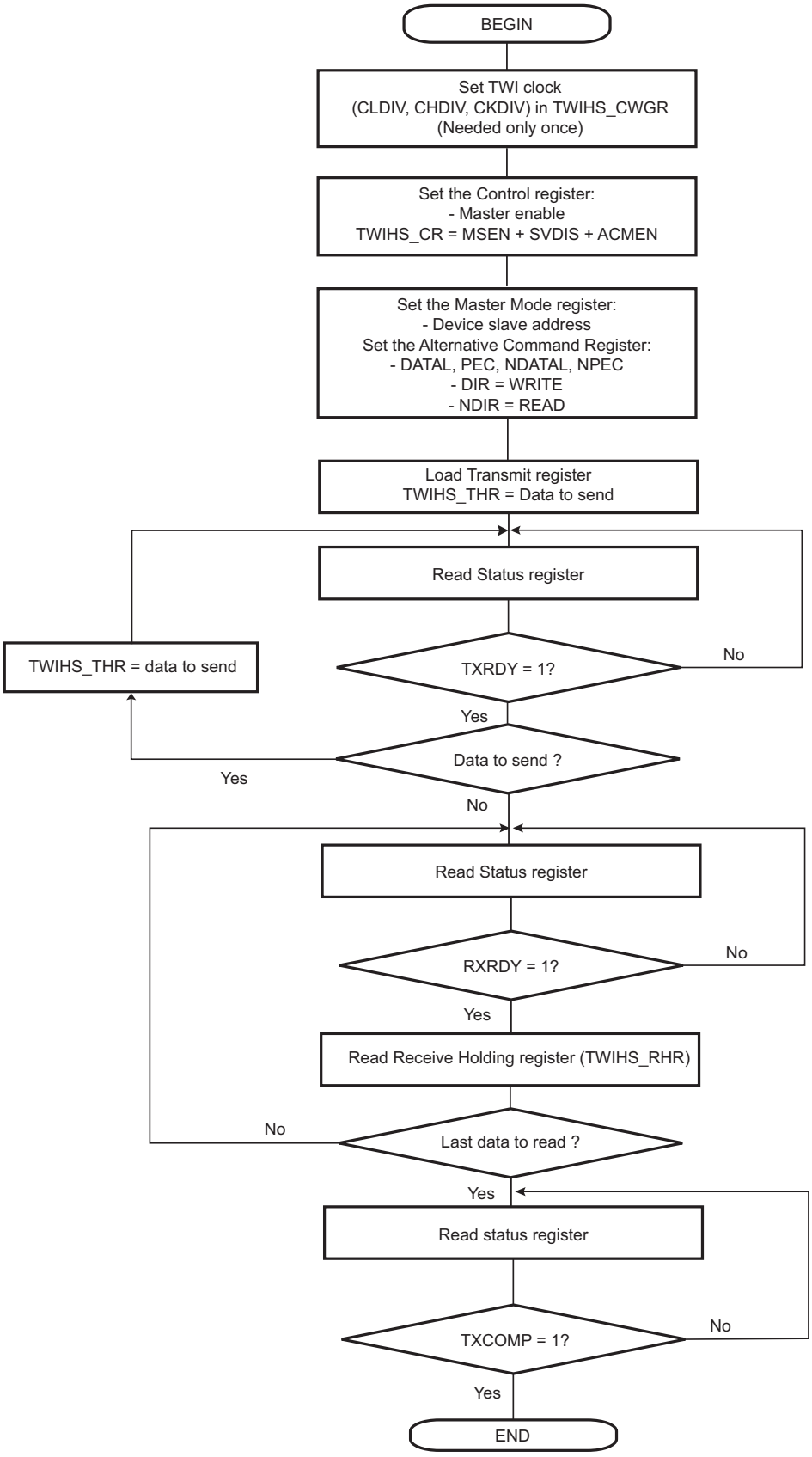


Figure 27-24. TWI Read Operation with Single Data Byte without Internal Address

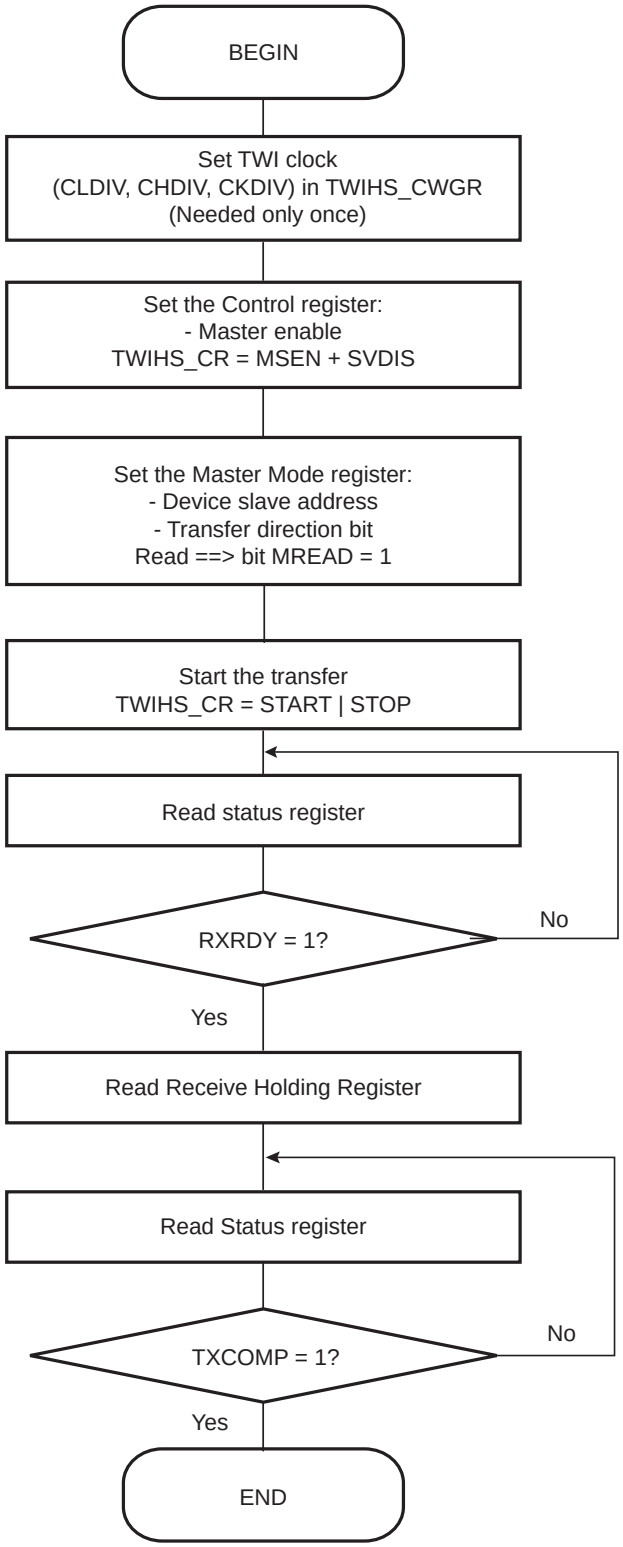


Figure 27-25. TWI Read Operation with Single Data Byte and Internal Address

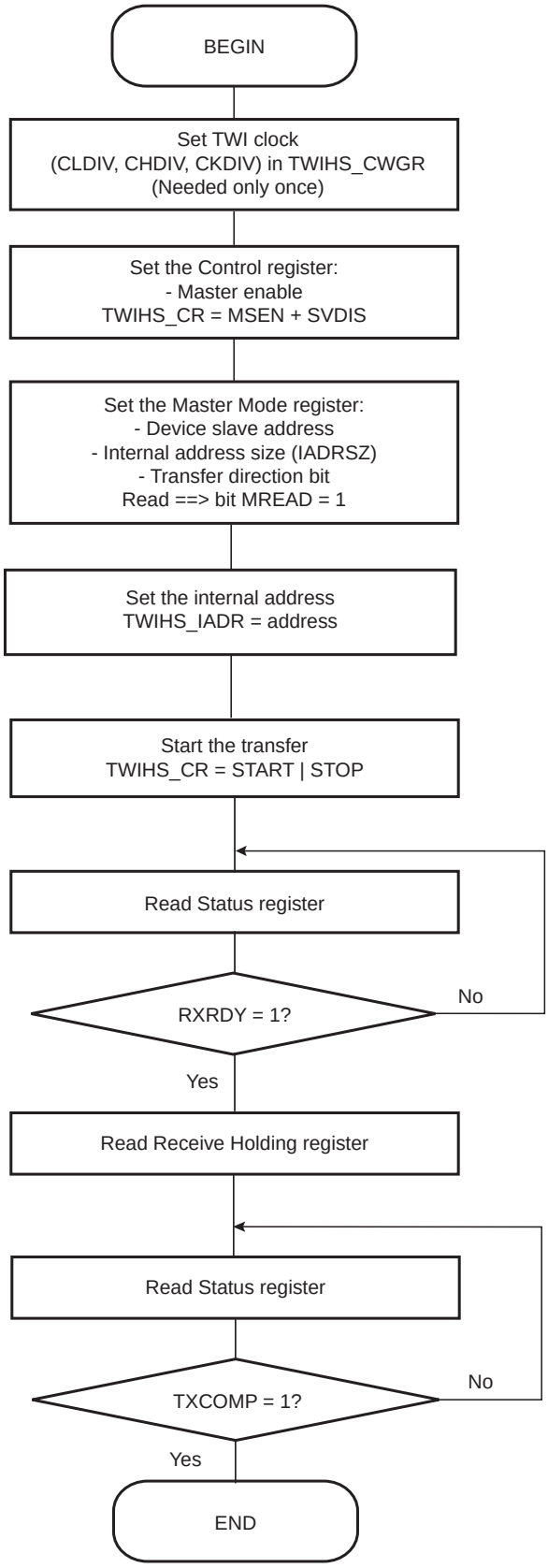


Figure 27-26. TWI Read Operation with Multiple Data Bytes with or without Internal Address

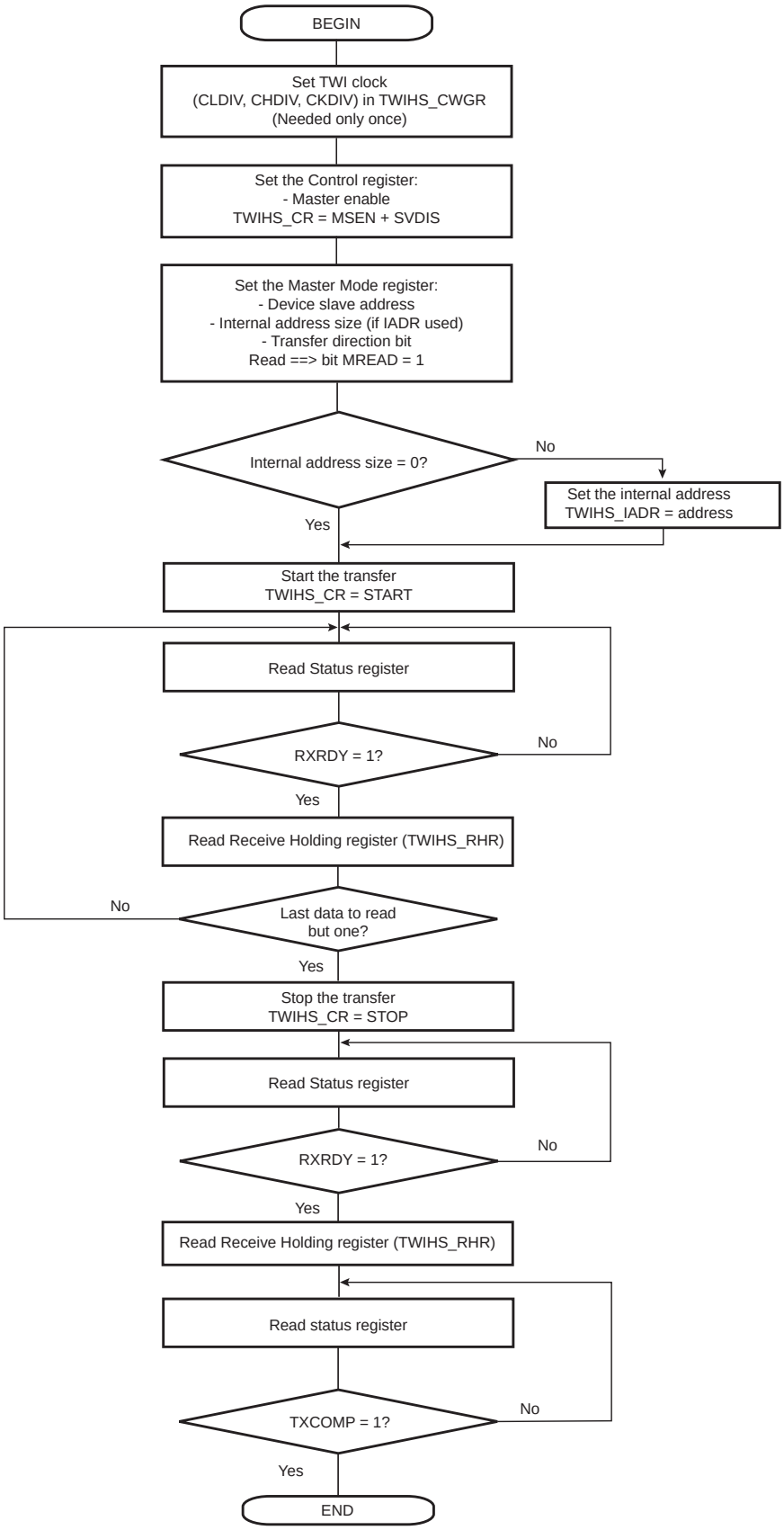


Figure 27-27. TWI Read Operation with Multiple Data Bytes with or without Internal Address with PEC

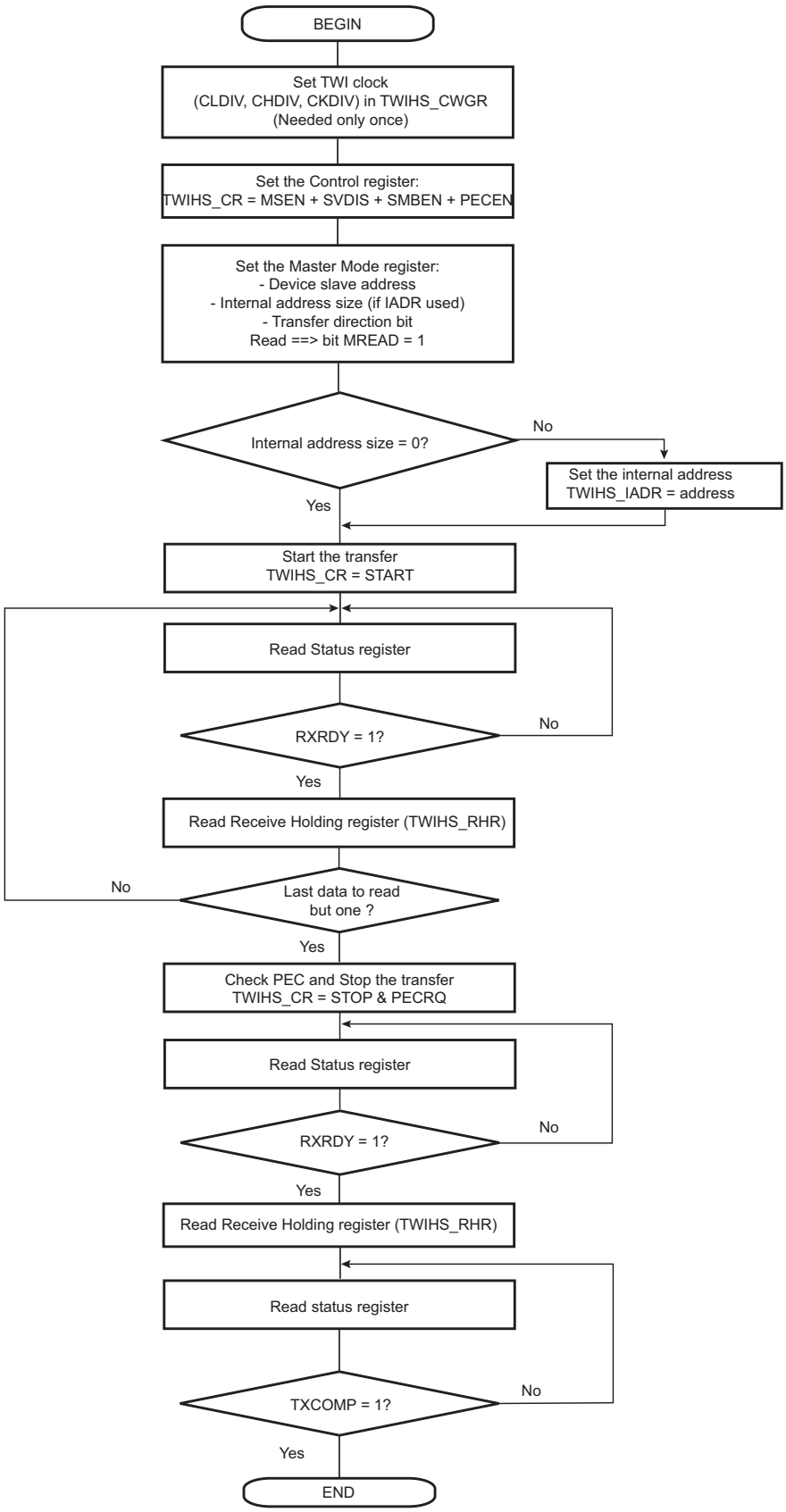


Figure 27-28. TWI Read Operation with Multiple Data Bytes with Alternative Command Mode with PEC

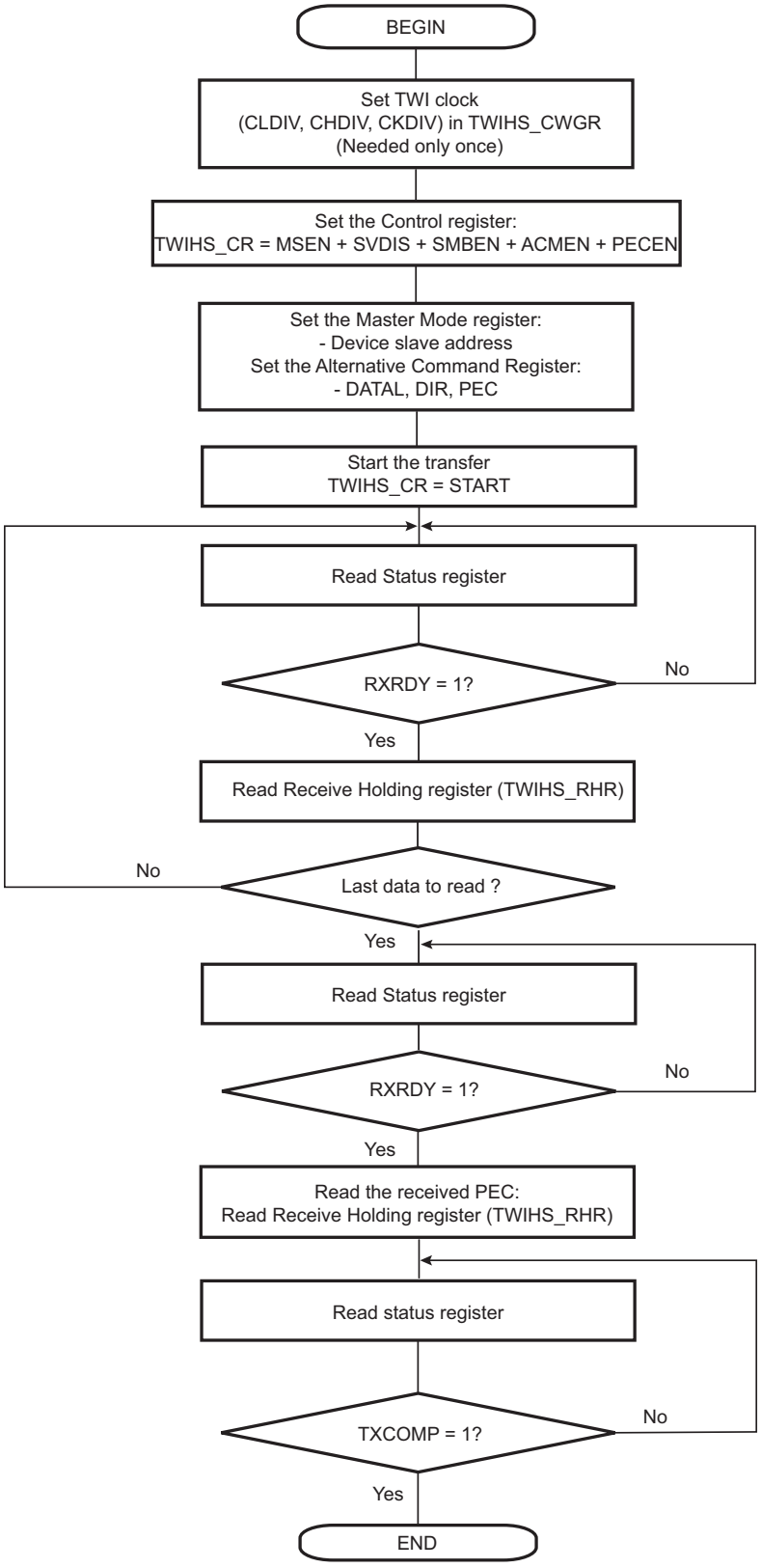


Figure 27-29. TWI Read Operation with Multiple Data Bytes + Write Operation with Multiple Data Bytes (Sr)

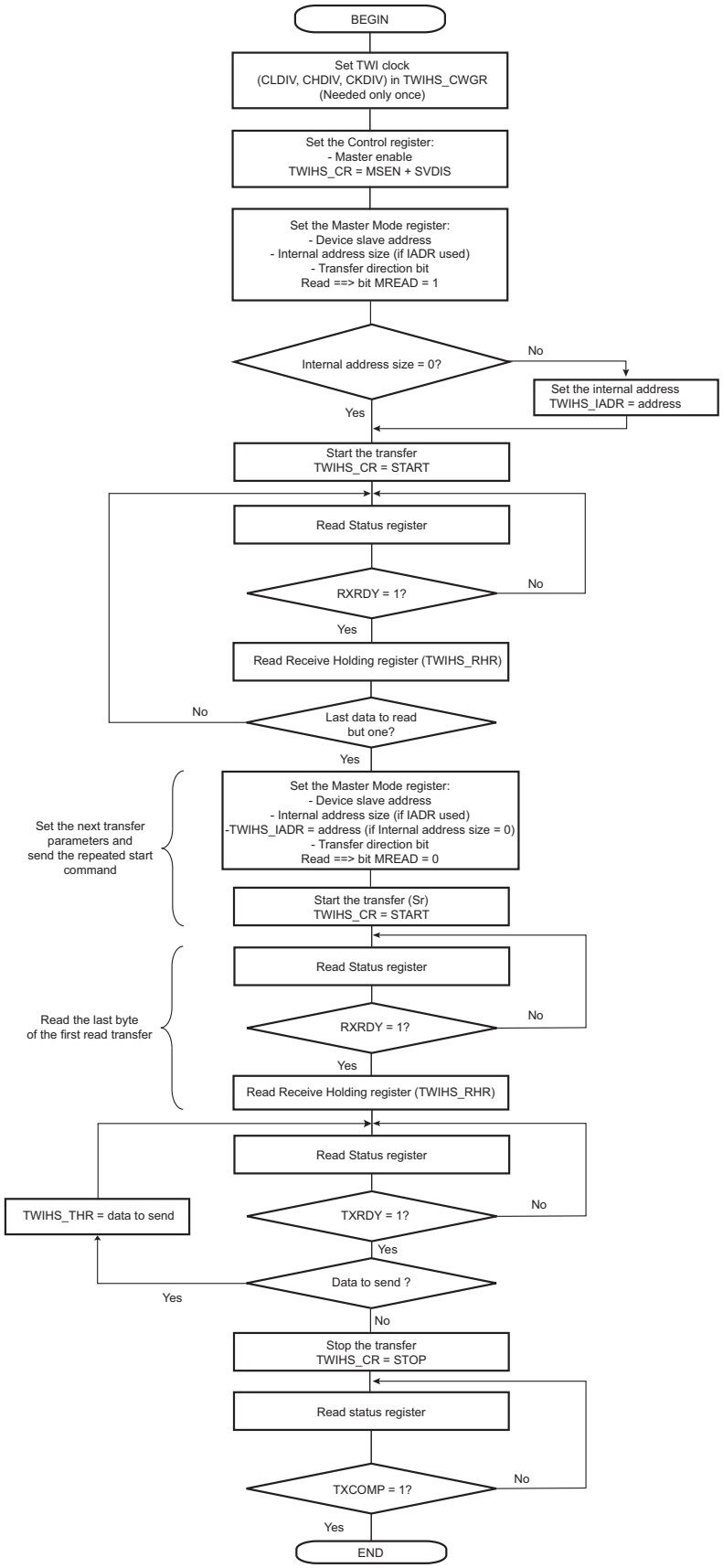
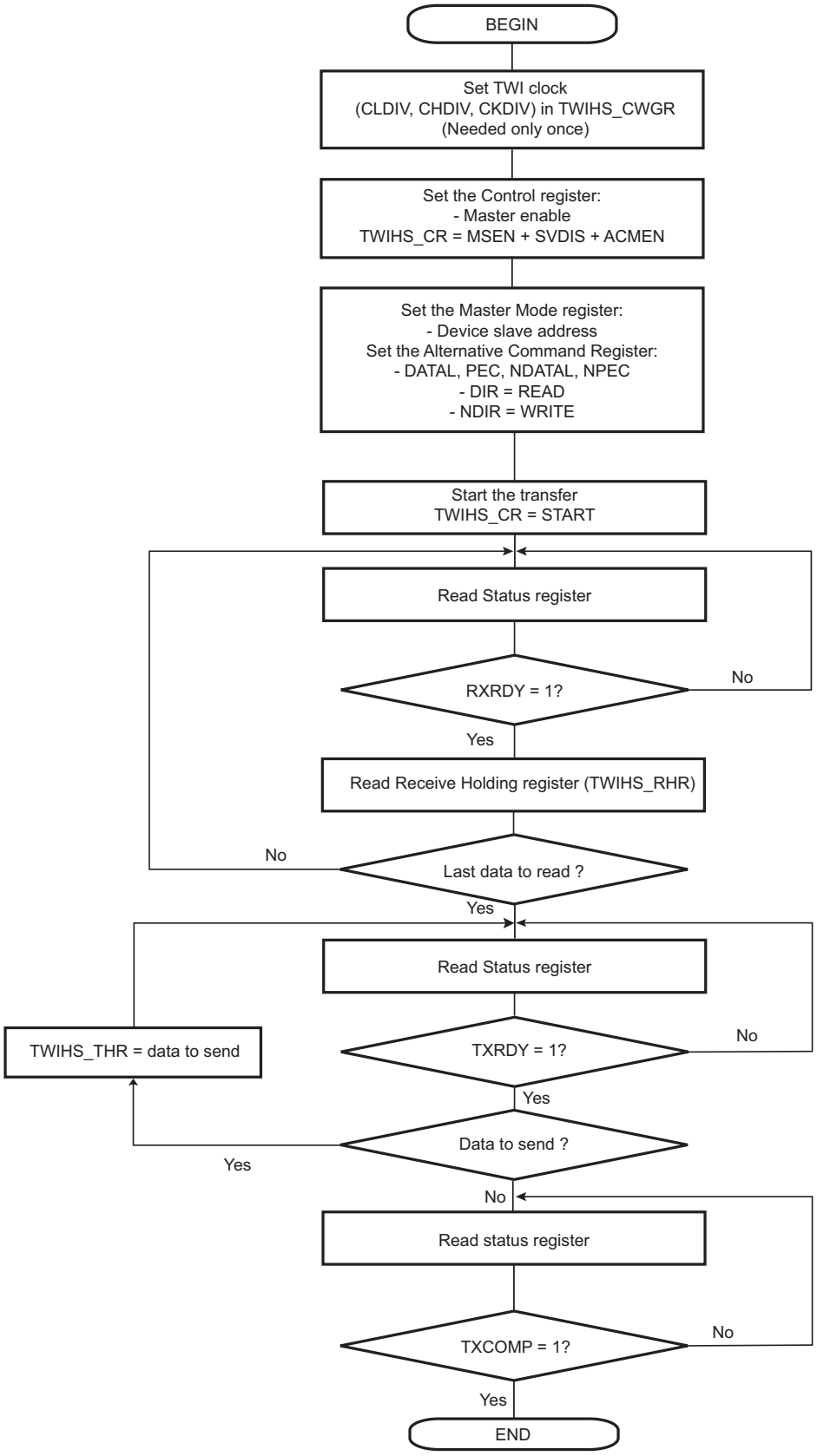


Figure 27-30. TWI Read Operation with Multiple Data Bytes + Write with Alternative Command Mode with PEC



27.7.4 Multi-Master Mode

27.7.4.1 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 arbitration is lost by a master, it stops sending data and listens to the bus in order to detect a STOP. When the STOP is detected, the master that has lost arbitration may put its data on the bus by respecting arbitration.

Arbitration is illustrated in Figure 27-32.

27.7.4.2 Different Multi-Master Modes

Two multi-master modes may be distinguished:

1. The TWI is considered as a master only and will never be addressed.
2. The TWI may be either a master or a slave and may be addressed.

Note: Arbitration in supported in both multi-master modes.

TWI as Master Only

In this mode, the TWI is considered as a master only (MSEN is always at one) and must be driven like a master with the ARBLST (ARBitration Lost) flag in addition.

If arbitration is lost (ARBLST = 1), the user must reinitiate the data transfer.

If the user starts a transfer (ex.: DADR + START + W + Write in THR) and if the bus is busy, the TWI automatically waits for a STOP condition on the bus to initiate the transfer (see Figure 27-31).

Note: The state of the bus (busy or free) is not indicated 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 the 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 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 as free, the 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 wants 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: In case 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 27-31. User Sends Data While the Bus is Busy

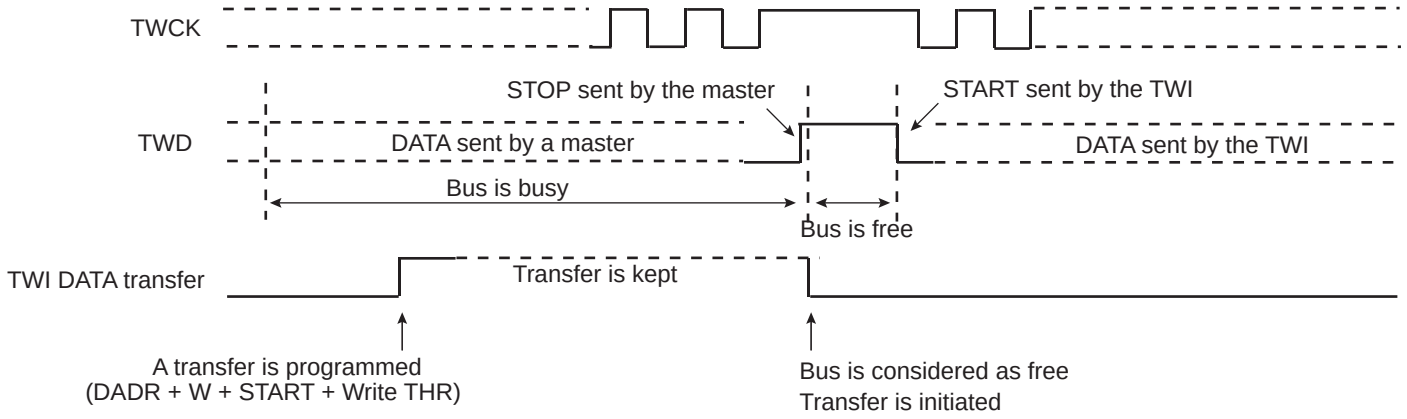
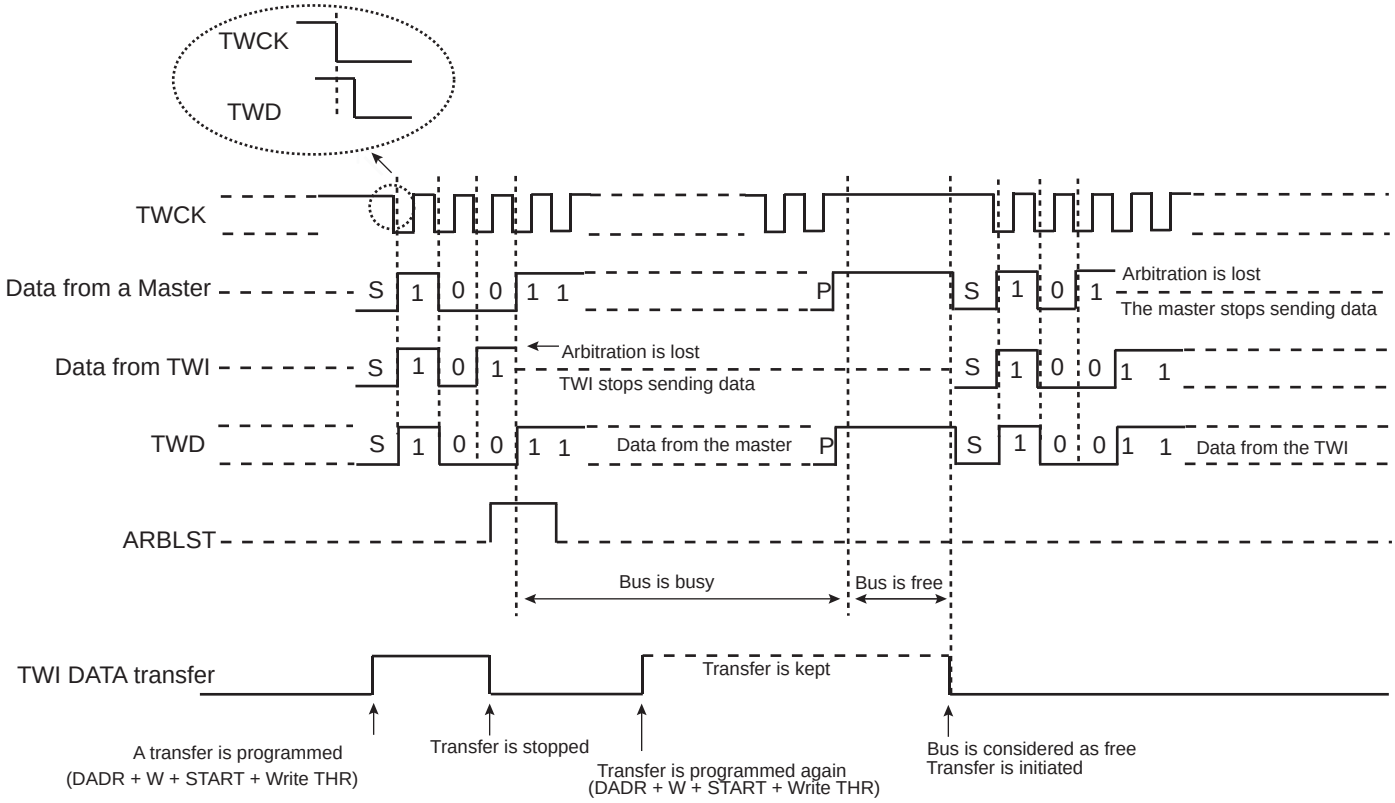
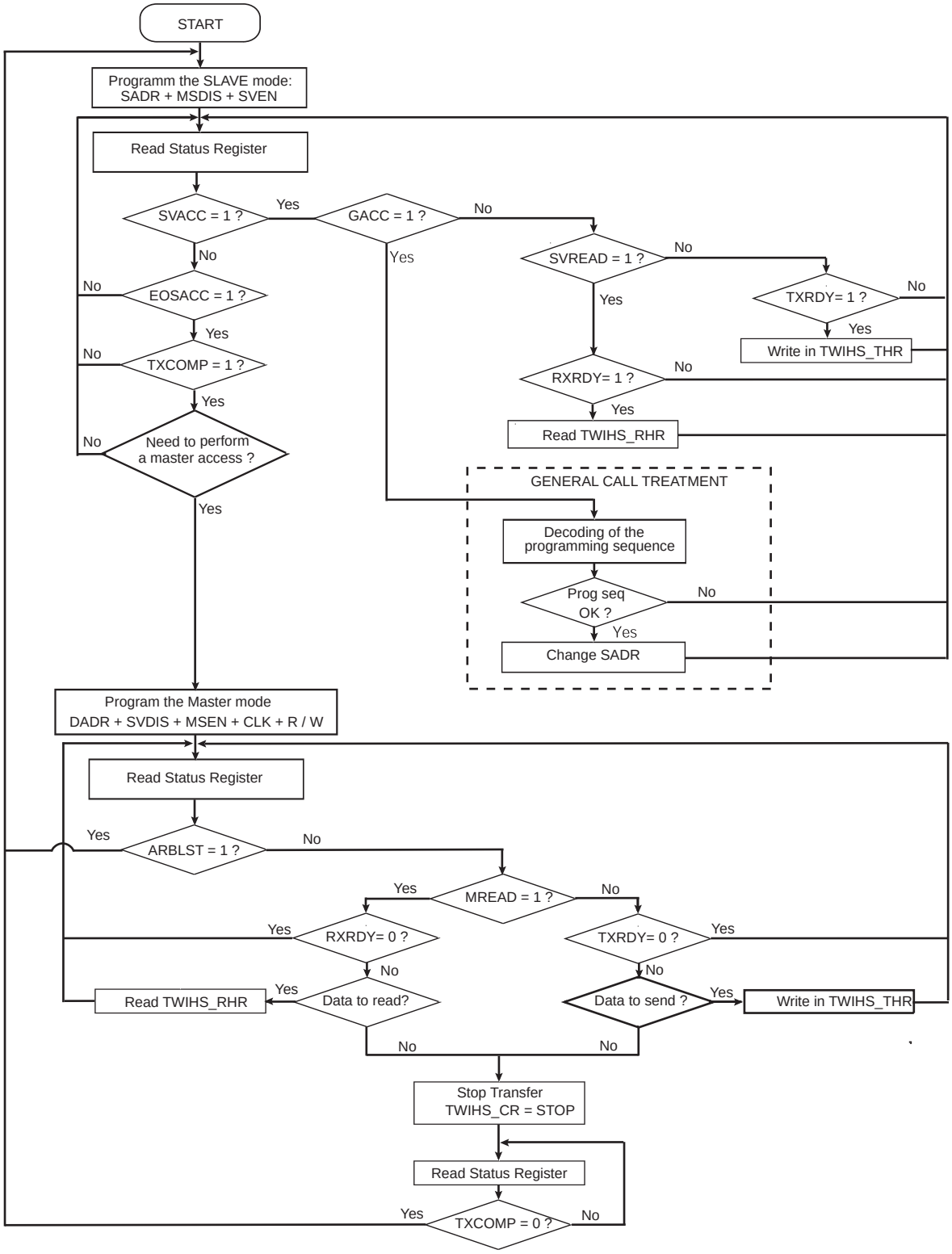


Figure 27-32. Arbitration Cases



The flowchart shown in Figure 27-33 gives an example of read and write operations in multi-master mode.

Figure 27-33. Multi-Master Flowchart



27.7.5 Slave Modes

27.7.5.1 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).

27.7.5.2 Application Block Diagram

Figure 27-34. High-Speed Mode Slave Mode Typical Application Block Diagram

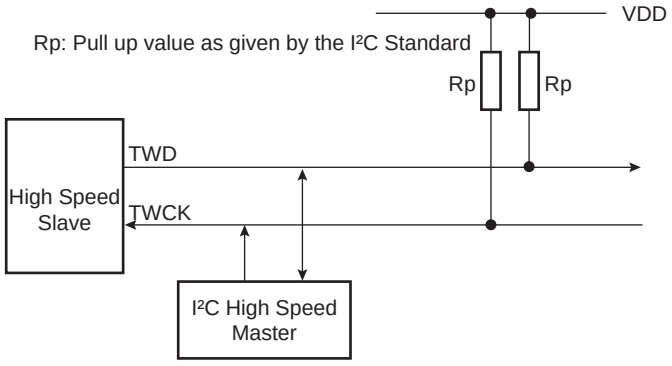
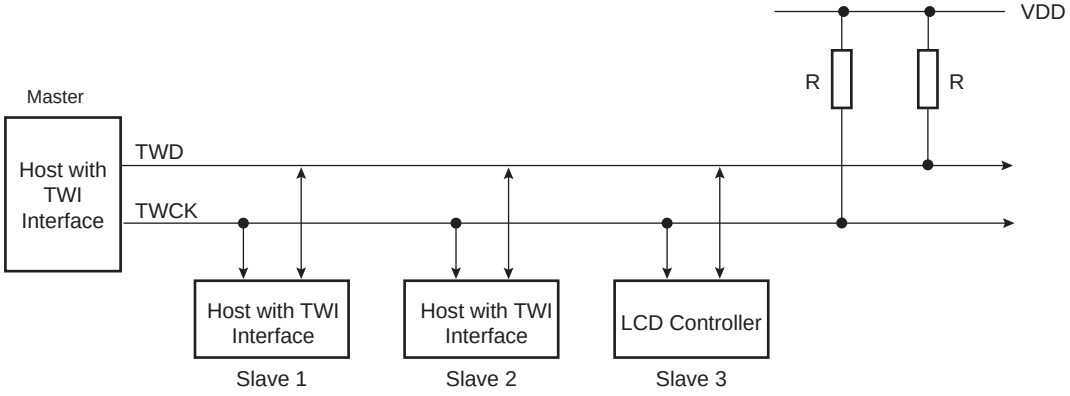


Figure 27-35. Fast Mode Slave Mode Typical Application Block Diagram



27.7.5.3 Programming Slave Mode

The following fields must be programmed before entering slave mode:

1. SADR (TWIHS_SMR): The slave device address is used in order to be accessed by master devices in read or write mode.
2. (Optional) MASK field (TWIHS_SMR) can be set to mask some SADR address bits and thus allow multiple address matching.
3. MSDIS (TWIHS_CR): Disables the master mode.
4. SVEN (TWIHS_CR): Enables the slave mode.

As the device receives the clock, values written in TWIHS_CWGR are not taken into account.

27.7.5.4 Receiving Data

After a START or repeated START condition is detected, and if the address sent by the master matches the slave address programmed in the SADR (Slave ADdress) field, the SVACC (Slave ACCess) flag is set and SVREAD (Slave READ) indicates the direction of the transfer.

SVACC remains high until a STOP condition or a repeated START is detected. When such a condition is detected, EOSACC (End Of Slave ACCess) flag is set.

Read Sequence

In the case of a read sequence (SVREAD is high), the TWI transfers data written in the TWIHS_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 TWIHS_THR, 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.

See Figure 27-36.

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 TWIHS_RHR (TWI Receive Holding Register). RXRDY is reset when reading the TWIHS_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 27-37.

Clock Stretching Sequence

If TWIHS_THR or TWIHS_RHR is not written/read in time, the TWI performs a clock stretching.

Clock stretching information is given by the SCLWS (Clock Wait state) bit.

See Figure 27-39 and Figure 27-40.

Note: Clock stretching can be disabled by configuring the SCLWSDIS bit in the TWIHS_SMR. In that case, UNRE and OVRE flags will indicate underrun (when TWIHS_THR is not filled on time) or overrun (when TWIHS_RHR is not read on time).

General Call

In the case where a GENERAL CALL is performed, GACC (General Call ACCess) flag is set.

After GACC is set, it is up to the user to interpret the meaning of the GENERAL CALL and to decode the new address programming sequence.

See Figure 27-38.

27.7.5.5 Data Transfer

Read Operation

The read mode is defined as a data requirement from the master.

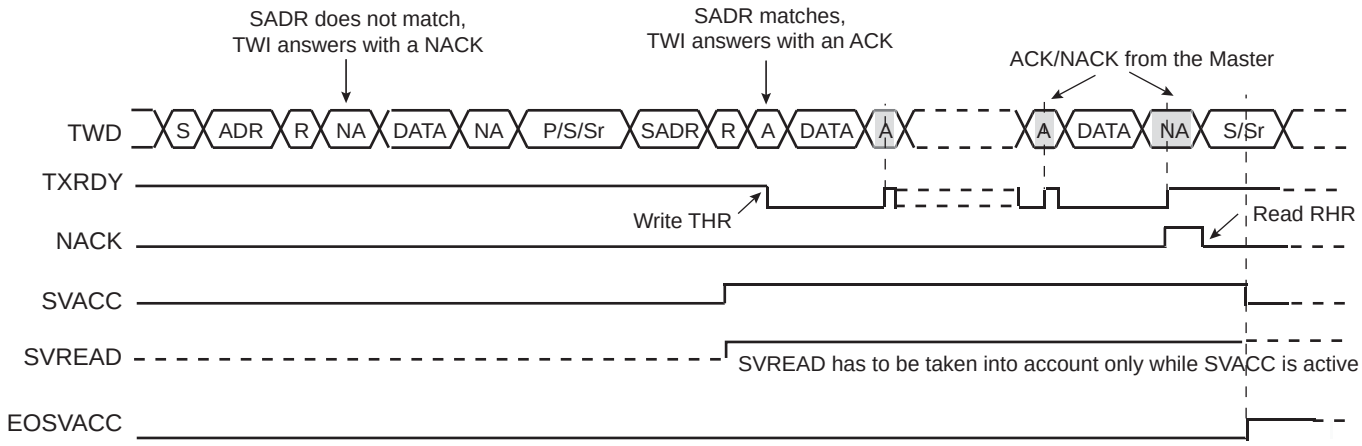
After a START or a REPEATED START condition is detected, the decoding of the address starts. If the slave address (SADR) is decoded, SVACC is set and SVREAD indicates the direction of the transfer.

Until a STOP or REPEATED START condition is detected, TWI continues sending data loaded in the TWIHS_THR.

If a STOP condition or a REPEATED START + an address different from SADR is detected, SVACC is reset.

Figure 27-36 describes the read operation.

Figure 27-36. 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 TWIHS_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.

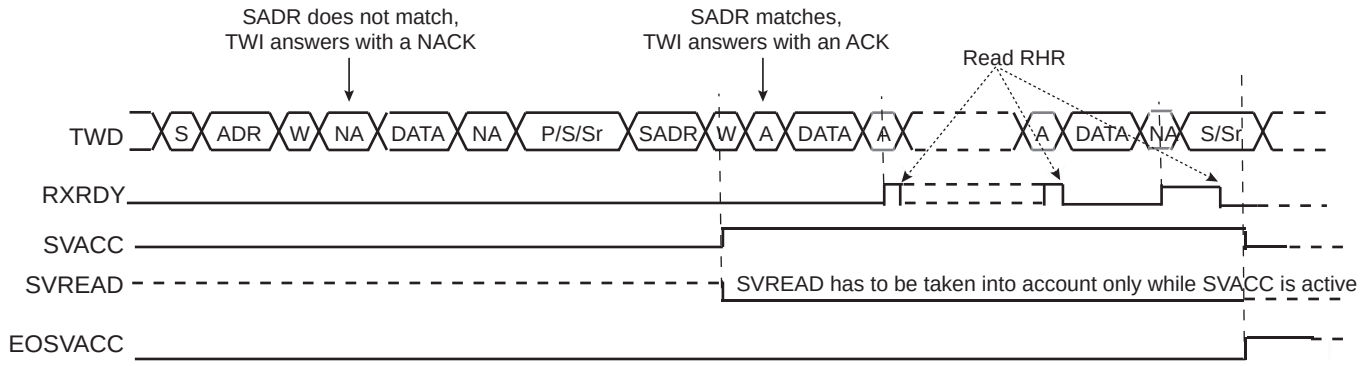
After a START or a REPEATED START, the decoding of the address starts. If the slave address is decoded, SVACC is set and SVREAD indicates the direction of the transfer (SVREAD is low in this case).

Until a STOP or REPEATED START condition is detected, TWI stores the received data in the TWIHS_RHR.

If a STOP condition or a REPEATED START + an address different from SADR is detected, SVACC is reset.

Figure 27-37 describes the write operation.

Figure 27-37. 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 TWIHS_RHR and reset when this data is read.

General Call

The general call is performed in order to change the address of the slave.

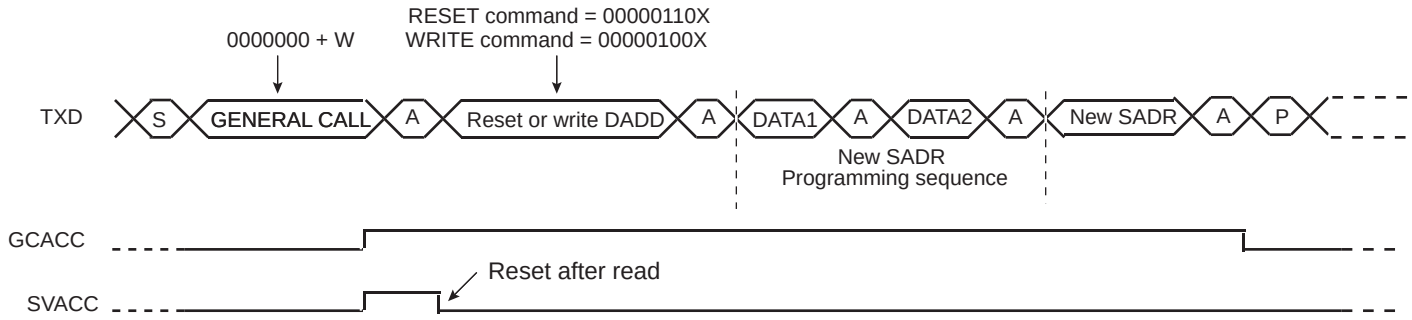
If a GENERAL CALL is detected, GACC is set.

After the detection of general call, it is up to the user to decode the commands which follow.

In case of a WRITE command, the user has to decode the programming sequence and program a new SADR if the programming sequence matches.

Figure 27-38 describes the general call access.

Figure 27-38. Master Performs a General Call



Note: This method allows the user to create an own programming sequence by choosing the programming bytes and the number of them. The programming sequence has to be provided to the master.

Clock Stretching

In both read and write modes, it may happen that TWIHS_THR/TWIHS_RHR buffer is not filled /emptied before the emission/reception of a new character. In this case, to avoid sending/receiving undesired data, a clock stretching mechanism is implemented.

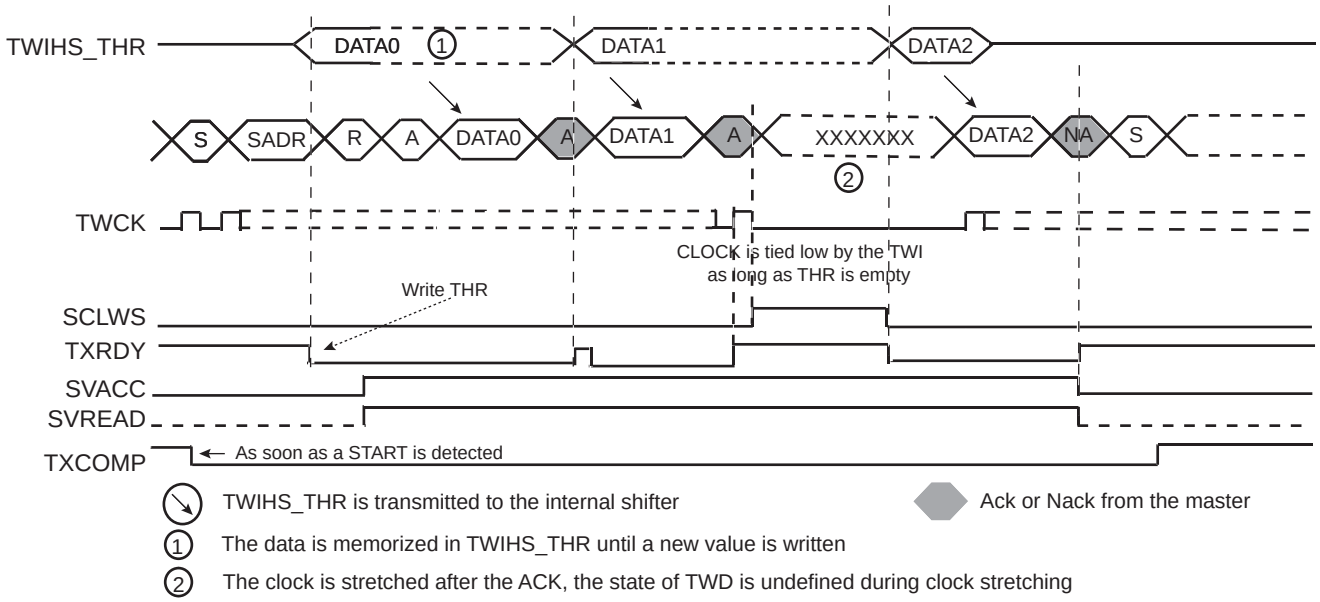
Note: Clock stretching can be disabled by setting the SCLWSDIS bit in the TWIHS_SMR. In that case UNRE and OVRE flags will indicate underrun (when TWIHS_THR is not filled on time) or overrun (when TWIHS_RHR is not read on time).

— Clock Stretching in Read Mode

The clock is tied low 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 27-39 describes the clock stretching in read mode.

Figure 27-39. Clock Stretching in Read Mode



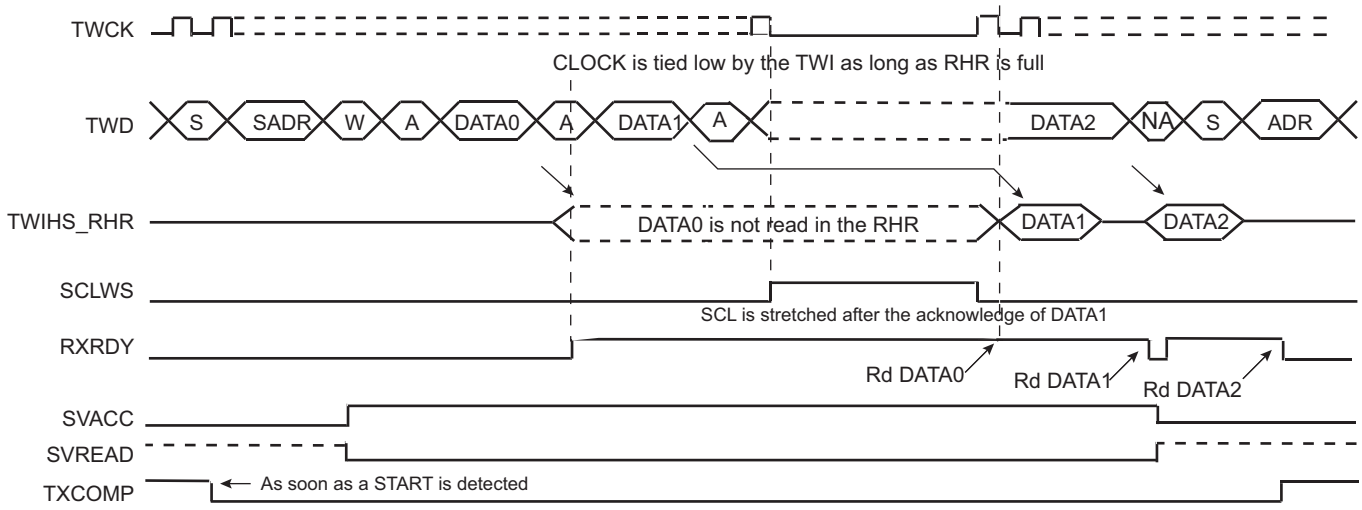
Notes:
1. TXRDY is reset when data has been written in the TWIHS_THR to the internal shifter and set when this data has been acknowledged or non acknowledged.
2. At the end of the read sequence, TXCOMP is set after a STOP or after a REPEATED_START + an address different from SADR.
3. SCLWS is automatically set when the clock stretching mechanism is started.

— Clock Stretching in Write Mode

The clock is tied low if the internal shifter and the TWIHS_RHR is full. If a STOP or REPEATED_START condition was not detected, it is tied low until TWIHS_RHR is read.

Figure 27-40 describes the clock stretching in write mode.

Figure 27-40. Clock Stretching 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 stretching mechanism is started and automatically reset when the mechanism is finished.

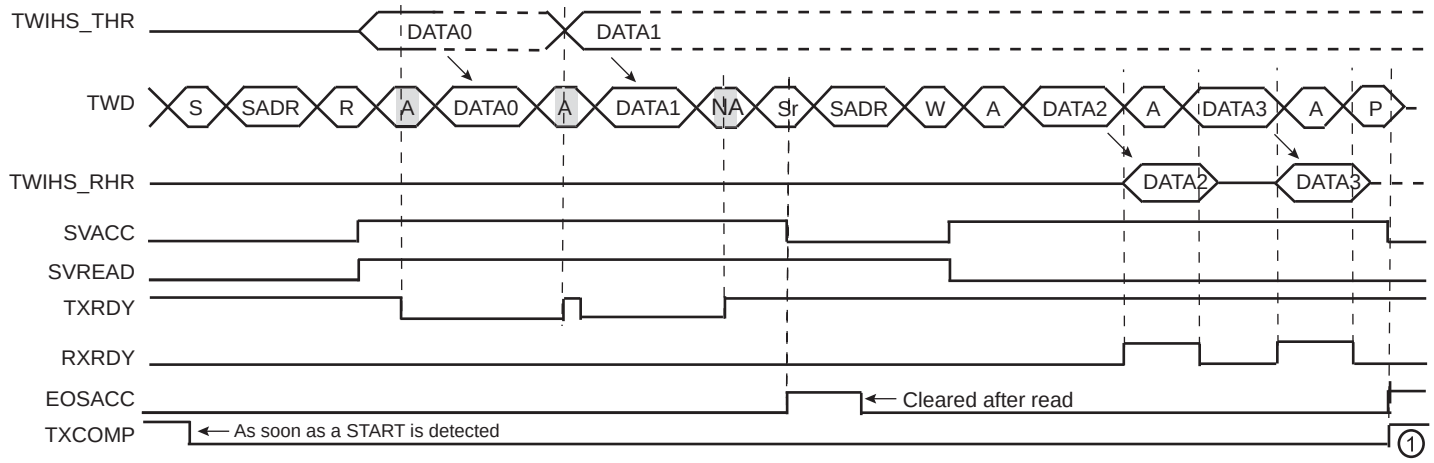
Reversal after a Repeated Start

— Reversal of Read to Write

The master initiates the communication by a read command and finishes it by a write command.

Figure 27-41 describes the repeated start and the reversal from read mode to write mode.

Figure 27-41. Repeated Start and Reversal from Read Mode to Write Mode

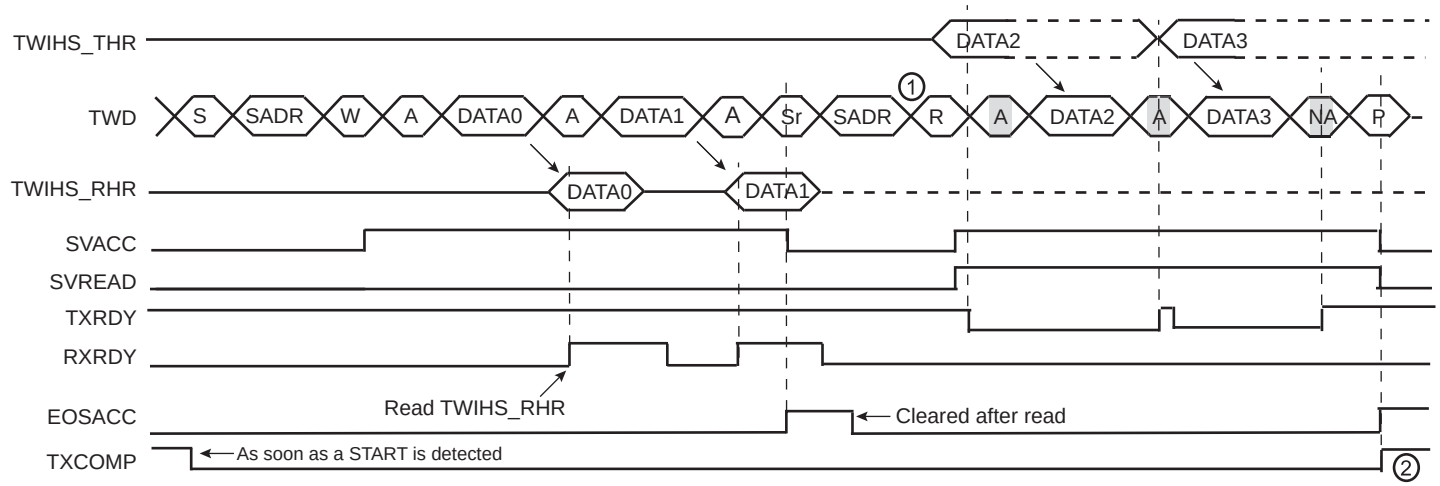


Note: 1. TXCOMP is only set at the end of the transmission because after the repeated start, SADR is detected again.

— Reversal of Write to Read

The master initiates the communication by a write command and finishes it by a read command. Figure 27-42 describes the repeated start and the reversal from write mode to read mode.

Figure 27-42. Repeated Start and Reversal from Write Mode to Read Mode



Notes: 1. In this case, if TWIHS_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.

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 to transmit data 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 either by using the polling method or ENDTX interrupt.
4. Disable the PDC by setting the PDC TXTDIS bit.
5. (Optional) Wait for the TXCOMP flag in TWIHS_SR before disabling the peripheral clock if required.

— Data Receive with the PDC in Slave Mode

The following procedure shows an example to transmit data with PDC where the number of characters to receive is known.

1. Initialize the receive PDC (memory pointers, transfer size).
2. Set the PDC RXTEN bit.
3. Wait for the PDC ENDRX flag either by using polling method or ENDRX interrupt.
4. Disable the PDC by setting the PDC RXTDIS bit.
5. (Optional) Wait for the TXCOMP flag in TWIHS_SR before disabling the peripheral clock if required.

SMBus Mode

SMBus mode is enabled when SMEN bit is written to one in TWIHS_CR. SMBus mode operation is similar to I²C operation with the following exceptions:

1. Only 7-bit addressing can be used.
2. The SMBus standard describes a set of timeout values to ensure progress and throughput on the bus. These time-out values must be programmed into the TWIHS_SMBTR.
3. Transmissions can optionally include a CRC byte, called Packet Error Check (PEC).
4. A dedicated bus line, SMBALERT, allows a slave to get a master attention.
5. A set of addresses have been reserved for protocol handling, such as alert response address (ARA) and host header (HH) address. Address matching on these addresses can be enabled by configuring TWIHS_CR appropriately.

— Packet Error Checking

Each SMBus transfer can optionally end with a CRC byte, called the PEC byte. Writing the PECEN bit in TWIHS_CR to one will send/check the PEC field in the current transfer. The PEC generator is always updated on every bit transmitted or received, so that PEC handling on following linked transfers will be correct.

In slave receiver mode, the master calculates a PEC value and transmits it to the slave after all data bytes have been transmitted. Upon reception of this PEC byte, the slave will compare it to the PEC value it has computed itself. If the values match, the data was received correctly, and the slave will return an ACK to the master. If the PEC values differ, data was corrupted, and the slave will return a NACK value. The PECERR bit in TWIHS_SR is set automatically if a PEC error occurred.

In slave transmitter mode, the slave calculates a PEC value and transmits it to the master after all data bytes have been transmitted. Upon reception of this PEC byte, the master will compare it to the PEC value it has computed itself. If the values match, the data was received correctly. If the PEC values differ, data was corrupted, and the master must take appropriate action.

See Section 27.7.5.7 ”Slave Read Write Flowcharts” for detailed flowcharts.

— Timeouts

The TWI SMBus Timing Register (TWIHS_SMBTR) configures the SMBus timeout values. If a timeout occurs, the slave will leave the bus. The TOUT bit is also set in TWIHS_SR.

27.7.5.6 High-Speed Slave Mode

High-speed mode is enabled when the HSEN bit is written to one in TWIHS_CR. Furthermore, the analog pad filter must be enabled, the PADFEN bit must be written to one in TWIHS_FILTR and the FILT bit must be cleared. TWI High-speed mode operation is similar to TWI operation with the following exceptions:

1. A master code is received first at normal speed before entering high-speed mode period.
2. When TWI high-speed mode is active, clock stretching is only allowed after acknowledge (ACK), not-acknowledge (NACK), START (S) or repeated START (Sr) (as consequence OVF may happen).

TWI high-speed mode allows transfers of up to 3.4 Mbit/s.

The TWI slave in high-speed mode requires that slave clock stretching is disabled (SCLWSDIS bit at ‘1’). The peripheral clock must run at a minimum of 11 MHz.

Note: When slave clock stretching is disabled, the TWIHS_RHR must always be read before receiving the next data (MASTER write frame). It is strongly recommended to use either the polling method on the RXRDY flag in TWIHS_SR, or the PDC. If the receive is managed by an interrupt, the TWI interrupt priority must be set to the right level and its latency minimized to avoid receive overrun.

Note: When slave clock stretching is disabled, the TWIHS_THR must be filled with the first data to send before the beginning of the frame (MASTER read frame). It is strongly recommended to use either the polling method on the TXRDY flag in TWIHS_SR, or the PDC. If the transmit is managed by an interrupt, the TWI interrupt priority must be set to the right level and its latency minimized to avoid transmit underrun.

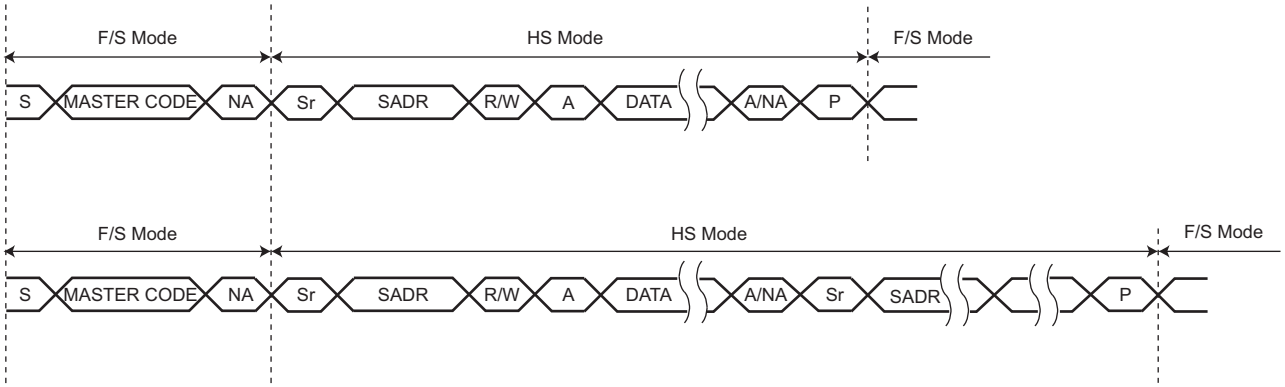
Read/Write Operation

A TWI high-speed frame always begins with the following sequence:

1. START condition (S)
2. Master Code (0000 1XXX)
3. Not-acknowledge (NACK)

When the TWI is programmed in slave mode and TWI high-speed mode is activated, master code matching is activated and internal timings are set to match the TWI high-speed mode requirements.

Figure 27-43. High-Speed Mode Read/Write



Usage

TWI high-speed mode usage is the same as the standard TWI (See Section 27.7.3.12).

27.7.5.7 Slave Read Write Flowcharts

The flowchart shown in Figure 27-44 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 (TWIHS_IER) be configured first.

Figure 27-44. Read Write Flowchart in Slave Mode

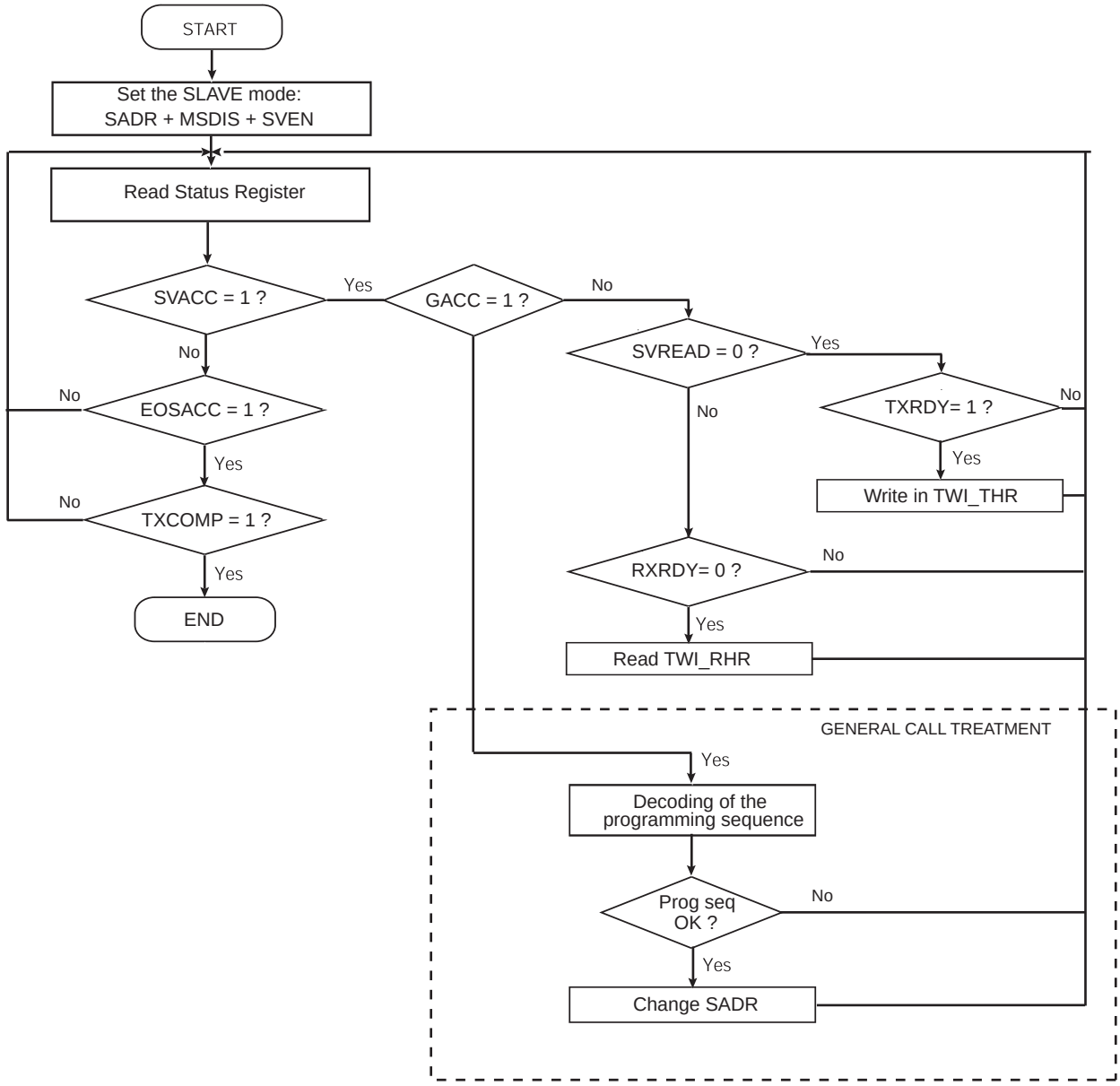


Figure 27-45. Read Write Flowchart in Slave Mode with SMBus PEC

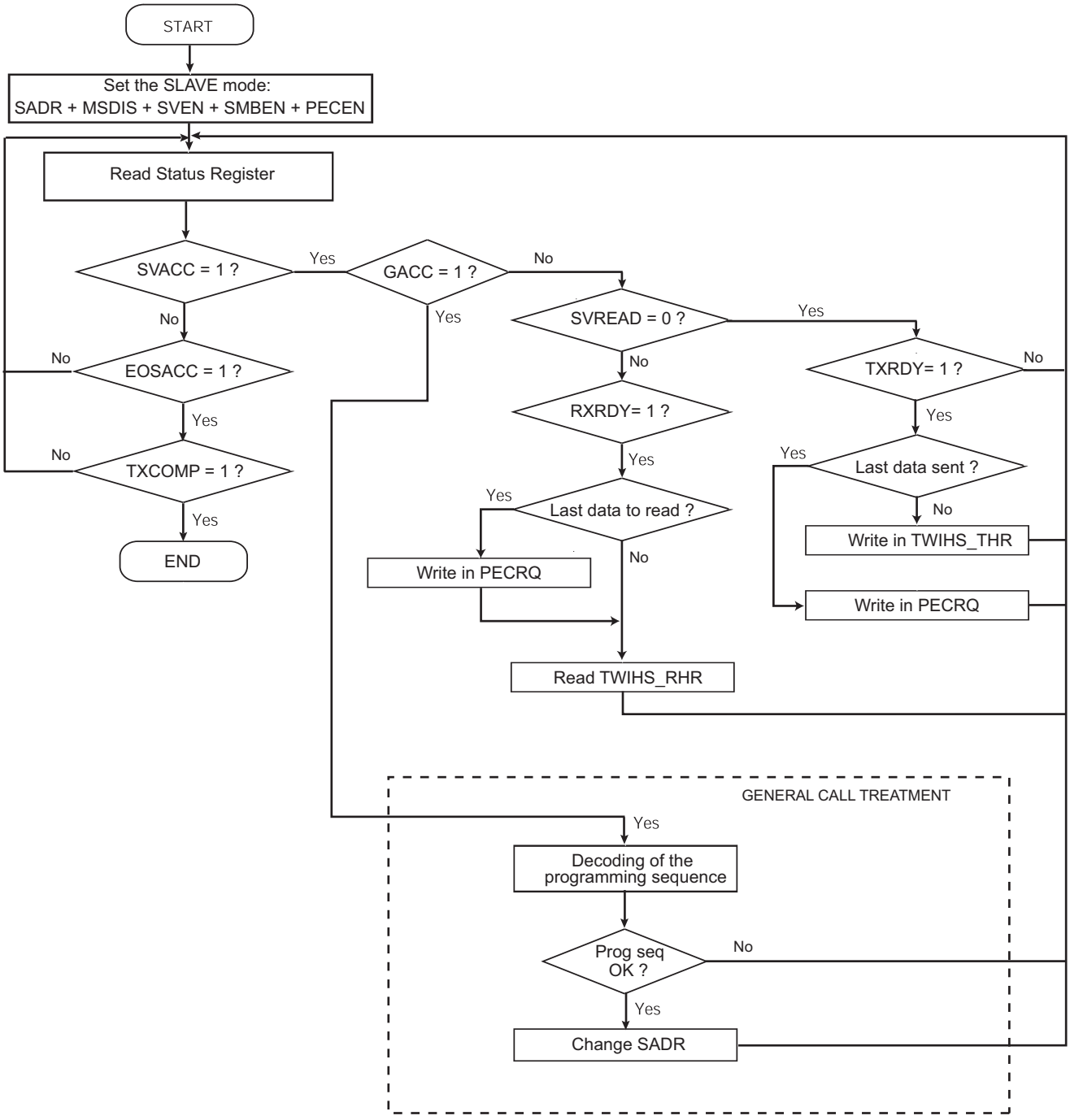
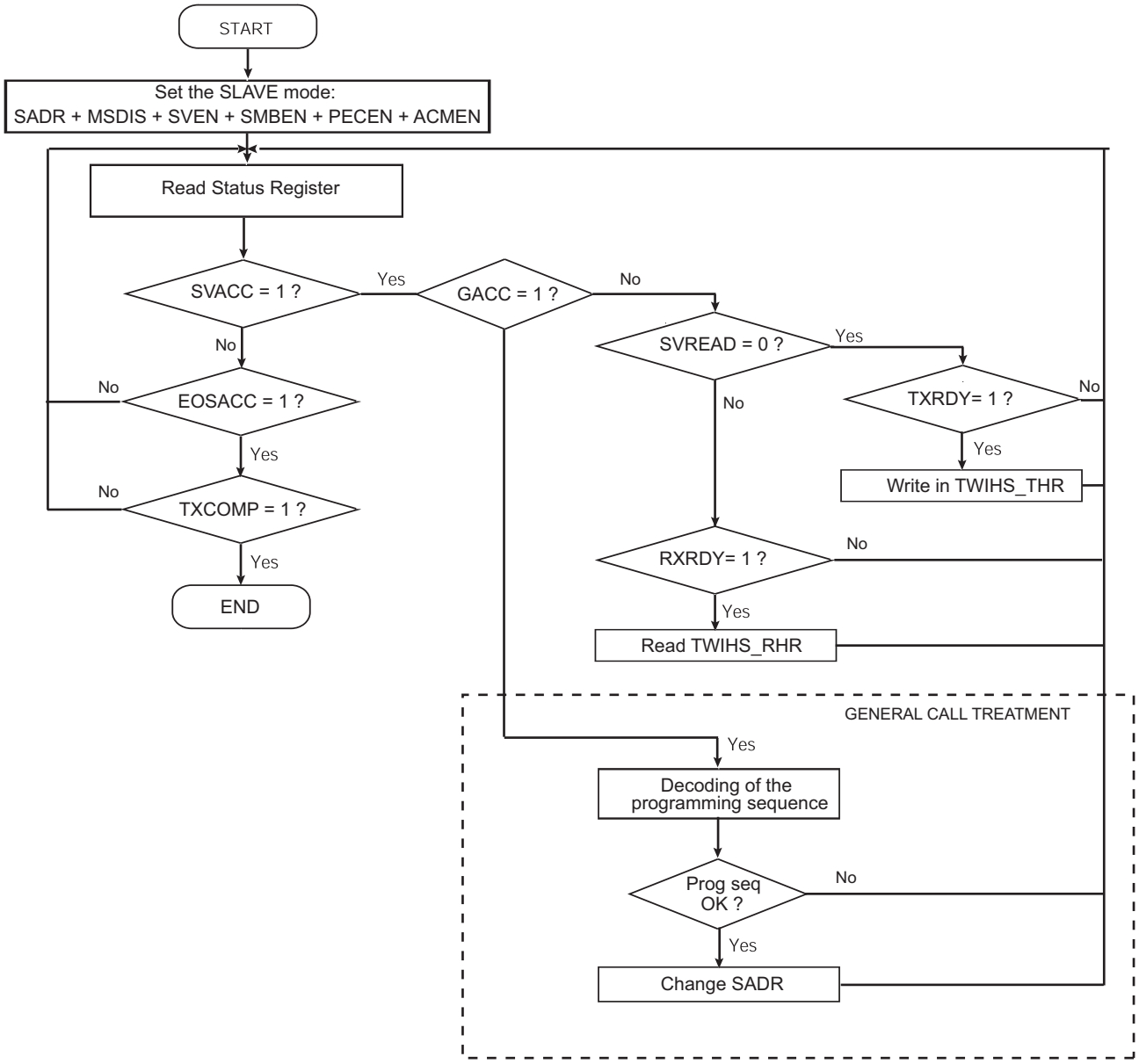


Figure 27-46. Read Write Flowchart in Slave Mode with SMBus PEC and Alternative Command Mode



27.7.6 Register Write Protection

To prevent any single software error from corrupting TWIHS behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the TWI Write Protection Mode Register (TWIHS_WPMR).

If a write access to a write-protected register is detected, the WPVS bit in the TWI Write Protection Status Register (TWIHS_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 TWIHS_WPSR.

The following register(s) can be write-protected:

- TWI Slave Mode Register
- TWI Clock Waveform Generator Register

27.8 Two-wire Interface High Speed (TWIHS) User Interface

Table 27-6. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	TWIHS_CR	Write-only	–
0x04	Master Mode Register	TWIHS_MMR	Read/Write	0x00000000
0x08	Slave Mode Register	TWIHS_SMR	Read/Write	0x00000000
0x0C	Internal Address Register	TWIHS_IADR	Read/Write	0x00000000
0x10	Clock Waveform Generator Register	TWIHS_CWGR	Read/Write	0x00000000
0x14–0x1C	Reserved	–	–	–
0x20	Status Register	TWIHS_SR	Read-only	0x0300F009
0x24	Interrupt Enable Register	TWIHS_IER	Write-only	–
0x28	Interrupt Disable Register	TWIHS_IDR	Write-only	–
0x2C	Interrupt Mask Register	TWIHS_IMR	Read-only	0x00000000
0x30	Receive Holding Register	TWIHS_RHR	Read-only	0x00000000
0x34	Transmit Holding Register	TWIHS_THR	Write-only	0x00000000
0x38	SMBus Timing Register	TWIHS_SMBTR	Read/Write	0x00000000
0x3C	Reserved	–	–	–
0x40	Alternative Command Register	TWIHS_ACR	Read/Write	–
0x44	Filter Register	TWIHS_FILTR	Read/Write	0x00000000
0x48	Reserved	–	–	–
0x4C	Reserved	–	–	–
0x50–0xCC	Reserved	–	–	–
0x0D0	Reserved	–	–	–
0xD4–0xE0	Reserved	–	–	–
0xE4	Protection Mode Register	TWIHS_WPMR	Read/Write	0x00000000
0xE8	Protection Status Register	TWIHS_WPSR	Read-only	0x00000000
0xEC–0xFC[1]	Reserved	–	–	–
0x100–0x128	Reserved for PDC registers	–	–	–

Note: 1. All unlisted offset values are considered as “reserved”.

27.8.1 TWI Control Register

Name: TWIHS_CR

Address: 0x40018000

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
CLEAR	PECRQ	PECDIS	PECEN	SMBDIS	SMBEN	HSDIS	HSEN
7	6	5	4	3	2	1	0
SWRST	QUICK	SVDIS	SVEN	MSDIS	MSEN	STOP	START

- **START: Send a START Condition**

0: No effect.

1: A frame beginning with a START bit is transmitted according to the features defined in the mode register.

This action is necessary when the TWI peripheral wants to read data from a slave. When configured in master mode with a write operation, a frame is sent as soon as the user writes a character in the Transmit Holding Register (TWIHS_THR).

- **STOP: Send a STOP Condition**

0: No effect.

1: STOP Condition is sent just after completing the current byte transmission in master read mode.

  – In single data byte master read, the START and STOP must both be set.
  – In multiple data bytes master read, the STOP must be set after the last data received but one.
  – In master read mode, if a NACK bit is received, the STOP is automatically performed.
  – In master data write operation, a STOP condition will be sent after the transmission of the current data is finished.

- **MSEN: TWI Master Mode Enabled**

0: No effect.

1: 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.

- **HSEN: TWI High-Speed Mode Enabled**

0: No effect.

1: High-speed mode enabled.

- **HSDIS: TWI High-Speed Mode Disabled**

0: No effect.

1: High-speed mode disabled.

- **SMBEN: SMBus Mode Enabled**

0: No effect.

1: If SMBDIS = 0, SMBus mode enabled.

- **SMBDIS: SMBus Mode Disabled**

0: No effect.

1: SMBus mode disabled.

- **PECEN: Packet Error Checking Enable**

0: No effect.

1: SMBus PEC (CRC) generation and check enabled.

- **PECDIS: Packet Error Checking Disable**

0: No effect.

1: SMBus PEC (CRC) generation and check disabled.

- **PECRQ: PEC Request**

0: No effect.

1: A PEC check or transmission is requested.

- **CLEAR: Bus CLEAR Command**

0: No effect.

1: If master mode is enabled, send a bus clear command.

27.8.2 TWI Master Mode Register

Name: TWIHS_MMR

Address: 0x40018004

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	DADR						

15	14	13	12	11	10	9	8
–	–	–	MREAD	–	–	IADRSZ	

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

- **IADRSZ: Internal Device Address Size**

Value	Name	Description
0	NONE	No internal device address
1	1_BYTE	One-byte internal device address
2	2_BYTE	Two-byte internal device address
3	3_BYTE	Three-byte internal device address

- **MREAD: Master Read Direction**

0: Master write direction.

1: Master read direction.

- **DADR: Device Address**

The device address is used to access slave devices in read or write mode. Those bits are only used in master mode.

27.8.3 TWI Slave Mode Register

Name: TWIHS_SMR

Address: 0x40018008

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	SADR						

15	14	13	12	11	10	9	8
–	MASK						

7	6	5	4	3	2	1	0
–	SCLWSDIS	–	–	SMHH	SMDA	–	NACKEN

This register can only be written if the WPEN bit is cleared in the TWI Write Protection Mode Register.

- **NACKEN: Slave Receiver Data Phase NACK enable**

0: Normal value to be returned in the ACK cycle of the data phase in slave receiver mode.

1: NACK value to be returned in the ACK cycle of the data phase in slave receiver mode.

- **SMDA: SMBus Default Address**

0: Acknowledge of the SMBus Default Address disabled.

1: Acknowledge of the SMBus Default Address enabled.

- **SMHH: SMBus Host Header**

0: Acknowledge of the SMBus Host Header disabled.

1: Acknowledge of the SMBus Host Header enabled.

- **SCLWSDIS: Clock Wait State Disable**

0: No effect.

1: Clock stretching disabled in slave mode, OVRE and UNRE will indicate overrun and underrun.

- **MASK: Slave Address Mask**

A mask can be applied on the slave device address in slave mode in order to allow multiple address answer. For each bit of the MASK field set to one the corresponding SADR bit will be masked.

If MASK field is set to 0 no mask is applied to SADR field.

- **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.

27.8.4 TWI Internal Address Register

Name: TWIHS_IADR

Address: 0x4001800C

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
IADR							

15	14	13	12	11	10	9	8
IADR							

7	6	5	4	3	2	1	0
IADR							

- **IADR: Internal Address**

0, 1, 2 or 3 bytes depending on IADRSZ.

27.8.5 TWI Clock Waveform Generator Register

Name: TWIHS_CWGR

Address: 0x40018010

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	HOLD				

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.

TWIHS_CWGR is only used in master mode.

- **CLDIV: Clock Low Divider**

The SCL low period is defined as follows:

$$t_{low} = ((\text{CLDIV} \times 2^{\text{CKDIV}}) + 3) \times t_{MCK}$$

- **CHDIV: Clock High Divider**

The SCL high period is defined as follows:

$$t_{high} = ((\text{CHDIV} \times 2^{\text{CKDIV}}) + 3) \times t_{MCK}$$

- **CKDIV: Clock Divider**

The CKDIV is used to increase both SCL high and low periods.

- **HOLD: TWD Hold Time Versus TWCK Falling**

If high-speed mode is selected TWD is internally modified on the TWCK falling edge to meet the I2C specified maximum hold time, else if high-speed mode is not configured TWD is kept unchanged after TWCK falling edge for a period of (HOLD+3)*Tmck.

27.8.6 TWI Status Register

Name: TWIHS_SR

Address: 0x40018020

Access: Read-only

Reset: 0x0300F009

31	30	29	28	27	26	25	24
–	–	–	–	–	–	SDA	SCL
23	22	21	20	19	18	17	16
–	–	SMBHHM	SMBDAM	PECERR	TOUT	–	MCACK
15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	ENDTX	ENDRX	EOSACC	SCLWS	ARBLST	NACK
7	6	5	4	3	2	1	0
UNRE	OVRE	GACC	SVACC	SVREAD	TXRDY	RXRDY	TXCOMP

- **TXCOMP: Transmission Completed (automatically set / reset)**

TXCOMP used in master mode:

0: During the length of the current frame.

1: When both holding register and internal shifter are empty and STOP condition has been sent.

TXCOMP behavior in master mode can be seen in Figure 27-9 and in Figure 27-11.

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 27-39, Figure 27-40, Figure 27-41 and Figure 27-42.

- **RXRDY: Receive Holding Register Ready (automatically set / reset)**

0: No character has been received since the last TWIHS_RHR read operation.

1: A byte has been received in the TWIHS_RHR since the last read.

RXRDY behavior in master mode can be seen in Figure 27-11.

RXRDY behavior in slave mode can be seen in Figure 27-37, Figure 27-40, Figure 27-41 and Figure 27-42.

- **TXRDY: Transmit Holding Register Ready (automatically set / reset)**

TXRDY used in master mode:

0: The transmit holding register has not been transferred into the internal shifter. Set to 0 when writing into TWIHS_THR.

1: As soon as a data byte is transferred from TWIHS_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 (enables TWI).

TXRDY behavior in master mode can be seen in Figure 27.7.3.4.

TXRDY used in slave mode:

0: As soon as data is written in the TWIHS_THR, until this data has been transmitted and acknowledged (ACK or NACK).

1: Indicates that the TWIHS_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 user must not fill TWIHS_THR to avoid losing it.

TXRDY behavior in slave mode can be seen in Figure 27-36, Figure 27-39, Figure 27-41 and Figure 27-42.

- **SVREAD: Slave Read (automatically set / reset)**

This bit is only used in slave mode. When SVACC is low (no slave access has been detected) SVREAD is irrelevant.

0: Indicates that a write access is performed by a master.

1: Indicates that a read access is performed by a master.

SVREAD behavior can be seen in Figure 27-36, Figure 27-37, Figure 27-41 and Figure 27-42.

- **SVACC: Slave Access (automatically set / reset)**

This bit is only used in slave mode.

0: TWI is not addressed. SVACC is automatically cleared after a NACK or a STOP condition is detected.

1: Indicates that the address decoding sequence has matched (A master has sent SADR). SVACC remains high until a NACK or a STOP condition is detected.

SVACC behavior can be seen in Figure 27-36, Figure 27-37, Figure 27-41 and Figure 27-42.

- **GACC: General Call Access (clear on read)**

This bit is only used in slave mode.

0: No general call has been detected.

1: A general call has been detected. After the detection of general call, if need be, the user 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 27-38.

- **OVRE: Overrun Error (clear on read)**

This bit is only used if clock stretching is disabled.

0: TWIHS_RHR has not been loaded while RXRDY was set.

1: TWIHS_RHR has been loaded while RXRDY was set. Reset by read in TWIHS_SR when TXCOMP is set.

- **UNRE: Underrun Error (clear on read)**

This bit is only used if clock stretching is disabled.

0: TWIHS_THR has been filled on time.

1: TWIHS_THR has not been filled on time.

- **NACK: Not Acknowledged (clear on read)**

NACK used in master mode:

0: Each data byte has been correctly received by the far-end side TWI slave component.

1: A data or 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 user must not fill TWIHS_THR even if TXRDY is set, because it means that the master will stop the data transfer or re initiate it.

Note that in slave write mode all data are acknowledged by the TWI.

- **ARBLST: Arbitration Lost (clear on read)**

This bit is only used in master mode.

0: Arbitration won.

1: Arbitration lost. Another master of the TWI bus has won the multi-master arbitration. TXCOMP is set at the same time.

- **SCLWS: Clock Wait State (automatically set / reset)**

This bit is only used in slave mode.

0: The clock is not stretched.

1: The clock is stretched. TWIHS_THR / TWIHS_RHR buffer is not filled / emptied before the emission / reception of a new character.

SCLWS behavior can be seen in Figure 27-39 and Figure 27-40.

- **EOSACC: End Of Slave Access (clear on read)**

This bit is only used in slave mode.

0: A slave access is being performing.

1: The Slave Access is finished. End Of Slave Access is automatically set as soon as SVACC is reset.

EOSACC behavior can be seen in Figure 27-41 and Figure 27-42.

- **ENDRX: End of RX Buffer**

0: The Receive Counter Register has not reached 0 since the last write in TWIHS_RCR or TWIHS_RNCR.

1: The Receive Counter Register has reached 0 since the last write in TWIHS_RCR or TWIHS_RNCR.

- **ENDTX: End of TX Buffer**

0: The Transmit Counter Register has not reached 0 since the last write in TWIHS_TCR or TWIHS_TNCR.

1: The Transmit Counter Register has reached 0 since the last write in TWIHS_TCR or TWIHS_TNCR.

- **RXBUFF: RX Buffer Full**

0: TWIHS_RCR or TWIHS_RNCR have a value other than 0.

1: Both TWIHS_RCR and TWIHS_RNCR have a value of 0.

- **TXBUFE: TX Buffer Empty**

0: TWIHS_TCR or TWIHS_TNCR have a value other than 0.

1: Both TWIHS_TCR and TWIHS_TNCR have a value of 0.

- **MCACK: Master Code Acknowledge**

MACK used in slave mode:

0: No Master Code has been received.

1: A master Code has been received.

- **TOUT: Timeout Error**

0: No SMBus timeout occurred.

1: SMBus timeout occurred.

- **PECERR: PEC Error**

0: No SMBus PEC error occurred.

1: A SMBus PEC error occurred.

- **SMBDAM: SMBus Default Address Match**

0: No SMBus Default Address received.

1: A SMBus Default Address was received.

- **SMBHHM: SMBus Host Header Address Match**

0: No SMBus Host Header Address received.

1: A SMBus Host Header Address was received.

- **SCL: SCL line value**

0: SCL line sampled value is ‘0’.

1: SCL line sampled value is ‘1.’

- **SDA: SDA line value**

0: SDA line sampled value is ‘0’.

1: SDA line sampled value is ‘1’.

27.8.7 TWI SMBus Timing Register

Name: TWIHS_SMBTR

Address: 0x40018038

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
THMAX							

23	22	21	20	19	18	17	16
TLOWM							

15	14	13	12	11	10	9	8
TLOWS							

7	6	5	4	3	2	1	0
–	–	–	–	PRESC			

- **PRESC: SMBus Clock Prescaler**

Used to specify how to prescale the TLOWS, TLOWM and THMAX counters in SMBTR. Counters are prescaled according to the following formula:

$$f_{Prescaled} = \frac{f_{MCK}}{2^{(PRESC+1)}}$$

- **TLOWS: Slave Clock Stretch Maximum Cycles**

0: TLOW:SEXT timeout check disabled.

1–255 = Clock cycles in slave maximum clock stretch count. Prescaled by PRESC. Used to time TLOW:SEXT.

- **TLOWM: Master Clock Stretch Maximum Cycles**

0: TLOW:MEXT timeout check disabled.

1–255 = Clock cycles in master maximum clock stretch count. Prescaled by PRESC. Used to time TLOW:MEXT.

- **THMAX: Clock High Maximum Cycles**

Clock cycles in clock high maximum count. Prescaled by PRESC. Used for bus free detection. Used to time THIGH:MAX.

27.8.8 TWI Filter Register

Name: TWIHS_FILTR

Address: 0x40018044

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
–	–	–	–	–	THRES		
7	6	5	4	3	2	1	0
–	–	–	–	–	PADFCFG	PADFEN	FILT

- **FILT: RX Digital Filter**

0: No filtering applied on TWI inputs.

1: TWI input filtering is active. (Only in standard and fast modes)

Note: TWI digital input filtering follows a majority decision based on three samples from SDA/SCL lines at MCK frequency.

- **PADFEN: PAD Filter Enable**

0: PAD analog filter is disabled.

1: PAD analog filter is enabled. (The analog filter must be enabled if high-speed mode is enabled.)

- **PADFCFG: PAD Filter Config**

See the electrical characteristics section for filter configuration details.

- **THRES: Digital Filter Threshold**

0: No filtering applied on TWI inputs.

1–7 = Maximum pulse width of spikes which will be suppressed by the input filter, defined in MCK clock cycles.

27.8.9 TWI Interrupt Enable Register

Name: TWIHS_IER

Address: 0x40018024

Access: Write-only

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	SMBHHM	SMBDAM	PECERR	TOUT	–	MCACK

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
UNRE	OVRE	GACC	SVACC	–	TXRDY	RXRDY	TXCOMP

- **TXCOMP: Transmission Completed Interrupt Enable**
- **RXRDY: Receive Holding Register Ready Interrupt Enable**
- **TXRDY: Transmit Holding Register Ready Interrupt Enable**
- **SVACC: Slave Access Interrupt Enable**
- **GACC: General Call Access Interrupt Enable**
- **OVRE: Overrun Error Interrupt Enable**
- **UNRE: Underrun 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**
- **MCACK: Master Code Acknowledge Interrupt Enable**
- **TOUT: Timeout Error Interrupt Enable**

- **PECERR: PEC Error Interrupt Enable**
- **SMBDAM: SMBus Default Address Match Interrupt Enable**
- **SMBHHM: SMBus Host Header Address Match Interrupt Enable**

0: No effect.

1: Enables the corresponding interrupt.

27.8.10 TWI Interrupt Disable Register

Name: TWIHS_IDR

Address: 0x40018028

Access: Write-only

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	SMBHHM	SMBDAM	PECERR	TOUT	–	MCACK

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
UNRE	OVRE	GACC	SVACC	–	TXRDY	RXRDY	TXCOMP

- **TXCOMP: Transmission Completed Interrupt Disable**
- **RXRDY: Receive Holding Register Ready Interrupt Disable**
- **TXRDY: Transmit Holding Register Ready Interrupt Disable**
- **SVACC: Slave Access Interrupt Disable**
- **GACC: General Call Access Interrupt Disable**
- **OVRE: Overrun Error Interrupt Disable**
- **UNRE: Underrun 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**
- **MCACK: Master Code Acknowledge Interrupt Disable**
- **TOUT: Timeout Error Interrupt Disable**

- **PECERR: PEC Error Interrupt Disable**
- **SMBDAM: SMBus Default Address Match Interrupt Disable**
- **SMBHHM: SMBus Host Header Address Match Interrupt Disable**

0: No effect.

1: Disables the corresponding interrupt.

27.8.11 TWI Interrupt Mask Register

Name: TWIHS_IMR

Address: 0x4001802C

Access: Read-only

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	SMBHHM	SMBDAM	PECERR	TOUT	–	MCACK
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
UNRE	OVRE	GACC	SVACC	–	TXRDY	RXRDY	TXCOMP

- **TXCOMP: Transmission Completed Interrupt Mask**
- **RXRDY: Receive Holding Register Ready Interrupt Mask**
- **TXRDY: Transmit Holding Register Ready Interrupt Mask**
- **SVACC: Slave Access Interrupt Mask**
- **GACC: General Call Access Interrupt Mask**
- **OVRE: Overrun Error Interrupt Mask**
- **UNRE: Underrun 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**
- **MCACK: Master Code Acknowledge Interrupt Mask**
- **TOUT: Timeout Error Interrupt Mask**

- **PECERR: PEC Error Interrupt Mask**
- **SMBDAM: SMBus Default Address Match Interrupt Mask**
- **SMBHHM: SMBus Host Header Address Match Interrupt Mask**

0: The corresponding interrupt is disabled.

1: The corresponding interrupt is enabled.

27.8.12 TWI Receive Holding Register

Name: TWIHS_RHR

Address: 0x40018030

Access: Read-only

Reset: 0x00000000

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

- **RXDATA: Master or Slave Receive Holding Data**

27.8.13 TWI Transmit Holding Register

Name: TWIHS_THR

Address: 0x40018034

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
TXDATA							

- **TXDATA: Master or Slave Transmit Holding Data**

27.8.14 TWI Write Protection Mode Register

Name: TWIHS_WPMR

Address: 0x400180E4

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 27.7.6 ”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

27.8.15 TWI Write Protection Status Register

Name: TWIHS_WPSR

Address: 0x400180E8

Access: Read-only

31	30	29	28	27	26	25	24
WPVSRC							

23	22	21	20	19	18	17	16
WPVSRC							

15	14	13	12	11	10	9	8
WPVSRC							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WPVS

- **WPVS: Write Protect Violation Status**

0: No Write Protection Violation has occurred since the last read of the TWIHS_WPSR.

1: A Write Protection Violation has occurred since the last read of the TWIHS_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.

28. Two-wire Interface (TWI)

28.1 Description

The Atmel Two-wire Interface (TWI) interconnects components on a unique two-wire bus, made up of one clock line and one data line with speeds of up to 400 Kbits per second, based on a byte-oriented transfer format. It can be used with any Atmel Two-wire Interface bus Serial EEPROM and I²C compatible device such as Real Time Clock (RTC), Dot Matrix/Graphic LCD Controllers and Temperature Sensor, to name but a few. The TWI is programmable as a master or a slave with sequential or single-byte access. Multiple master capability is supported.

Arbitration of the bus is performed internally and puts the TWI in slave mode automatically if the bus arbitration is lost.

A configurable baud rate generator permits the output data rate to be adapted to a wide range of core clock frequencies.

Table 28-1 lists the compatibility level of the Atmel Two-wire Interface in Master Mode and a full I^2C compatible device.

Table 28-1. Atmel TWI compatibility with I^2C Standard

I^2C Standard	Atmel TWI
Standard Mode Speed (100 kHz)	Supported
Fast Mode Speed (400 kHz)	Supported
7 or 10 bits Slave Addressing	Supported
START BYTE(1)	Not Supported
Repeated Start (Sr) Condition	Supported
ACK and NACK Management	Supported
Slope control and input filtering (Fast mode)	Not Supported
Clock stretching	Supported
Multi Master Capability	Supported

Note: 1. START + b000000001 + Ack + Sr

28.2 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 28-1 for details on compatibility with I²C Standard.

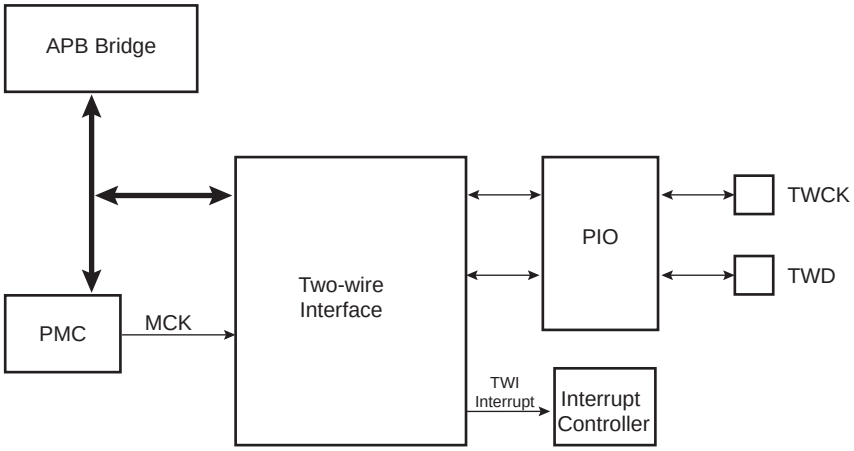
28.3 List of Abbreviations

Table 28-2. 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

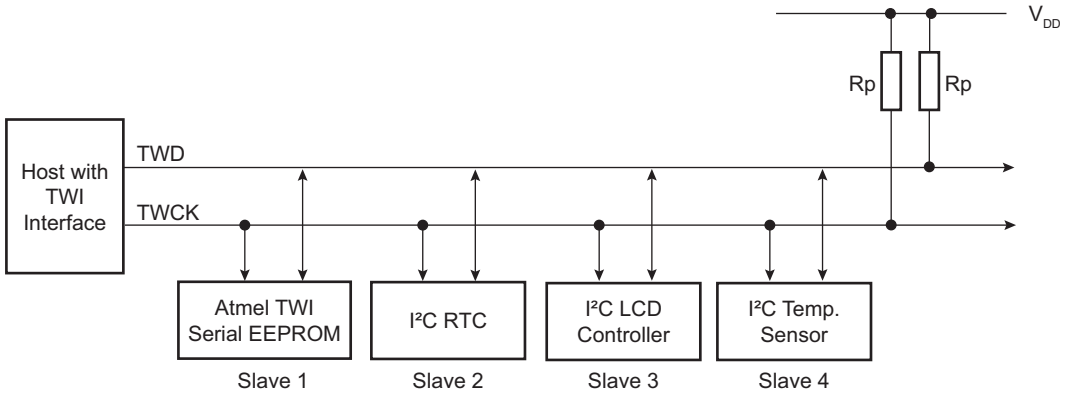
28.4 Block Diagram

Figure 28-1. Block Diagram



28.5 Application Block Diagram

Figure 28-2. Application Block Diagram



Rp: Pull-up value as given by the I²C Standard

28.5.1 I/O Lines Description

Table 28-3. I/O Lines Description

Pin Name	Pin Description	Type
TWD	Two-wire Serial Data	Input/Output
TWCK	Two-wire Serial Clock	Input/Output

28.6 Product Dependencies

28.6.1 I/O Lines

Both TWD and TWCK are bidirectional lines, connected to a positive supply voltage via a current source or pull-up resistor (see Figure 28-2). When the bus is free, both lines are high. The output stages of devices connected to the bus must have an open-drain or open-collector to perform the wired-AND function.

TWD and TWCK pins may be multiplexed with PIO lines. To enable the TWI, the programmer must perform the following step:

- Program the PIO controller to dedicate TWD and TWCK as peripheral lines.

The user must not program TWD and TWCK as open-drain. This is already done by the hardware.

28.6.2 Power Management

The TWI interface may be clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the TWI clock.

28.6.3 Interrupt

The TWI interface has an interrupt line connected to the Interrupt Controller. In order to handle interrupts, the Interrupt Controller must be programmed before configuring the TWI.

28.7 Functional Description

28.7.1 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 28-4).

Each transfer begins with a START condition and terminates with a STOP condition (see Figure 28-3).

- A high-to-low transition on the TWD line while TWCK is high defines the START condition.
- A low-to-high transition on the TWD line while TWCK is high defines a STOP condition.

Figure 28-3. START and STOP Conditions

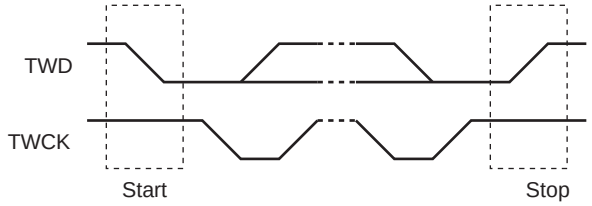
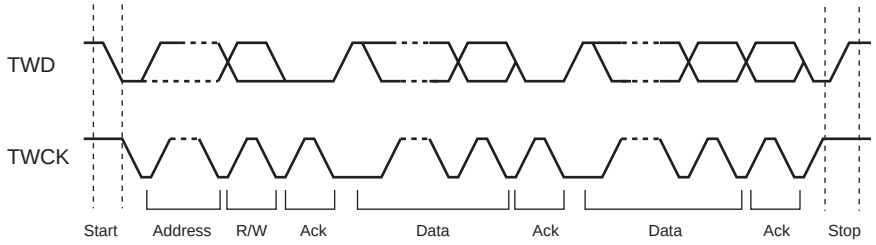


Figure 28-4. Transfer Format



28.7.2 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.

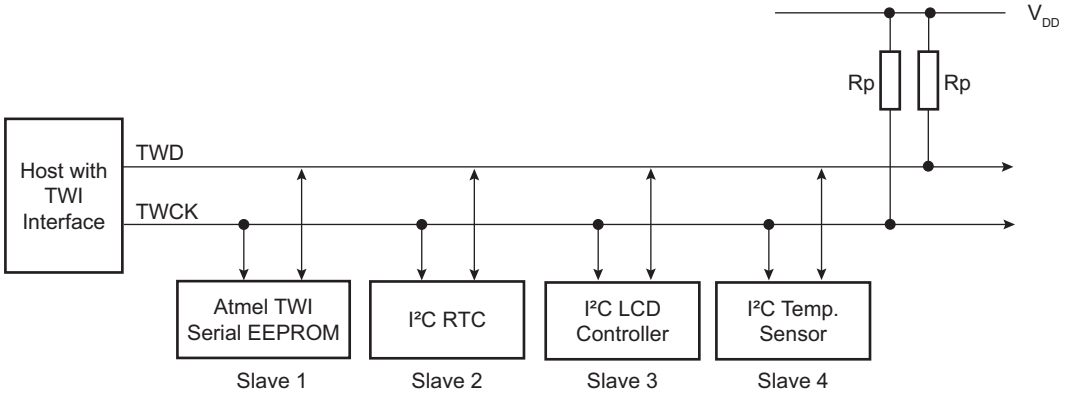
28.7.3 Master Mode

28.7.3.1 Definition

The Master is the device that starts a transfer, generates a clock and stops it.

28.7.3.2 Application Block Diagram

Figure 28-5. Master Mode Typical Application Block Diagram



Rp: Pull-up value as given by the I²C Standard

28.7.3.3 Programming Master Mode

The following registers have to be programmed before entering Master mode:

1. DADR (+ IADRSZ + IADR if a 10-bit device is addressed): The device address is used to access slave devices in read or write mode.
2. CKDIV + CHDIV + CLDIV: Clock Waveform.
3. SVDIS: Disable the slave mode.
4. MSEN: Enable the master mode.

Note: If the TWI is already in Master mode, the device address (DADR) can be configured without disabling the Master mode.

28.7.3.4 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. The master polls the data line during this clock pulse and sets the Not Acknowledge bit (NACK**)** in the status register if the slave does not acknowledge the byte. As with the other status bits, an interrupt can be generated if enabled in the Interrupt Enable Register (TWI_IER). If the slave acknowledges the byte, the data written in the TWI_THR, is then shifted in the internal shifter and transferred. When an acknowledge is detected, the TXRDY bit is set until a new write in the TWI_THR.

TXRDY is used as Transmit Ready for the PDC transmit channel.

While no new data is written in the TWI_THR, the Serial Clock Line is tied low. When new data is written in the TWI_THR, the SCL is released and the data is sent. To generate a STOP event, the STOP command must be performed by writing in the STOP field of TWI_CR.

After a Master Write transfer, the Serial Clock line is stretched (tied low) while no new data is written in the TWI_THR or until a STOP command is performed.

See Figure 28-6, Figure 28-7, and Figure 28-8.

Figure 28-6. Master Write with One Data Byte

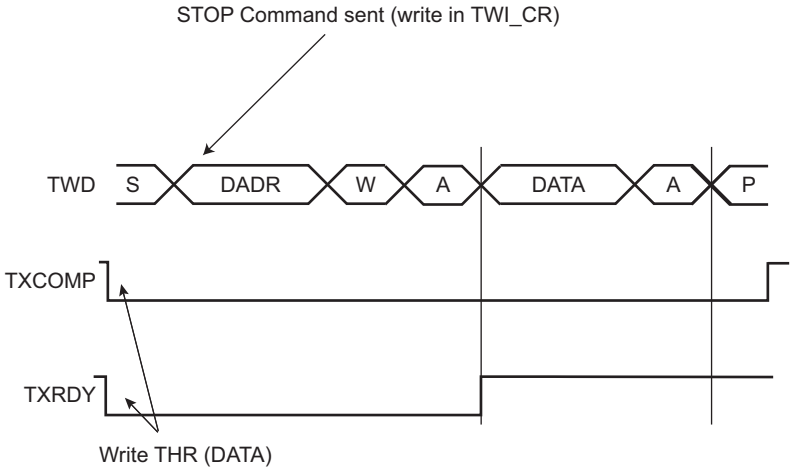


Figure 28-7. Master Write with Multiple Data Bytes

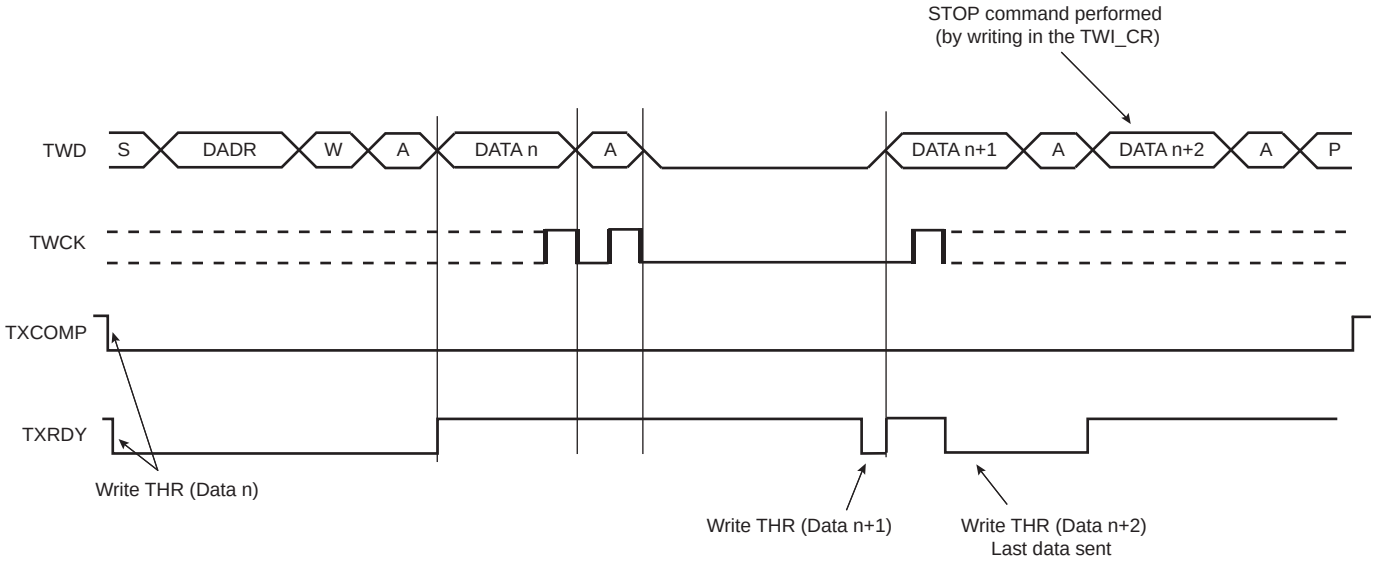
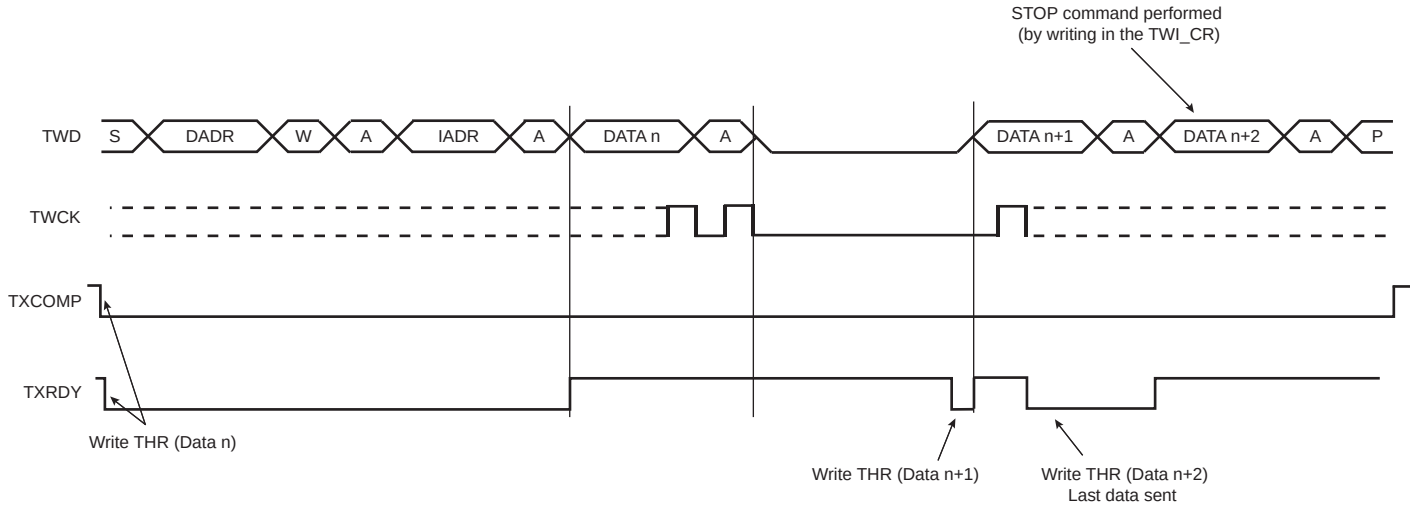


Figure 28-8. Master Write with One Byte Internal Address and Multiple Data Bytes



28.7.3.5 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 status register if the slave does not acknowledge the byte.

If an acknowledge is received, the master is then ready to receive data from the slave. After data has been received, the master sends an acknowledge condition to notify the slave that the data has been received except for the last data. See Figure 28-9. When the RXRDY bit is set in the status register, a character has been received in the Receive Holding Register (TWI_RHR). The RXRDY bit is reset when reading the TWI_RHR.

When a single data byte read is performed, with or without internal address (IADR), the START and STOP bits must be set at the same time. See Figure 28-9. When a multiple data byte read is performed, with or without internal address (IADR), the STOP bit must be set after the next-to-last data received. See Figure 28-10. For Internal Address usage see Section 28.7.3.6.

If the Receive Holding Register (TWI_RHR) is full (RXRDY high) and the master is receiving data, the Serial Clock Line will be tied low before receiving the last bit of the data and until the TWI_RHR is read. Once the TWI_RHR is read, the master will stop stretching the Serial Clock Line and end the data reception. See Figure 28-11.

Warning: When receiving multiple bytes in master read mode, if the next-to-last access is not read (the RXRDY flag remains high), the last access will not be completed until TWI_RHR is read. The last access stops on the next-to-last bit (clock stretching). When the TWI_RHR is read there is only half a bit period to send the stop bit command, else another read access might occur (spurious access).

A possible workaround is to raise the STOP BIT command before reading the TWI_RHR on the next-to-last access (within IT handler).

Figure 28-9. Master Read with One Data Byte

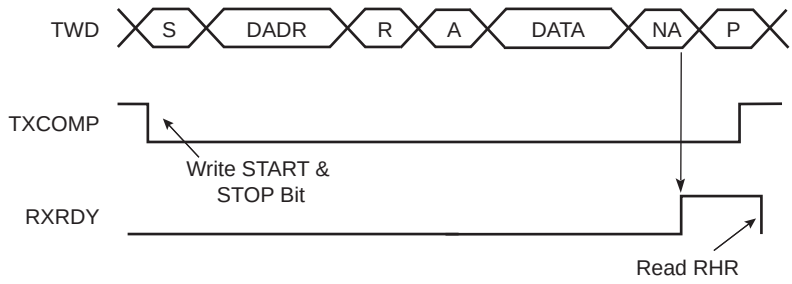


Figure 28-10. Master Read with Multiple Data Bytes

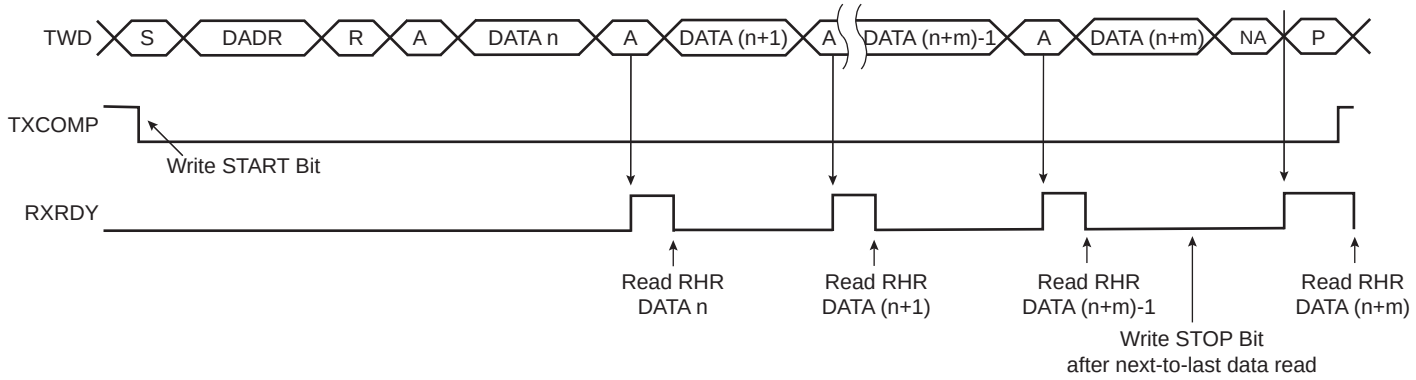
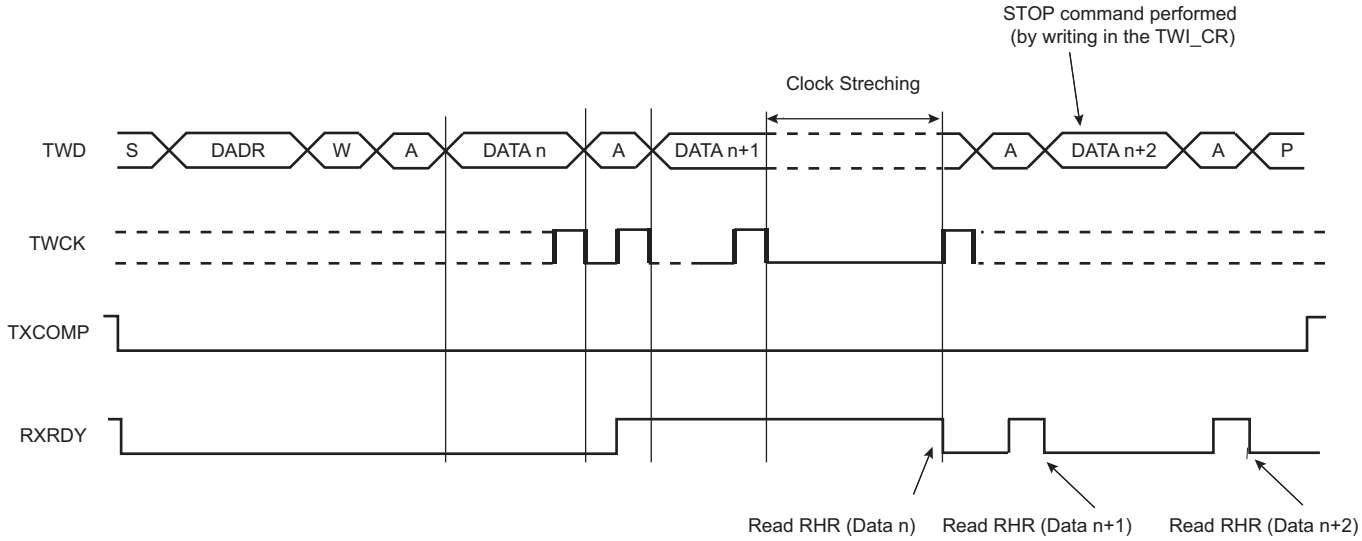


Figure 28-11. Master Read Clock Stretching with Multiple Data Bytes



RXRDY is used as Receive Ready for the PDC receive channel.

28.7.3.6 Internal Address

The TWI interface can perform various transfer formats: 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, within a memory page location in a serial memory, for example. When performing read operations with an internal address, the TWI performs a write operation to set the internal address into the slave device, and then switch to Master Receiver mode. Note that the second start condition (after sending the IADR) is sometimes called “repeated start” (Sr) in I^2C fully-compatible devices. See Figure 28-13. See Figure 28-12 and Figure 28-14 for Master Write operation with internal address.

The three internal address bytes are configurable through the Master Mode Register (TWI_MMR).

If the slave device supports only a 7-bit address, i.e., no internal address, IADRSZ must be set to 0.

Table 28-4 shows the abbreviations used in Figure 28-12 and Figure 28-13.

Table 28-4. 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 28-12. Master Write with One, Two or Three Bytes Internal Address and One Data Byte

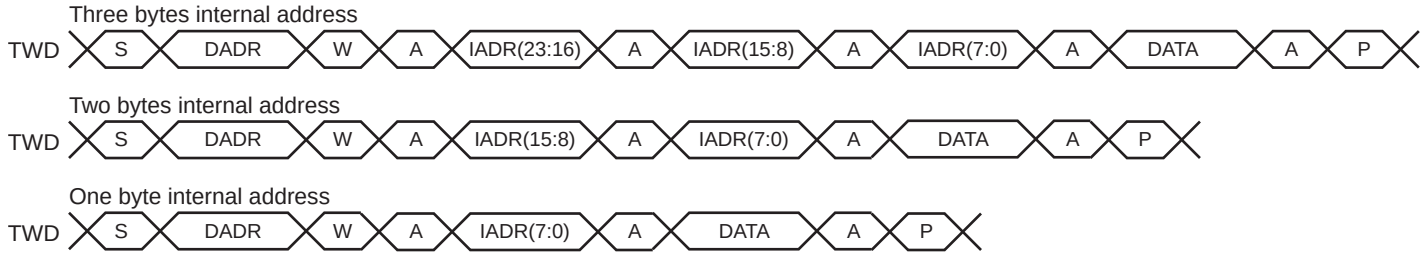
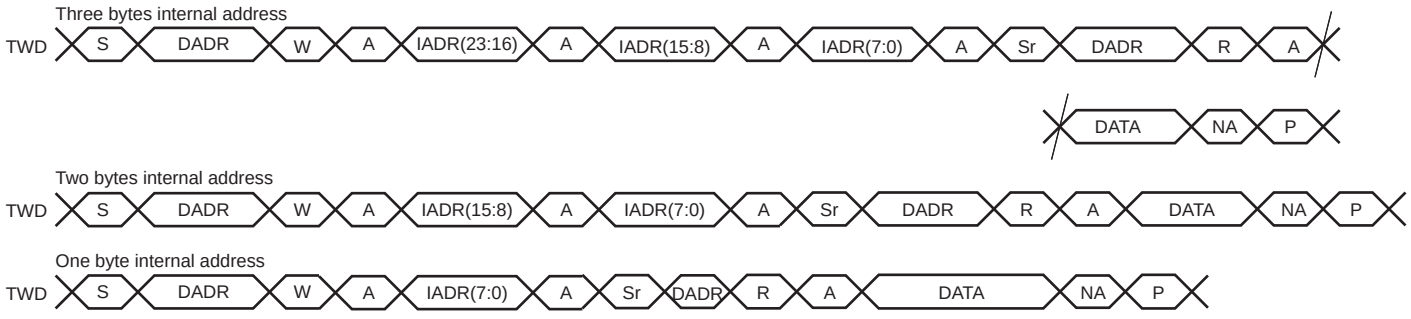


Figure 28-13. Master Read with One, Two or Three Bytes Internal Address and One Data Byte



10-bit Slave Addressing

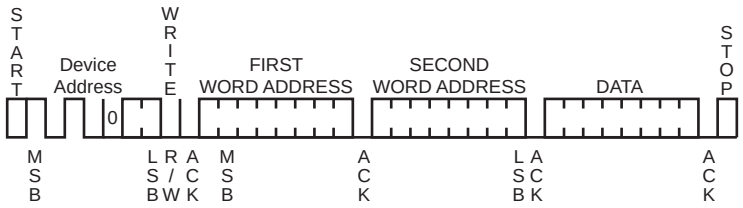
For a slave address higher than 7 bits, the user must configure the address size **(**IADRSZ) and set the other slave address bits in the Internal Address Register (TWI_IADR). The two remaining Internal address bytes, IADR[15:8] and IADR[23:16] can be used the same as in 7-bit Slave Addressing.

Example: Address a 10-bit device (10-bit device address is b1 b2 b3 b4 b5 b6 b7 b8 b9 b10)

1. Program IADRSZ = 1,
2. Program DADR with 1 1 1 1 0 b1 b2 (b1 is the MSB of the 10-bit address, b2, etc.)
3. Program TWI_IADR with b3 b4 b5 b6 b7 b8 b9 b10 (b10 is the LSB of the 10-bit address)

Figure 28-14 below shows a byte write to an Atmel AT24LC512 EEPROM. This demonstrates the use of internal addresses to access the device.

Figure 28-14. Internal Address Usage



28.7.3.7 Using the Peripheral DMA Controller (PDC)

The use of the PDC significantly reduces the CPU load.

To assure correct implementation, respect the following programming sequences:

Data Transmit with the PDC

1. Initialize the transmit PDC (memory pointers, transfer size - 1).
2. Configure the master (DADR, CKDIV, etc.) or slave mode.
3. Start the transfer by setting the PDC TXTEN bit.
4. Wait for the PDC ENDTX Flag either by using the polling method or ENDTX interrupt.
5. Disable the PDC by setting the PDC TXTDIS bit.
6. Wait for the TXRDY flag in TWI_SR.
7. Set the STOP command in TWI_CR.
8. Write the last character in TWI_THR.
9. (Optional) Wait for the TXCOMP flag in TWI_SR before disabling the peripheral clock if required.

Data Receive with the PDC

The PDC transfer size must be defined with the buffer size minus 2. The two remaining characters must be managed without PDC to ensure that the exact number of bytes are received whatever the system bus latency conditions encountered during the end of buffer transfer period.

In slave mode, the number of characters to receive must be known in order to configure the PDC.

1. Initialize the receive PDC (memory pointers, transfer size - 2).
2. Configure the master (DADR, CKDIV, etc.) or slave mode.
3. Set the PDC 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 command in TWI_CR.
9. Read the penultimate character in TWI_RHR.

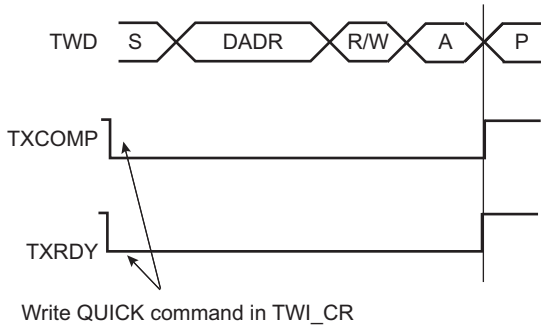
10. Wait for the RXRDY flag in TWI_SR.
11. Read the last character in TWI_RHR.
12. (Optional) Wait for the TXCOMP flag in TWI_SR before disabling the peripheral clock if required.

28.7.3.8 SMBUS Quick Command (Master Mode Only)

The TWI interface can perform a Quick Command:

1. Configure the master mode (DADR, CKDIV, etc.).
2. Write the MREAD bit in the TWI_MMR 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 28-15. SMBUS Quick Command



28.7.3.9 Read/Write Flowcharts

The following flowcharts shown in Figure 28-17 on page 631, Figure 28-18 on page 632, Figure 28-19 on page 633, Figure 28-20 on page 634 and Figure 28-21 on page 635 give examples for read and write operations. A polling or interrupt method can be used to check the status bits. The interrupt method requires that the Interrupt Enable Register (TWI_IER) be configured first.

Figure 28-16. TWI Write Operation with Single Data Byte without Internal Address

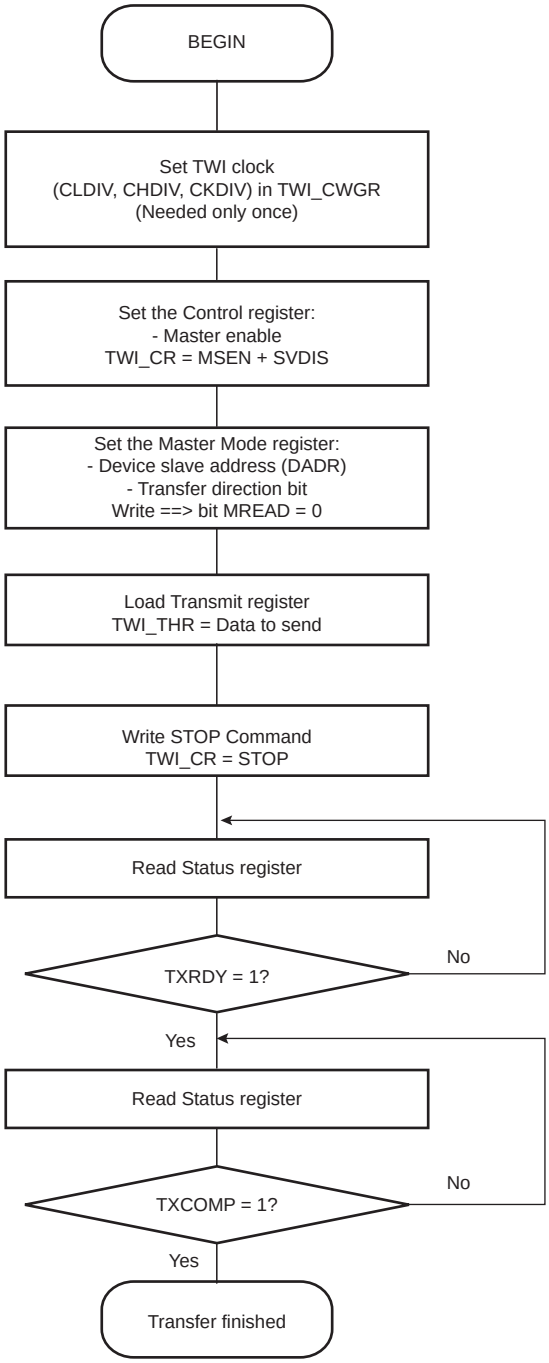


Figure 28-17. TWI Write Operation with Single Data Byte and Internal Address

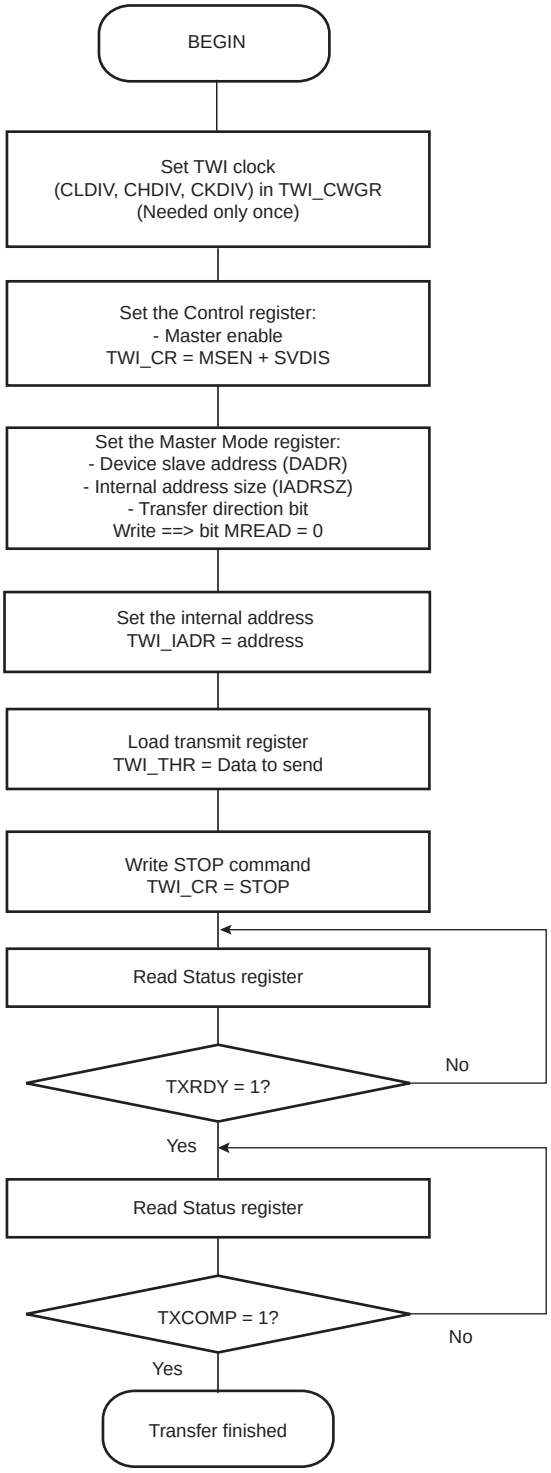


Figure 28-18. TWI Write Operation with Multiple Data Bytes with or without Internal Address

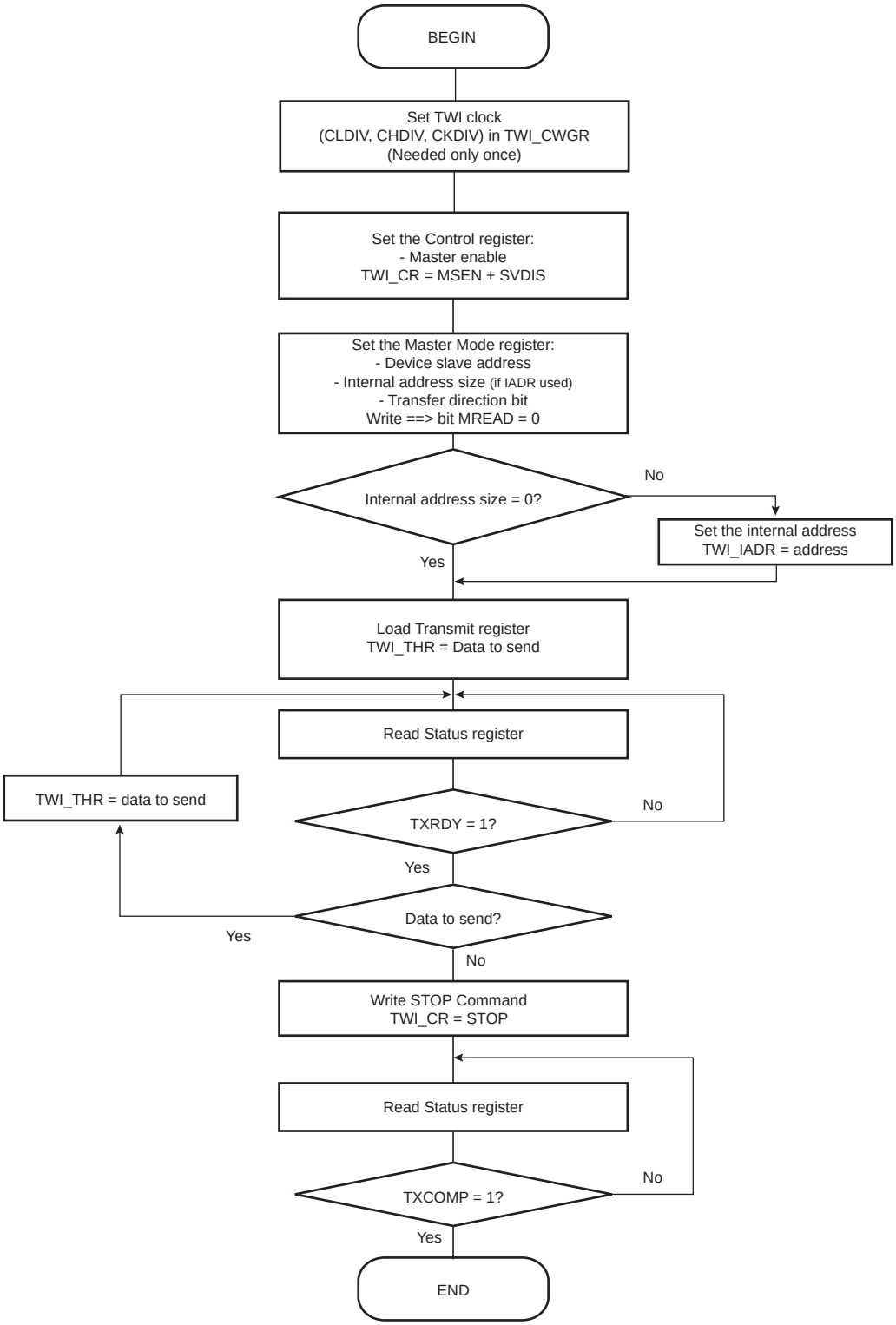


Figure 28-19. TWI Read Operation with Single Data Byte without Internal Address

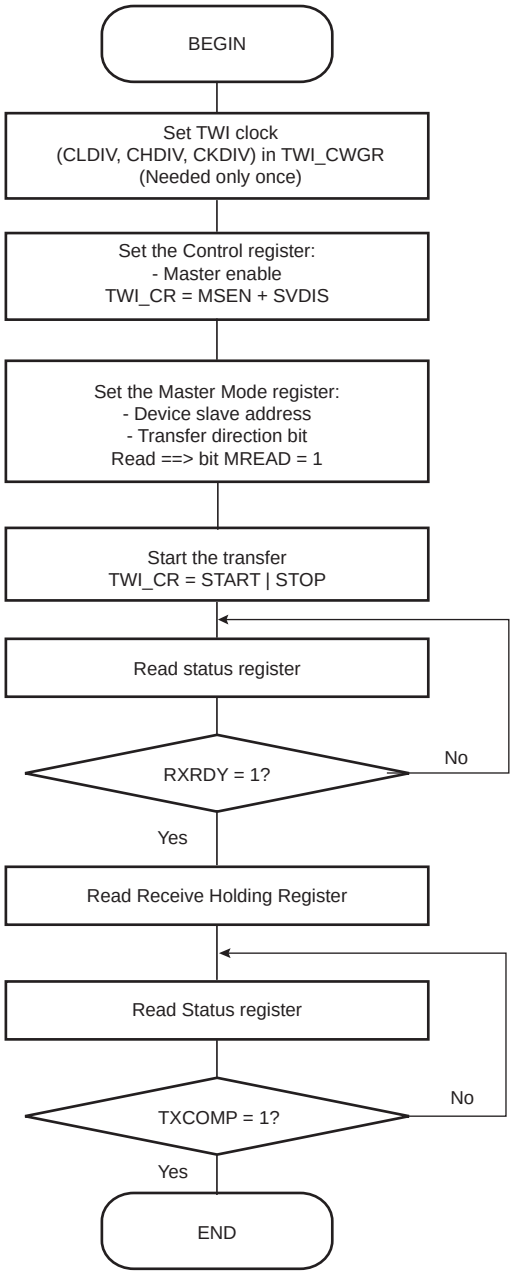


Figure 28-20. TWI Read Operation with Single Data Byte and Internal Address

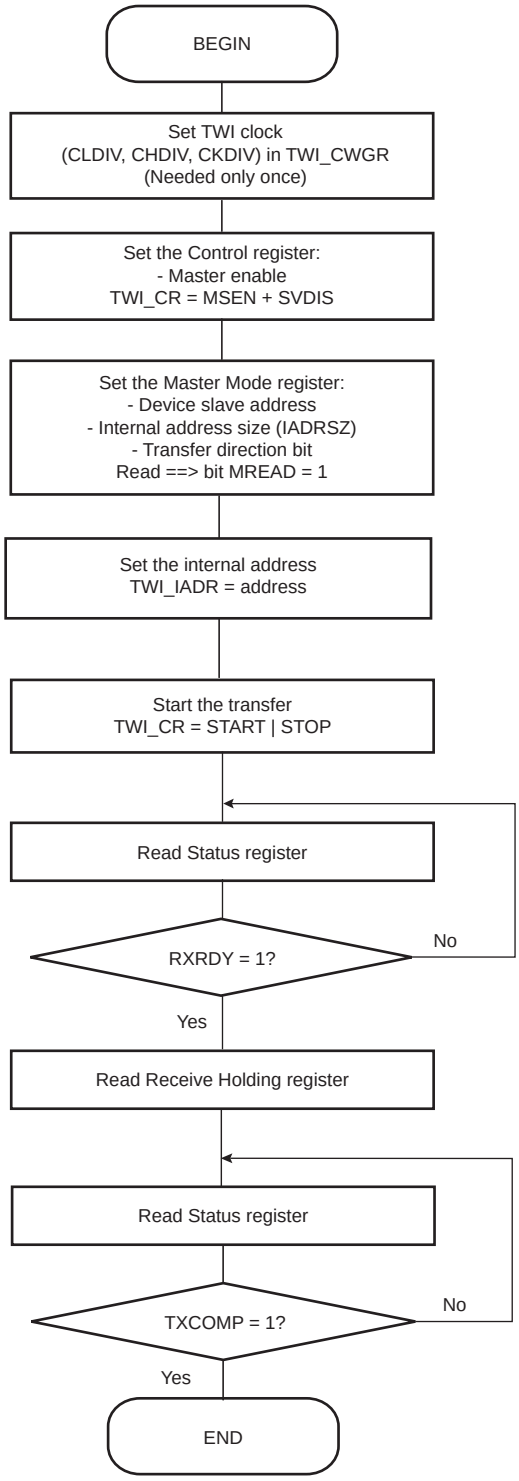
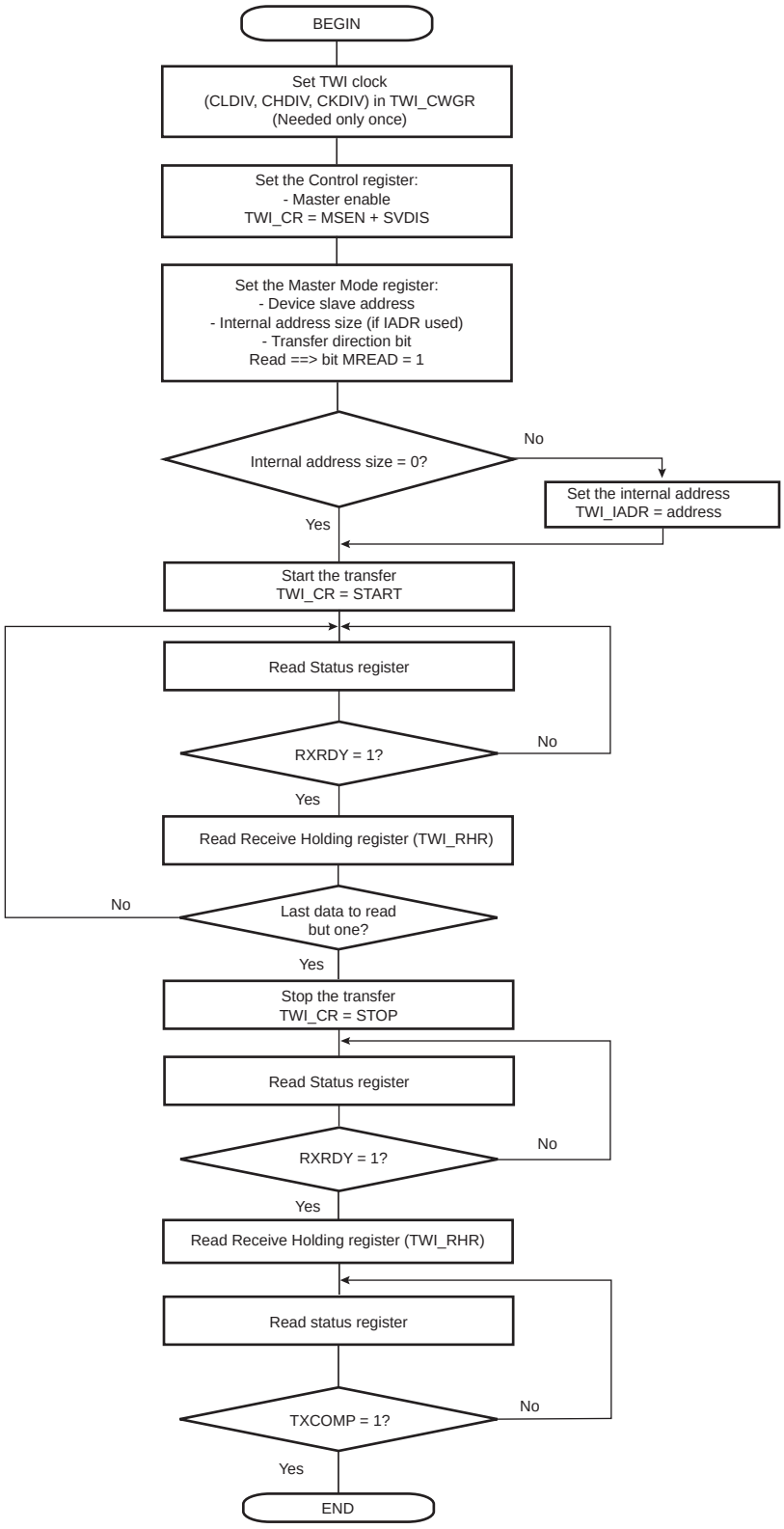


Figure 28-21. TWI Read Operation with Multiple Data Bytes with or without Internal Address



28.7.4 Multi-master Mode

28.7.4.1 Definition

More than one master may handle the bus at the same time without data corruption by using arbitration.

Arbitration starts as soon as two or more masters place information on the bus at the same time, and stops (arbitration is lost) for the master that intends to send a logical one while the other master sends a logical zero.

As soon as arbitration is lost by a master, it stops sending data and listens to the bus in order to detect a stop. When the stop is detected, the master who has lost arbitration may put its data on the bus by respecting arbitration.

Arbitration is illustrated in Figure 28-23 on page 637.

28.7.4.2 Different Multi-master Modes

Two multi-master modes may be distinguished:

1. TWI is considered as a Master only and will never be addressed.
2. TWI may be either a Master or a Slave and may be addressed.

Note: In both Multi-master modes arbitration is supported.

TWI as Master Only

In this mode, TWI is considered as a Master only (MSEN is always at one) and must be driven like a Master with the ARBLST (ARBitration Lost) flag in addition.

If arbitration is lost (ARBLST = 1), the programmer must reinitiate the data transfer.

If the user starts a transfer (ex.: DADR + START + W + Write in THR) and if the bus is busy, the TWI automatically waits for a STOP condition on the bus to initiate the transfer (see Figure 28-22 on page 637).

Note: The state of the bus (busy or free) is not indicated 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 programmer must manage the pseudo Multi-master mode described in the steps below.

1. Program TWI in Slave mode (SADR + MSDIS + SVEN) and perform Slave Access (if TWI is addressed).
2. If TWI has to be set in Master mode, wait until TXCOMP flag is at 1.
3. Program Master mode (DADR + SVDIS + MSEN) and start the transfer (ex: START + Write in THR).
4. As soon as the Master mode is enabled, TWI scans the bus in order to detect if it is busy or free. When the bus is considered as free, TWI initiates the transfer.
5. As soon as the transfer is initiated and until a STOP condition is sent, the arbitration becomes relevant and the user must monitor the ARBLST flag.
6. If the arbitration is lost (ARBLST is set to 1), the user must program the TWI in Slave mode in the case where the Master that won the arbitration wanted to access the TWI.
7. If TWI has to be set in Slave mode, wait until TXCOMP flag is at 1 and then program the Slave mode.

Note: In the case where the arbitration is lost and TWI is addressed, TWI will not acknowledge even if it is programmed in Slave mode as soon as ARBLST is set to 1. Then, the Master must repeat SADR.

Figure 28-22. Programmer Sends Data While the Bus is Busy

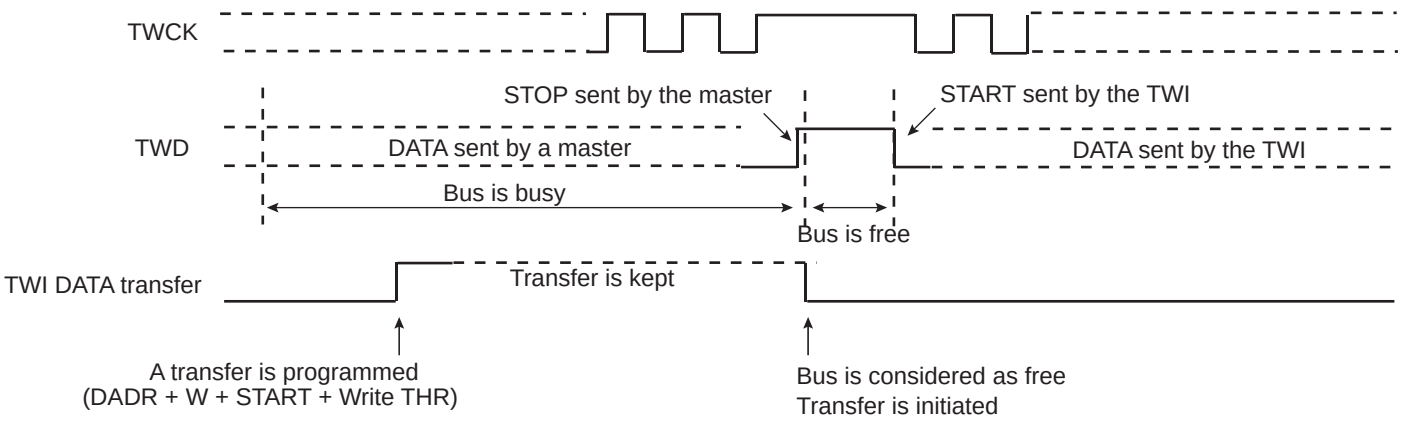
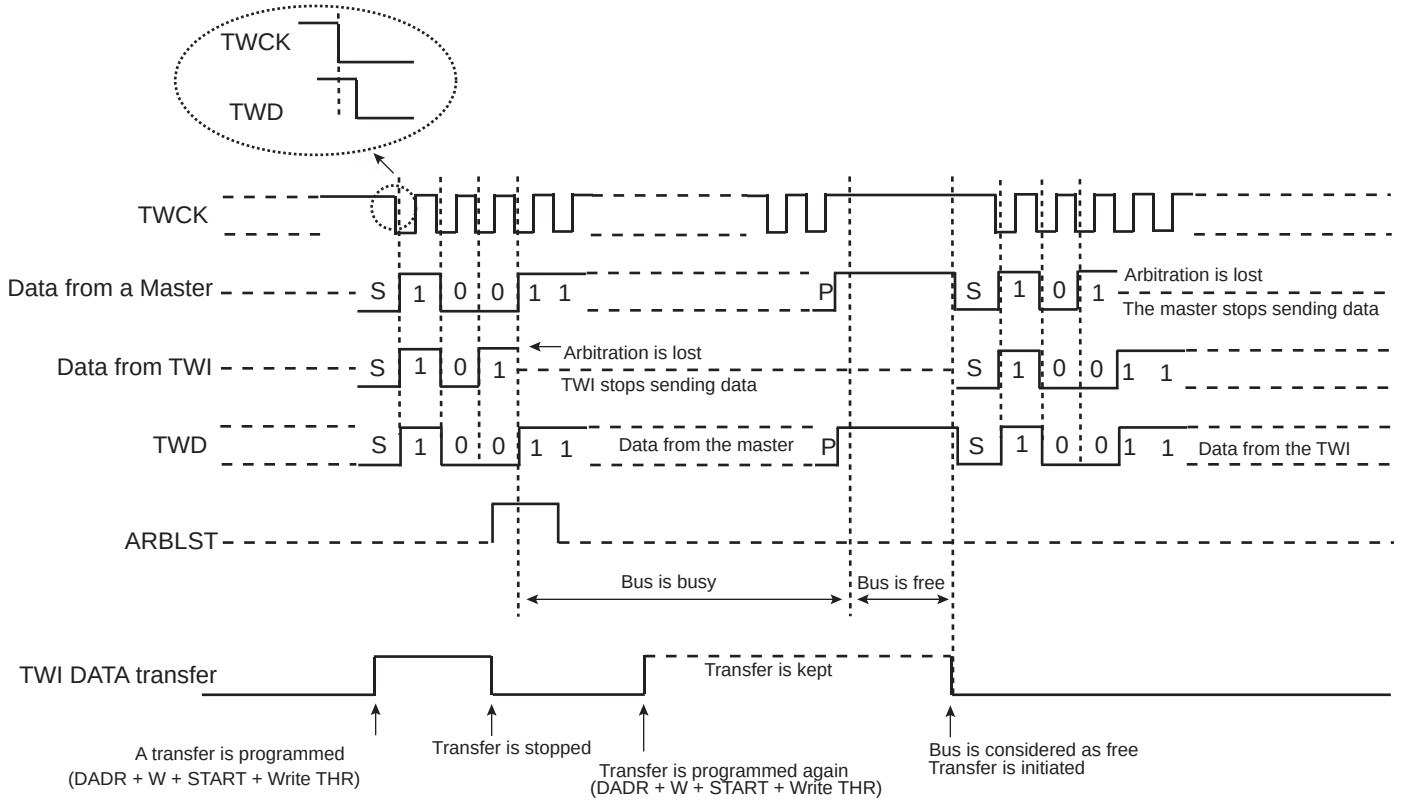
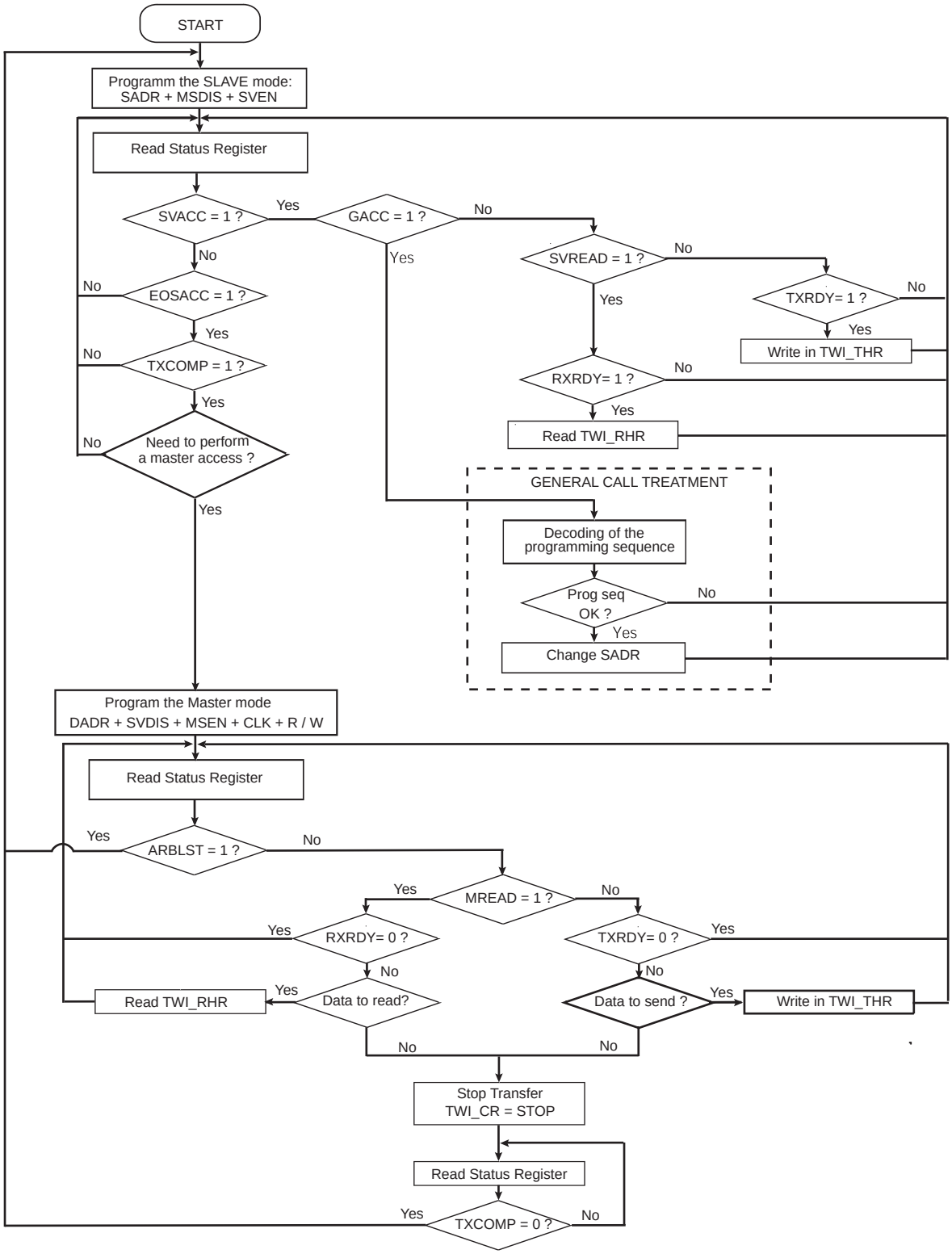


Figure 28-23. Arbitration Cases



The flowchart shown in Figure 28-24 on page 638 gives an example of read and write operations in Multi-master mode.

Figure 28-24. Multi-master Flowchart



28.7.5 Slave Mode

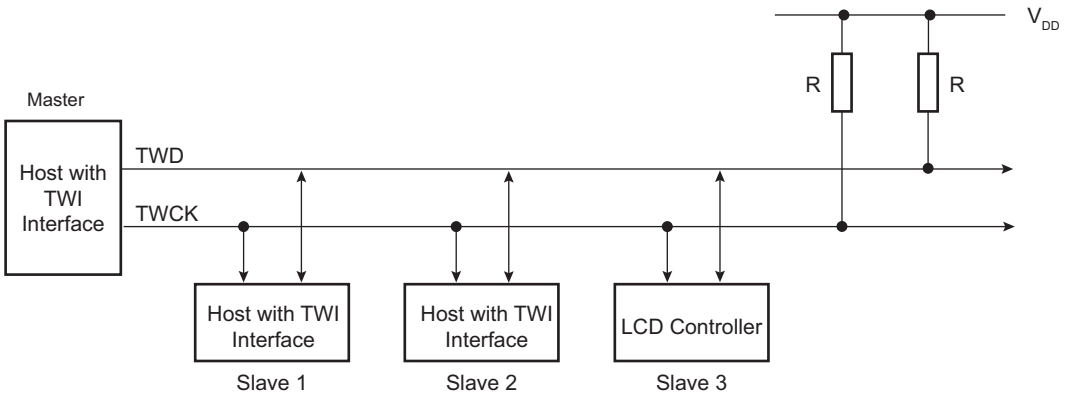
28.7.5.1 Definition

The Slave Mode is defined as a mode where the device receives the clock and the address from another device called the master.

In this mode, the device never initiates and never completes the transmission (START, REPEATED_START and STOP conditions are always provided by the master).

28.7.5.2 Application Block Diagram

Figure 28-25. Slave Mode Typical Application Block Diagram



28.7.5.3 Programming Slave Mode

The following fields must be programmed before entering Slave mode:

1. SADR (TWI_SMR): The slave device address is used in order to be accessed by master devices in read or write mode.
2. MSDIS (TWI_CR): Disable the master mode.
3. SVEN (TWI_CR): Enable the slave mode.

As the device receives the clock, values written in TWI_CWGR are not taken into account.

28.7.5.4 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, 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.

See Figure 28-26 on page 640.

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 28-27 on page 641.

Clock Synchronization Sequence

In the case where TWI_THR or TWI_RHR is not written/read in time, TWI performs a clock synchronization.

Clock stretching information is given by the SCLWS (Clock Wait state) bit.

See Figure 28-29 on page 642 and Figure 28-30 on page 643.

General Call

In the case where a GENERAL CALL is performed, GACC (General Call ACCess) flag is set.

After GACC is set, it is up to the programmer to interpret the meaning of the GENERAL CALL and to decode the new address programming sequence.

See Figure 28-28 on page 641.

28.7.5.5 Data Transfer

Read Operation

The read mode is defined as a data requirement from the master.

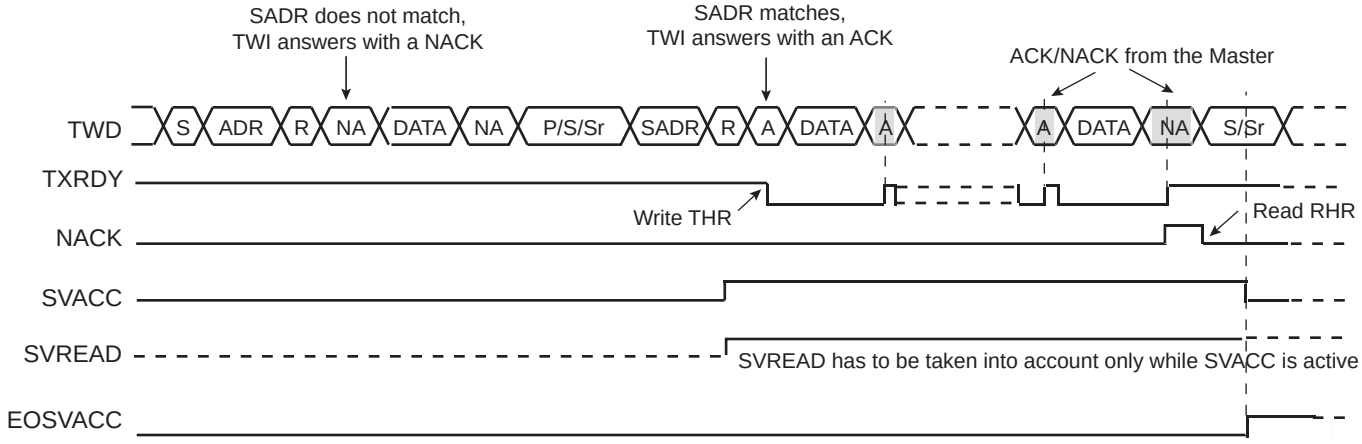
After a START or a REPEATED START condition is detected, the decoding of the address starts. If the slave address (SADR) is decoded, SVACC is set and SVREAD indicates the direction of the transfer.

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 28-26 on page 640 describes the write operation.

Figure 28-26. Read Access Ordered by a MASTER



Notes:
1. When SVACC is low, the state of SVREAD becomes irrelevant.
2. TXRDY is reset when data has been transmitted from TWI_THR to the 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.

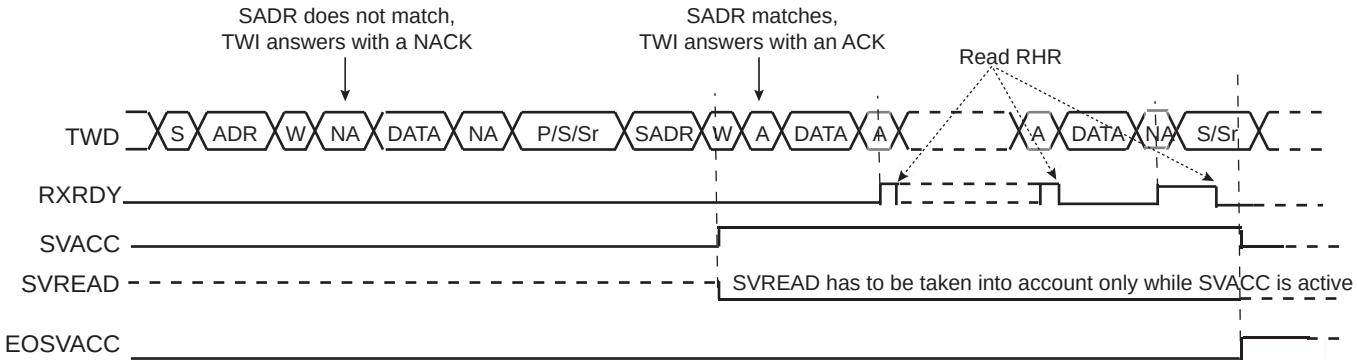
After a START or a REPEATED START, the decoding of the address starts. If the slave address is decoded, SVACC is set and SVREAD indicates the direction of the transfer (SVREAD is low in this case).

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 28-27 describes the Write operation.

Figure 28-27. Write Access Ordered by a Master



Notes:
1. When SVACC is low, the state of SVREAD becomes irrelevant.
2. RXRDY is set when data has been transmitted from the 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.

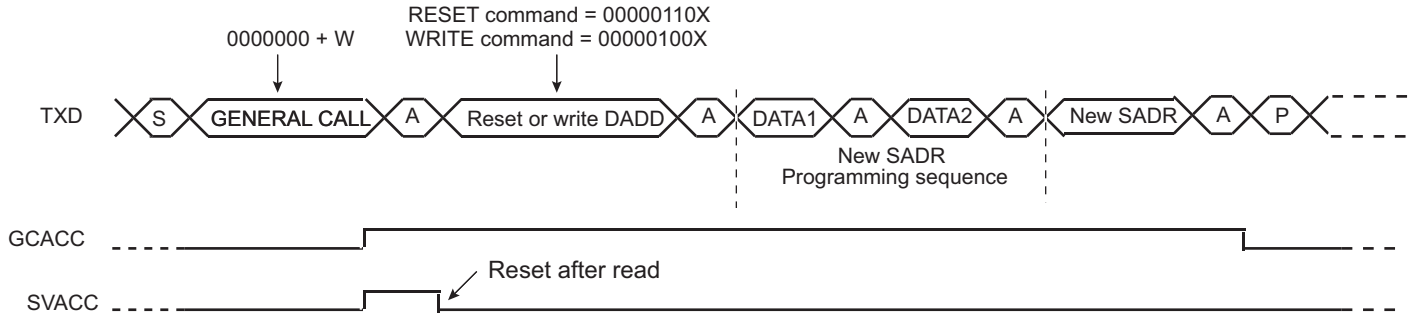
If a GENERAL CALL is detected, GACC is set.

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 28-28 describes the General Call access.

Figure 28-28. Master Performs a General Call



Note: This method allows the user to create an own programming sequence by choosing the programming bytes and the number of them. The programming sequence has to be provided to the master.

Clock Synchronization

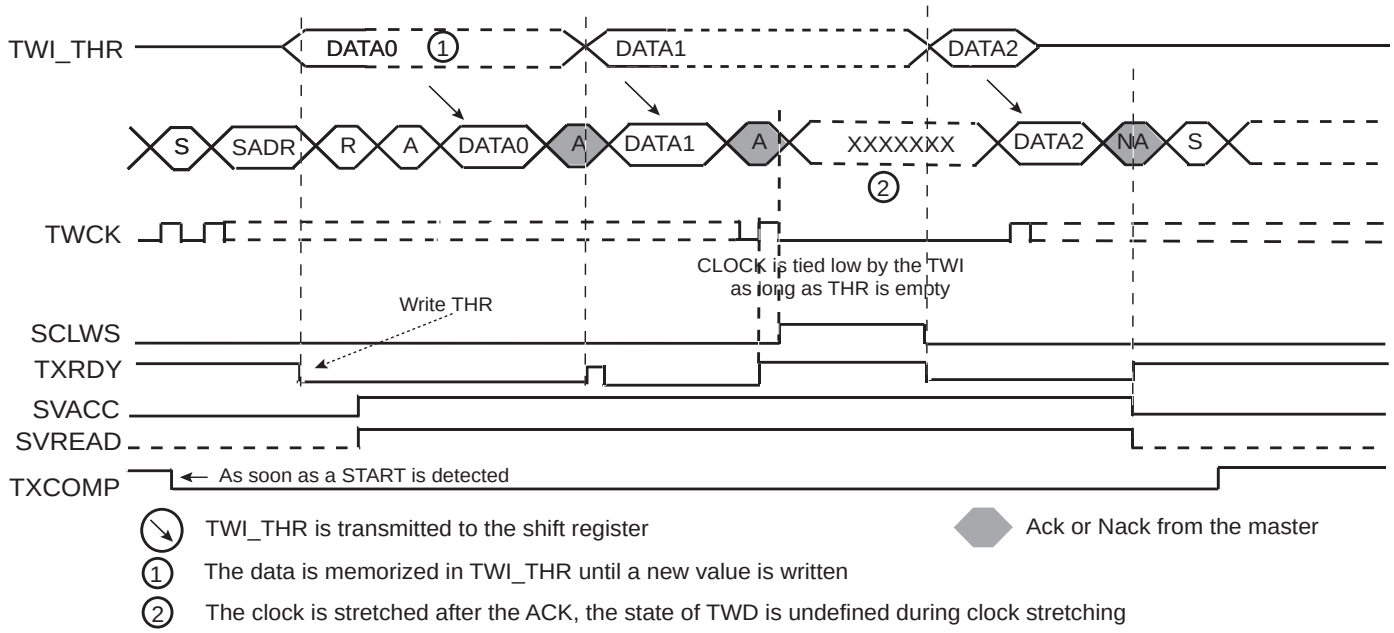
In both read and write modes, it may happen that TWI_THR/TWI_RHR buffer is not filled /emptied before the emission/reception of a new character. In this case, to avoid sending/receiving undesired data, a clock stretching mechanism is implemented.

— Clock Synchronization in Read Mode

The clock is tied low if the 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 28-29 describes the clock synchronization in Read mode.

Figure 28-29. Clock Synchronization in Read Mode



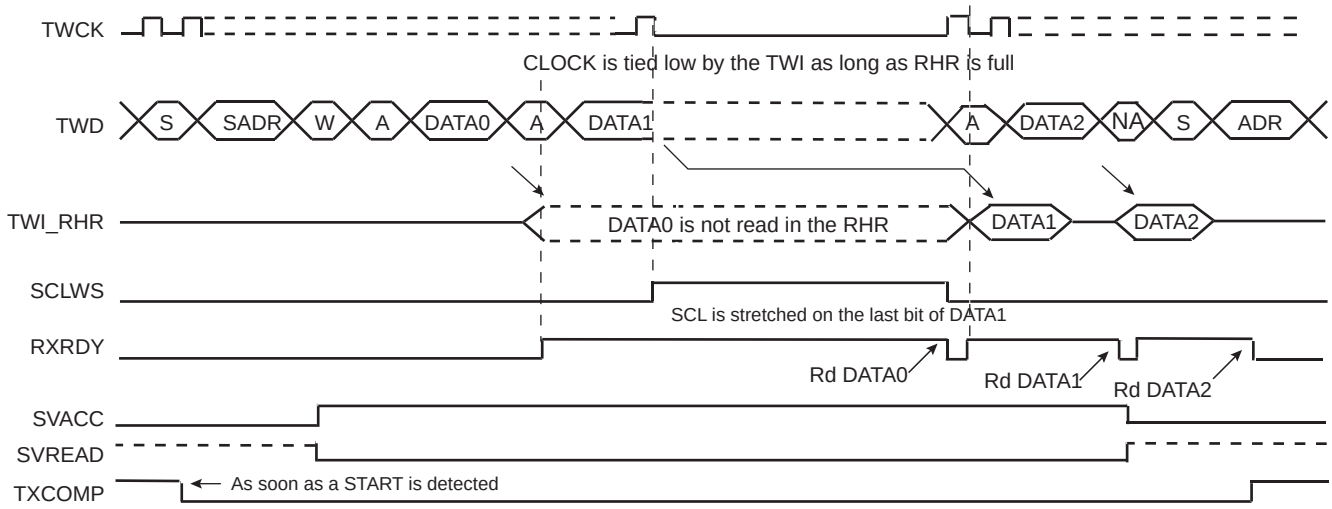
Notes:
1. TXRDY is reset when data has been written in the TWI_THR to the internal shifter and set when this data has been acknowledged or non acknowledged.
2. At the end of the read sequence, TXCOMP is set after a STOP or after a REPEATED_START + an address different from SADR.
3. SCLWS is automatically set when the clock synchronization mechanism is started.

— Clock Synchronization in Write Mode

The clock is tied low 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 28-30 describes the clock synchronization in Read mode.

Figure 28-30. Clock Synchronization in Write Mode



Notes: 1. At the end of the read sequence, TXCOMP is set after a STOP or after a REPEATED_START + an address different from SADR.

2. SCLWS is automatically set when the clock synchronization mechanism is started and automatically reset when the mechanism is finished.

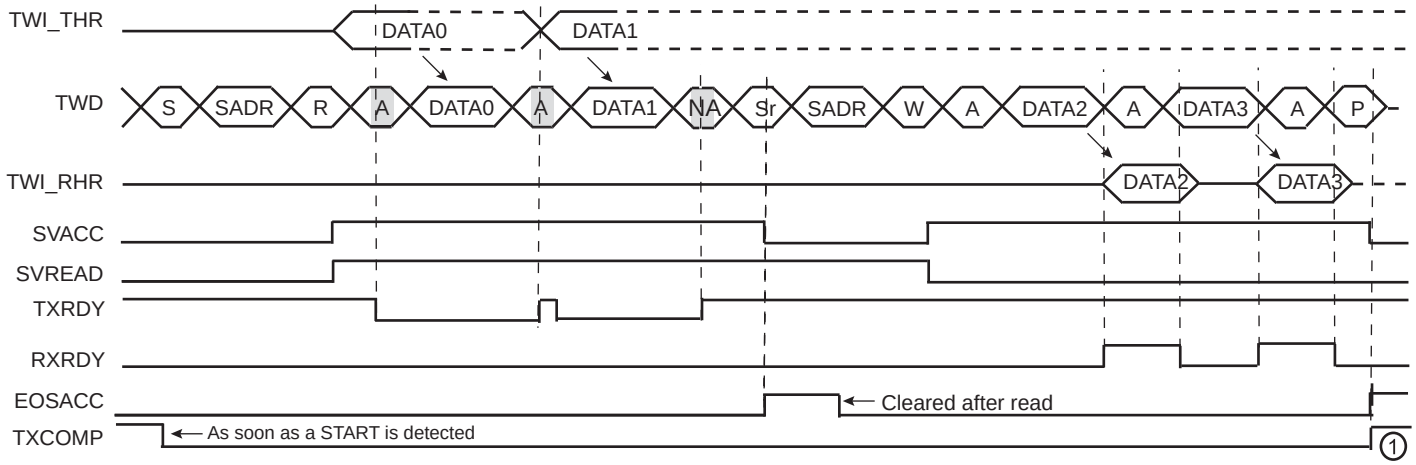
Reversal after a Repeated Start

— Reversal of Read to Write

The master initiates the communication by a read command and finishes it by a write command.

Figure 28-31 describes the repeated start + reversal from Read to Write mode.

Figure 28-31. Repeated Start + Reversal from Read to Write Mode

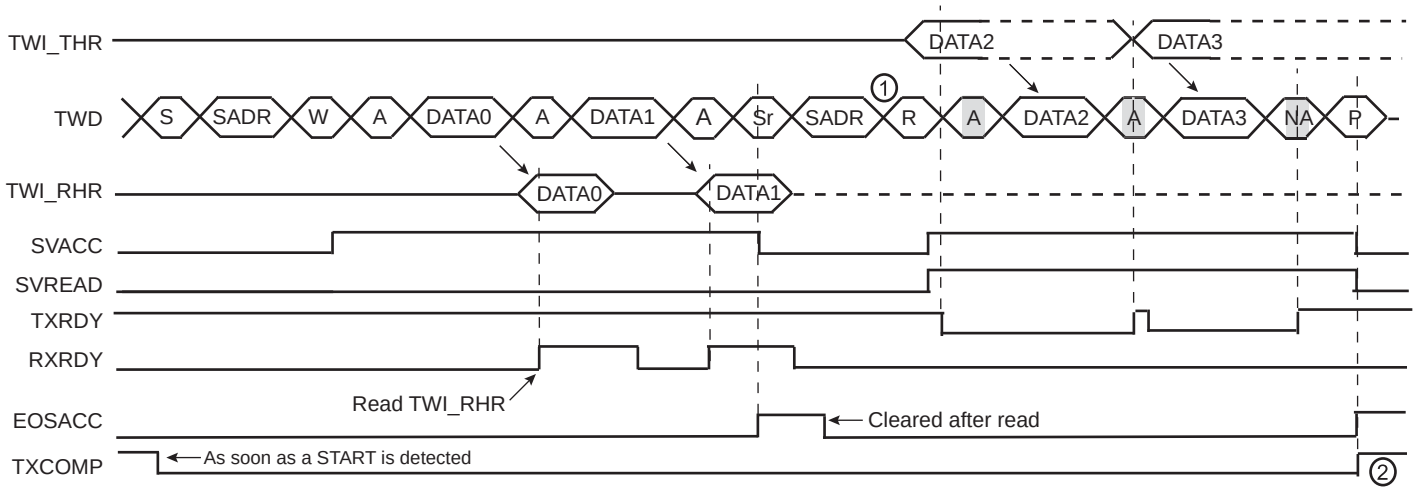


Note: 1. TXCOMP is only set at the end of the transmission because after the repeated start, SADR is detected again.

— Reversal of Write to Read

The master initiates the communication by a write command and finishes it by a read command. Figure 28-32 describes the repeated start + reversal from Write to Read mode.

Figure 28-32. Repeated Start + Reversal from Write to Read Mode



Notes:
1. In this case, if TWI_THR has not been written at the end of the read command, the clock is automatically stretched before the ACK.
2. TXCOMP is only set at the end of the transmission because after the repeated start, SADR is detected again.

28.7.5.6 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 to transmit data 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 either by using the polling method or ENDTX interrupt.
4. Disable the PDC by setting the PDC TXTDIS bit.
5. (Optional) Wait for the TXCOMP flag in TWI_SR before disabling the peripheral clock if required.

Data Receive with the PDC in Slave Mode

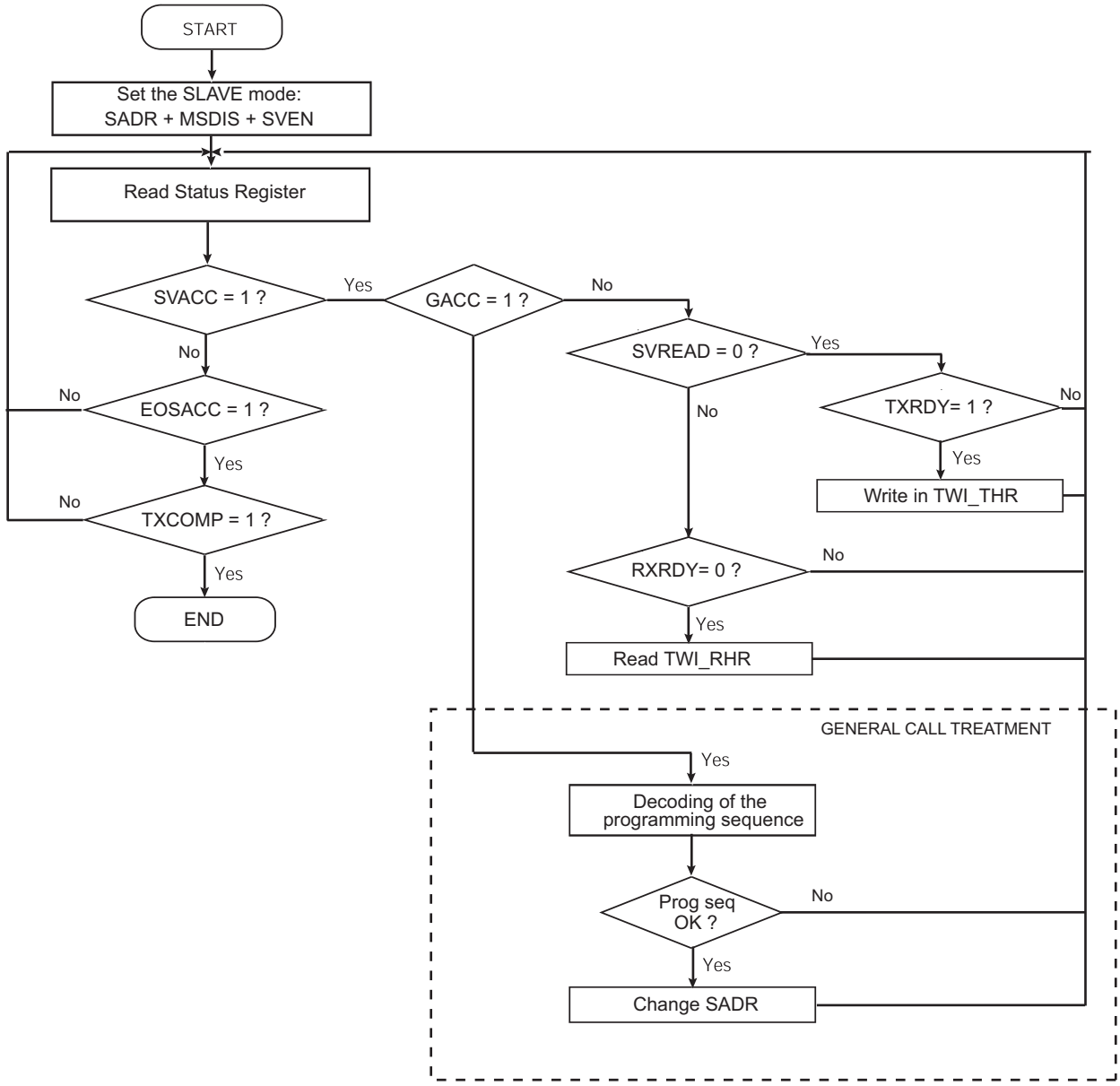
The following procedure shows an example to transmit data with PDC where the number of characters to receive is known.

1. Initialize the receive PDC (memory pointers, transfer size).
2. Set the PDC RXTEN bit.
3. Wait for the PDC ENDRX flag either by using polling method or ENDRX interrupt.
4. Disable the PDC by setting the PDC RXTDIS bit.
5. (Optional) Wait for the TXCOMP flag in TWI_SR before disabling the peripheral clock if required.

28.7.5.7 Read Write Flowcharts

The flowchart shown in Figure 28-33 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 28-33. Read Write Flowchart in Slave Mode



28.7.6 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 field WPVSRC indicates 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

28.8 Two-wire Interface (TWI) User Interface

Table 28-5. 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	0x00000000
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”.

28.8.1 TWI Control Register

Name: TWI_CR

Access: Write-only

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
SWRST	QUICK	SVDIS	SVEN	MSDIS	MSEN	STOP	START

- **START: Send a START Condition**

0: No effect.

1: A frame beginning with a START bit is transmitted according to the features defined in the mode register.

This action is necessary when the TWI peripheral wants to read data from a slave. When configured in Master Mode with a write operation, a frame is sent as soon as the user writes a character in the Transmit Holding Register (TWI_THR).

- **STOP: Send a STOP Condition**

0: No effect.

1: STOP Condition is sent just after completing the current byte transmission in master read mode.

  – In single data byte master read, the START and STOP must both be set.
  – In multiple data bytes master read, the STOP must be set after the last data received but one.
  – In master read mode, if a NACK bit is received, the STOP is automatically performed.
  – In master data write operation, a STOP condition will be sent after the transmission of the current data is finished.

- **MSEN: TWI Master Mode Enabled**

0: No effect.

1: 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.

28.8.2 TWI Master Mode Register

Name: TWI_MMR

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	DADR						

15	14	13	12	11	10	9	8
–	–	–	MREAD	–	–	IADRSZ	

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

- **IADRSZ: Internal Device Address Size**

Value	Name	Description
0	NONE	No internal device address
1	1_BYTE	One-byte internal device address
2	2_BYTE	Two-byte internal device address
3	3_BYTE	Three-byte internal device address

- **MREAD: Master Read Direction**

0: Master write direction.

1: Master read direction.

- **DADR: Device Address**

The device address is used to access slave devices in read or write mode. Those bits are only used in Master mode.

28.8.3 TWI Slave Mode Register

Name: TWI_SMR

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	SADR						

15	14	13	12	11	10	9	8
–	–	–	–	–	–		

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

This register can only be written if the WPEN bit is cleared in the TWI Write Protection Mode Register.

- **SADR: Slave Address**

The slave device address is used in Slave mode in order to be accessed by master devices in read or write mode.

SADR must be programmed before enabling the Slave mode or after a general call. Writes at other times have no effect.

28.8.4 TWI Internal Address Register

Name: TWI_IADR

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
IADR							

15	14	13	12	11	10	9	8
IADR							

7	6	5	4	3	2	1	0
IADR							

- **IADR: Internal Address**

0, 1, 2 or 3 bytes depending on IADRSZ.

28.8.5 TWI Clock Waveform Generator Register

Name: TWI_CWGR

Access: Read/Write

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	HOLD				

23	22	21	20	19	18	17	16
–	–	–	–	–	CKDIV		

15	14	13	12	11	10	9	8
CHDIV							

7	6	5	4	3	2	1	0
CLDIV							

This register can only be written if the WPEN bit is cleared in the TWI Write Protection Mode Register.

TWI_CWGR is only used in Master mode.

- **CLDIV: Clock Low Divider**

The SCL low period is defined as follows:

$$t_{low} = ((\text{CLDIV} \times 2^{\text{CKDIV}}) + 4) \times t_{MCK}$$

- **CHDIV: Clock High Divider**

The SCL high period is defined as follows:

$$t_{high} = ((\text{CHDIV} \times 2^{\text{CKDIV}}) + 4) \times t_{MCK}$$

- **CKDIV: Clock Divider**

The CKDIV is used to increase both SCL high and low periods.

- **HOLD: TWD Hold Time versus TWCK falling**

TWD is kept unchanged after TWCK falling edge for a period of (HOLD+3)*t_{MCK}.

28.8.6 TWI Status Register

Name: TWI_SR

Access: Read-only

Reset: 0x0000F009

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	ENDTX	ENDRX	EOSACC	SCLWS	ARBLST	NACK

7	6	5	4	3	2	1	0
–	OVRE	GACC	SVACC	SVREAD	TXRDY	RXRDY	TXCOMP

- **TXCOMP: Transmission Completed (automatically set / reset)**

TXCOMP used in Master mode:

0: During the length of the current frame.

1: When both holding register and internal shifter are empty and STOP condition has been sent.

TXCOMP behavior in Master mode can be seen in Figure 28-8 on page 625 and in Figure 28-10 on page 626.

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 28-29 on page 642, Figure 28-30 on page 643, Figure 28-31 on page 643 and Figure 28-32 on page 644.

- **RXRDY: Receive Holding Register Ready (automatically set / reset)**

0: No character has been received since the last TWI_RHR read operation.

1: A byte has been received in the TWI_RHR since the last read.

RXRDY behavior in Master mode can be seen in Figure 28-10 on page 626.

RXRDY behavior in Slave mode can be seen in Figure 28-27 on page 641, Figure 28-30 on page 643, Figure 28-31 on page 643 and Figure 28-32 on page 644.

- **TXRDY: Transmit Holding Register Ready (automatically set / reset)**

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 28.7.3.4 on page 623.

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 28-26 on page 640, Figure 28-29 on page 642, Figure 28-31 on page 643 and Figure 28-32 on page 644.

- **SVREAD: Slave Read (automatically set / reset)**

This bit is only used in Slave mode. When SVACC is low (no Slave access has been detected) SVREAD is irrelevant.

0: Indicates that a write access is performed by a Master.

1: Indicates that a read access is performed by a Master.

SVREAD behavior can be seen in Figure 28-26 on page 640, Figure 28-27 on page 641, Figure 28-31 on page 643 and Figure 28-32 on page 644.

- **SVACC: Slave Access (automatically set / reset)**

This bit is only used in Slave mode.

0: TWI is not addressed. SVACC is automatically cleared after a NACK or a STOP condition is detected.

1: Indicates that the address decoding sequence has matched (A Master has sent SADR). SVACC remains high until a NACK or a STOP condition is detected.

SVACC behavior can be seen in Figure 28-26 on page 640, Figure 28-27 on page 641, Figure 28-31 on page 643 and Figure 28-32 on page 644.

- **GACC: General Call Access (clear on read)**

This bit is only used in Slave mode.

0: No General Call has been detected.

1: A General Call has been detected. After the detection of General Call, if need be, the programmer may acknowledge this access and decode the following bytes and respond according to the value of the bytes.

GACC behavior can be seen in Figure 28-28 on page 641.

- **OVRE: Overrun Error (clear on read)**

This bit is only used in Master mode.

0: TWI_RHR has not been loaded while RXRDY was set

1: TWI_RHR has been loaded while RXRDY was set. Reset by read in TWI_SR when TXCOMP is set.

- **NACK: Not Acknowledged (clear on read)**

NACK used in Master mode:

0: Each data byte has been correctly received by the far-end side TWI slave component.

1: A data byte 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 it means that the Master will stop the data transfer or re initiate it.

Note that in Slave Write mode all data are acknowledged by the TWI.

- **ARBLST: Arbitration Lost (clear on read)**

This bit is only used in Master mode.

0: Arbitration won.

1: Arbitration lost. Another master of the TWI bus has won the multi-master arbitration. TXCOMP is set at the same time.

- **SCLWS: Clock Wait State (automatically set / reset)**

This bit is only used in Slave mode.

0: The clock is not stretched.

1: The clock is stretched. TWI_THR / TWI_RHR buffer is not filled / emptied before the emission / reception of a new character.

SCLWS behavior can be seen in Figure 28-29 on page 642 and Figure 28-30 on page 643.

- **EOSACC: End Of Slave Access (clear on read)**

This bit is only used in Slave mode.

0: A slave access is being performing.

1: The Slave Access is finished. End Of Slave Access is automatically set as soon as SVACC is reset.

EOSACC behavior can be seen in Figure 28-31 on page 643 and Figure 28-32 on page 644

- **ENDRX: End of RX buffer**

0: The Receive Counter Register has not reached 0 since the last write in TWI_RCR or TWI_RNCR.

1: The Receive Counter Register has reached 0 since the last write in TWI_RCR or TWI_RNCR.

- **ENDTX: End of TX buffer**

0: The Transmit Counter Register has not reached 0 since the last write in TWI_TCR or TWI_TNCR.

1: The Transmit Counter Register has reached 0 since the last write in TWI_TCR or TWI_TNCR.

- **RXBUFF: RX Buffer Full**

0: TWI_RCR or TWI_RNCR have a value other than 0.

1: Both TWI_RCR and TWI_RNCR have a value of 0.

- **TXBUFE: TX Buffer Empty**

0: TWI_TCR or TWI_TNCR have a value other than 0.

1: Both TWI_TCR and TWI_TNCR have a value of 0.

28.8.7 TWI Interrupt Enable Register

Name: TWI_IER

Access: Write-only

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	ENDTX	ENDRX	EOSACC	SCL_WS	ARBLST	NACK
7	6	5	4	3	2	1	0
–	OVRE	GACC	SVACC	–	TXRDY	RXRDY	TXCOMP

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**

28.8.8 TWI Interrupt Disable Register

Name: TWI_IDR

Access: Write-only

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	ENDTX	ENDRX	EOSACC	SCL_WS	ARBLST	NACK

7	6	5	4	3	2	1	0
–	OVRE	GACC	SVACC	–	TXRDY	RXRDY	TXCOMP

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**

28.8.9 TWI Interrupt Mask Register

Name: TWI_IMR

Access: Read-only

Reset: 0x00000000

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	ENDTX	ENDRX	EOSACC	SCL_WS	ARBLST	NACK
7	6	5	4	3	2	1	0
–	OVRE	GACC	SVACC	–	TXRDY	RXRDY	TXCOMP

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**

28.8.10 TWI Receive Holding Register

Name: TWI_RHR

Access: Read-only

Reset: 0x00000000

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

- **RXDATA: Master or Slave Receive Holding Data**

28.8.11 TWI Transmit Holding Register

Name: TWI_THR

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
TXDATA							

- **TXDATA: Master or Slave Transmit Holding Data**

28.8.12 TWI Write Protection Mode Register

Name: TWI_WPMR

Access: Read/Write

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

- **WPEN: 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 12. “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

28.8.13 TWI Write Protection Status Register

Name: TWI_WPSR

Access: Read-only

31	30	29	28	27	26	25	24
WPVSRC							
23	22	21	20	19	18	17	16
WPVSRC							
15	14	13	12	11	10	9	8
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 associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protection Violation Source**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

29. Universal Asynchronous Receiver Transmitter (UART)

29.1 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.

29.2 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

29.3 Block Diagram

Figure 29-1. UART Functional Block Diagram

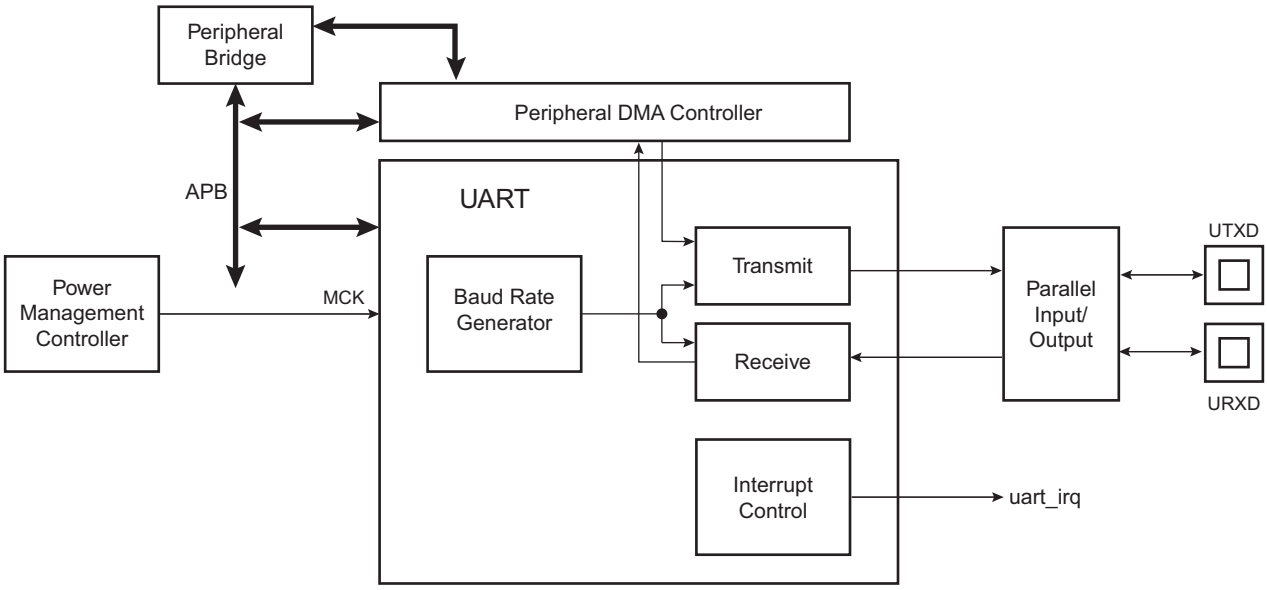


Table 29-1. UART Pin Description

Pin Name	Description	Type
URXD	UART Receive Data	Input
UTXD	UART Transmit Data	Output

29.4 Product Dependencies

29.4.1 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 29-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
UART0	URXD0	PA9	A
UART0	UTXD0	PA10	A
UART1	URXD1	PB2	A
UART1	UTXD1	PB3	A

29.4.2 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.

29.4.3 Interrupt Source

The UART interrupt line is connected to one of the interrupt sources of the Interrupt Controller. Interrupt handling requires programming of the Interrupt Controller before configuring the UART.

29.5 UART Operations

The UART operates in asynchronous mode only and supports only 8-bit character handling (with parity). It has no clock pin.

The UART is made up of a receiver and a transmitter that operate independently, and a common baud rate generator. Receiver timeout and transmitter time guard are not implemented. However, all the implemented features are compatible with those of a standard USART.

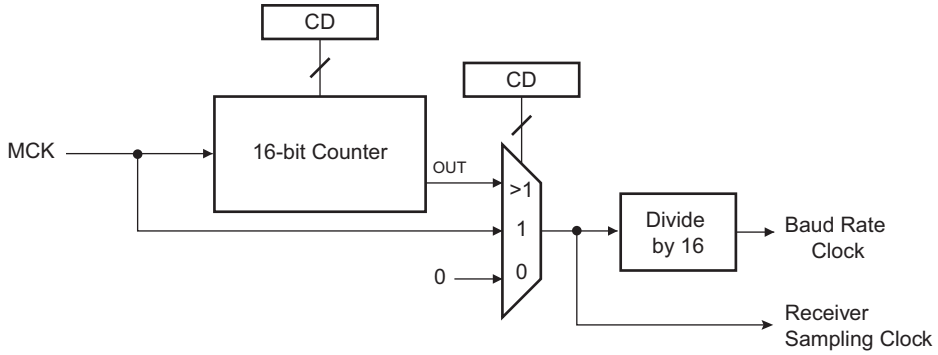
29.5.1 Baud Rate Generator

The baud rate generator provides the bit period clock named baud rate clock to both the receiver and the transmitter.

The baud rate clock is the master clock divided by 16 times the value (CD) written in UART_BRGR (Baud Rate Generator Register). If UART_BRGR is set to 0, the baud rate clock is disabled and the UART remains inactive. The maximum allowable baud rate is Master Clock divided by 16. The minimum allowable baud rate is Master Clock divided by (16 x 65536).

$$\text{Baud Rate} = \frac{\text{MCK}}{16 \times \text{CD}}$$

Figure 29-2. Baud Rate Generator



29.5.2 Receiver

29.5.2.1 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.

29.5.2.2 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 29-3. Start Bit Detection

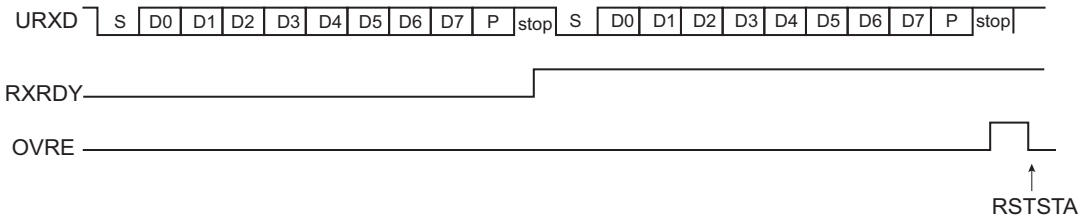
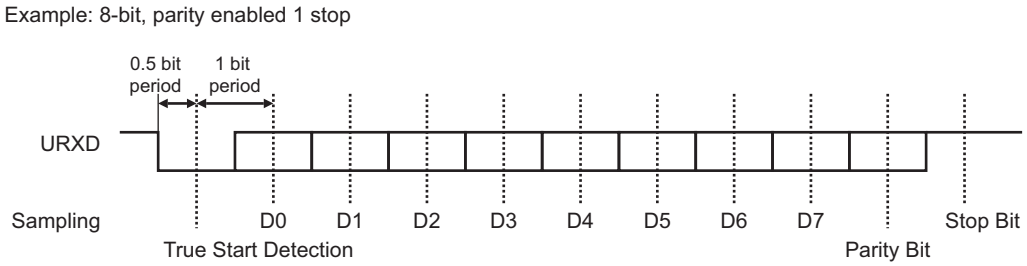


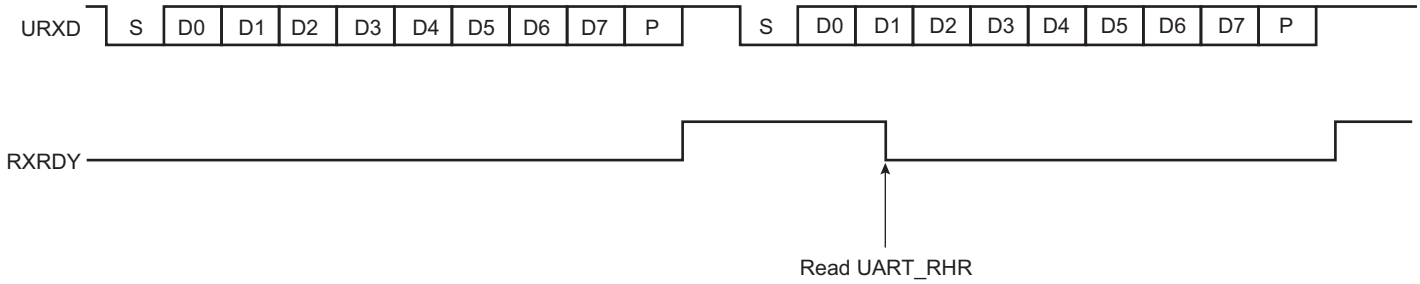
Figure 29-4. Character Reception



29.5.2.3 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.

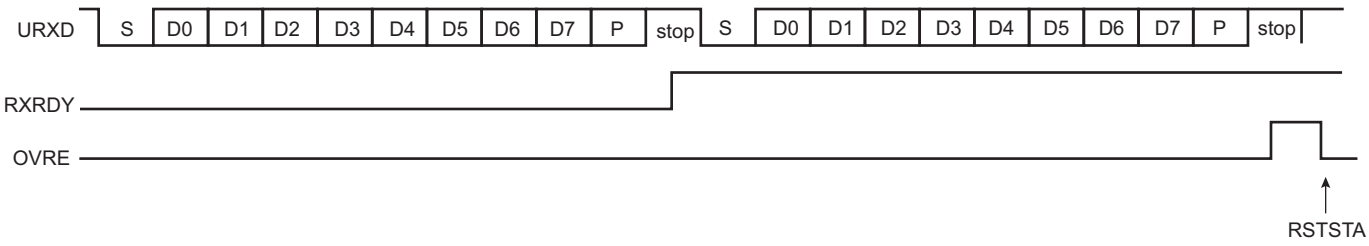
Figure 29-5. Receiver Ready



29.5.2.4 Receiver Overrun

The OVRE status bit in UART_SR is set if UART_RHR has not been read by the software (or the Peripheral Data Controller or DMA Controller) 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.

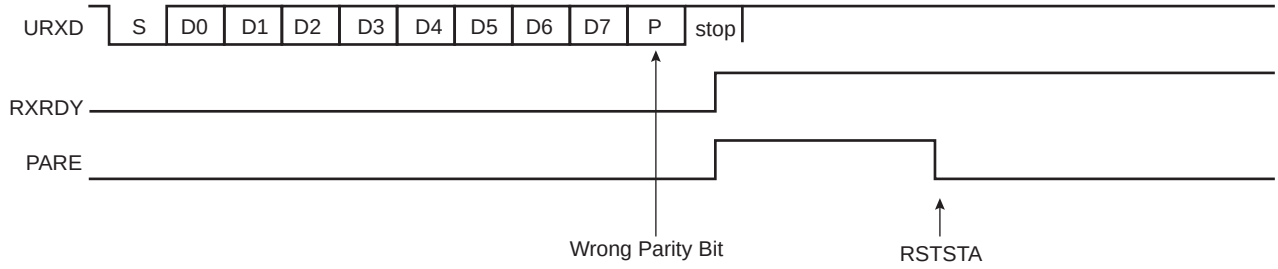
Figure 29-6. Receiver Overrun



29.5.2.5 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.

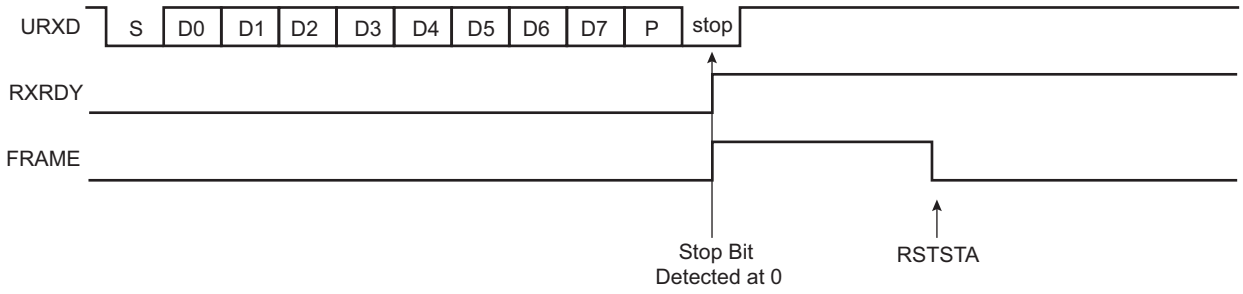
Figure 29-7. Parity Error



29.5.2.6 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 29-8. Receiver Framing Error



29.5.2.7 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.

29.5.3 Transmitter

29.5.3.1 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.

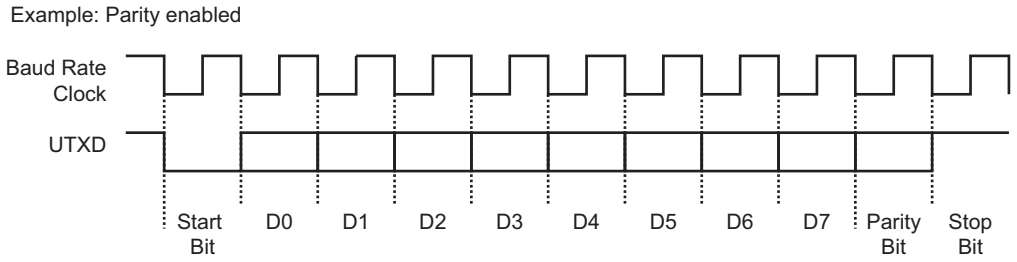
The programmer can disable the transmitter by writing UART_CR with the bit TXDIS at 1. If the transmitter is not operating, it is immediately stopped. However, if a character is being processed into the Shift Register and/or a character has been written in the Transmit Holding Register, the characters are completed before the transmitter is actually stopped.

The programmer can also put the transmitter in its reset state by writing the UART_CR with the bit RSTTX at 1. This immediately stops the transmitter, whether or not it is processing characters.

29.5.3.2 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 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 29-9. Character Transmission

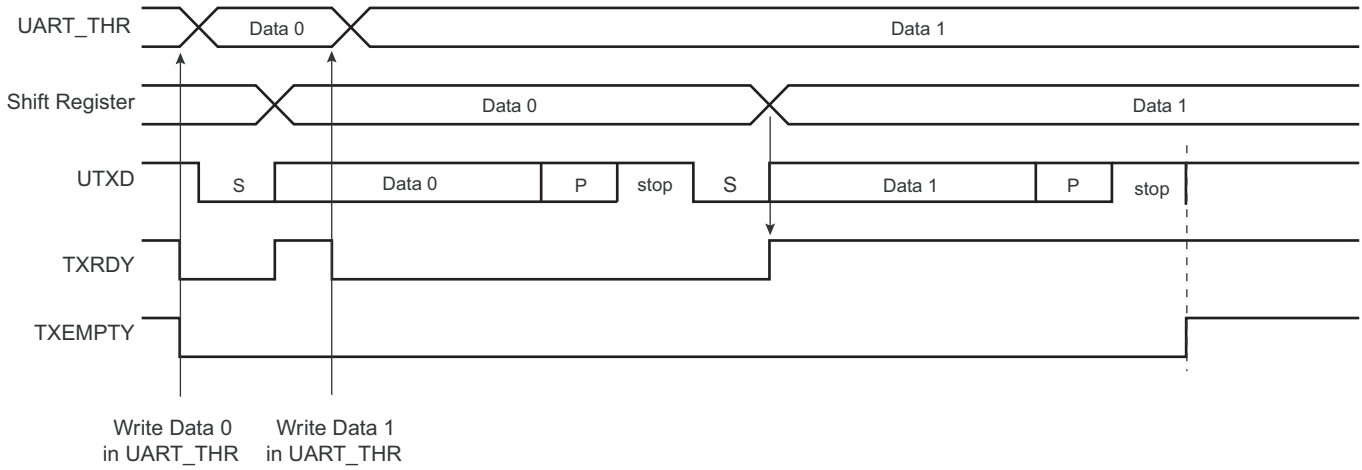


29.5.3.3 Transmitter Control

When the transmitter is enabled, the bit TXRDY (Transmitter Ready) is set in UART_SR. The transmission starts when the programmer writes in the Transmit Holding register (UART_THR), and after the written character is transferred from UART_THR to the Shift Register. The TXRDY bit remains high until a second character is written in UART_THR. As soon as the first character is completed, the last character written in UART_THR is transferred into the shift register and TXRDY rises again, showing that the holding register is empty.

When both the Shift Register and UART_THR are empty, i.e., all the characters written in UART_THR have been processed, the TXEMPTY bit rises after the last stop bit has been completed.

Figure 29-10. Transmitter Control



29.5.4 Peripheral DMA Controller (PDC)

Both the receiver and the transmitter of the UART are connected to a PDC.

The peripheral data controller channels are programmed via registers that are mapped within the UART user interface from the offset 0x100. The status bits are reported in 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.

29.5.5 Test Modes

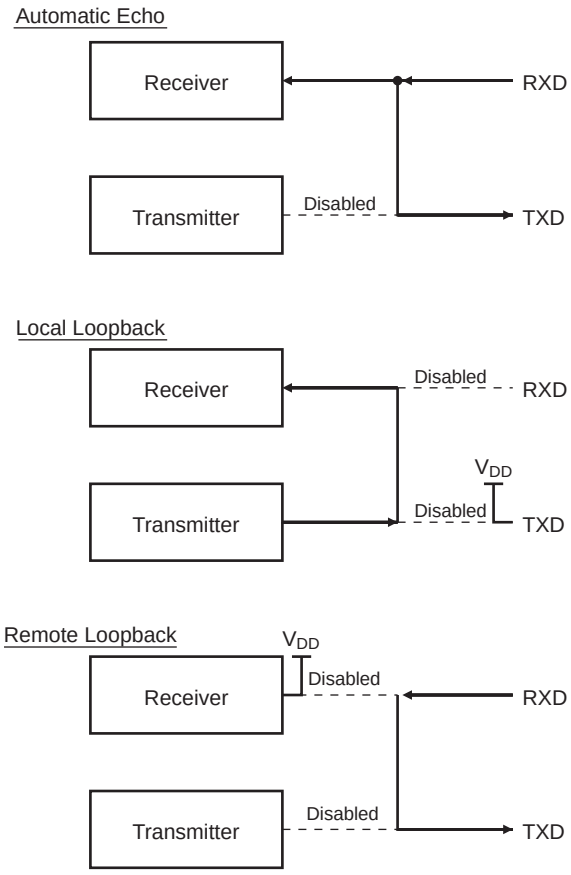
The UART supports three test modes. These modes of operation are programmed by using the CHMODE field in UART_MR.

The automatic echo mode allows 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 29-11. Test Modes



29.6 Universal Asynchronous Receiver Transmitter (UART) User Interface

Table 29-3. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	Control Register	UART_CR	Write-only	–
0x0004	Mode Register	UART_MR	Read/Write	0x0
0x0008	Interrupt Enable Register	UART_IER	Write-only	–
0x000C	Interrupt Disable Register	UART_IDR	Write-only	–
0x0010	Interrupt Mask Register	UART_IMR	Read-only	0x0
0x0014	Status Register	UART_SR	Read-only	–
0x0018	Receive Holding Register	UART_RHR	Read-only	0x0
0x001C	Transmit Holding Register	UART_THR	Write-only	–
0x0020	Baud Rate Generator Register	UART_BRGR	Read/Write	0x0
0x0024 - 0x003C	Reserved	–	–	–
0x0040 - 0x00E8	Reserved	–	–	–
0x00EC - 0x00FC	Reserved	–	–	–
0x0100 - 0x0128	Reserved for PDC registers	–	–	–

29.6.1 UART Control Register

Name: UART_CR

Address: 0x400E0600 (0), 0x400E0800 (1)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	RSTSTA
7	6	5	4	3	2	1	0
TXDIS	TXEN	RXDIS	RXEN	RSTTX	RSTRX	–	–

- **RSTRX: Reset Receiver**

0: No effect.

1: The receiver logic is reset and disabled. If a character is being received, the reception is aborted.

- **RSTTX: Reset Transmitter**

0: No effect.

1: The transmitter logic is reset and disabled. If a character is being transmitted, the transmission is aborted.

- **RXEN: Receiver Enable**

0: No effect.

1: The receiver is enabled if RXDIS is 0.

- **RXDIS: Receiver Disable**

0: No effect.

1: The receiver is disabled. If a character is being processed and RSTRX is not set, the character is completed before the receiver is stopped.

- **TXEN: Transmitter Enable**

0: No effect.

1: The transmitter is enabled if TXDIS is 0.

- **TXDIS: Transmitter Disable**

0: No effect.

1: The transmitter is disabled. If a character is being processed and a character has been written in the UART_THR and RSTTX is not set, both characters are completed before the transmitter is stopped.

- **RSTSTA: Reset Status Bits**

0: No effect.

1: Resets the status bits PARE, FRAME and OVRE in the UART_SR.

29.6.2 UART Mode Register

Name: UART_MR

Address: 0x400E0604 (0), 0x400E0804 (1)

Access: Read/Write

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

- **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

29.6.3 UART Interrupt Enable Register

Name: UART_IER

Address: 0x400E0608 (0), 0x400E0808 (1)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	–	TXEMPTY	–
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **RXRDY: Enable RXRDY Interrupt**
- **TXRDY: Enable TXRDY Interrupt**
- **ENDRX: Enable End of Receive Transfer Interrupt**
- **ENDTX: Enable End of Transmit Interrupt**
- **OVRE: Enable Overrun Error Interrupt**
- **FRAME: Enable Framing Error Interrupt**
- **PARE: Enable Parity Error Interrupt**
- **TXEMPTY: Enable TXEMPTY Interrupt**
- **TXBUFE: Enable Buffer Empty Interrupt**
- **RXBUFF: Enable Buffer Full Interrupt**

29.6.4 UART Interrupt Disable Register

Name: UART_IDR

Address: 0x400E060C (0), 0x400E080C (1)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	–	TXEMPTY	–
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Disables the corresponding interrupt.

- **RXRDY: Disable RXRDY Interrupt**
- **TXRDY: Disable TXRDY Interrupt**
- **ENDRX: Disable End of Receive Transfer Interrupt**
- **ENDTX: Disable End of Transmit Interrupt**
- **OVRE: Disable Overrun Error Interrupt**
- **FRAME: Disable Framing Error Interrupt**
- **PARE: Disable Parity Error Interrupt**
- **TXEMPTY: Disable TXEMPTY Interrupt**
- **TXBUFE: Disable Buffer Empty Interrupt**
- **RXBUFF: Disable Buffer Full Interrupt**

29.6.5 UART Interrupt Mask Register

Name: UART_IMR

Address: 0x400E0610 (0), 0x400E0810 (1)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	–	TXEMPTY	–
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is disabled.

1: The corresponding interrupt is enabled.

- **RXRDY: Mask RXRDY Interrupt**
- **TXRDY: Disable TXRDY Interrupt**
- **ENDRX: Mask End of Receive Transfer Interrupt**
- **ENDTX: Mask End of Transmit Interrupt**
- **OVRE: Mask Overrun Error Interrupt**
- **FRAME: Mask Framing Error Interrupt**
- **PARE: Mask Parity Error Interrupt**
- **TXEMPTY: Mask TXEMPTY Interrupt**
- **TXBUFE: Mask TXBUFE Interrupt**
- **RXBUFF: Mask RXBUFF Interrupt**

29.6.6 UART Status Register

Name: UART_SR

Address: 0x400E0614 (0), 0x400E0814 (1)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	–	TXEMPTY	–
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

- **RXRDY: Receiver Ready**

0: No character has been received since the last read of the UART_RHR, or the receiver is disabled.

1: At least one complete character has been received, transferred to UART_RHR and not yet read.

- **TXRDY: Transmitter Ready**

0: A character has been written to UART_THR and not yet transferred to the Shift Register, or the transmitter is disabled.

1: There is no character written to UART_THR not yet transferred to the Shift Register.

- **ENDRX: End of Receiver Transfer**

0: The end of transfer signal from the receiver Peripheral Data Controller channel is inactive.

1: The end of transfer signal from the receiver Peripheral Data Controller channel is active.

- **ENDTX: End of Transmitter Transfer**

0: The end of transfer signal from the transmitter Peripheral Data Controller channel is inactive.

1: The end of transfer signal from the transmitter Peripheral Data Controller channel is active.

- **OVRE: Overrun Error**

0: No overrun error has occurred since the last RSTSTA.

1: At least one overrun error has occurred since the last RSTSTA.

- **FRAME: Framing Error**

0: No framing error has occurred since the last RSTSTA.

1: At least one framing error has occurred since the last RSTSTA.

- **PARE: Parity Error**

0: No parity error has occurred since the last RSTSTA.

1: At least one parity error has occurred since the last RSTSTA.

- **TXEMPTY: Transmitter Empty**

0: There are characters in UART_THR, or characters being processed by the transmitter, or the transmitter is disabled.

1: There are no characters in UART_THR and there are no characters being processed by the transmitter.

- **TXBUFE: Transmission Buffer Empty**

0: The buffer empty signal from the transmitter PDC channel is inactive.

1: The buffer empty signal from the transmitter PDC channel is active.

- **RXBUFF: Receive Buffer Full**

0: The buffer full signal from the receiver PDC channel is inactive.

1: The buffer full signal from the receiver PDC channel is active.

29.6.7 UART Receiver Holding Register

Name: UART_RHR

Address: 0x400E0618 (0), 0x400E0818 (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.

29.6.8 UART Transmit Holding Register

Name: UART_THR

Address: 0x400E061C (0), 0x400E081C (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.

29.6.9 UART Baud Rate Generator Register

Name: UART_BRGR

Address: 0x400E0620 (0), 0x400E0820 (1)

Access: Read/Write

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

- **CD: Clock Divisor**

0: Baud Rate Clock is disabled

1 to 65,535: MCK / (CD x 16)

30. Universal Synchronous Asynchronous Receiver Transceiver (USART)

30.1 Description

The Universal Synchronous Asynchronous Receiver Transceiver (USART) provides one full duplex universal synchronous asynchronous serial link. Data frame format is widely programmable (data length, parity, number of stop bits) to support a maximum of standards. The receiver implements parity error, framing error and overrun error detection. The receiver time-out enables handling variable-length frames and the transmitter timeguard facilitates communications with slow remote devices. Multidrop communications are also supported through address bit handling in reception and transmission.

The USART features three test modes: remote loopback, local loopback and automatic echo.

The USART supports specific operating modes providing interfaces on RS485, and SPI buses, with ISO7816 T = 0 or T = 1 smart card slots 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.

30.2 Embedded Characteristics

- Programmable Baud Rate Generator
- 5- to 9-bit Full-duplex Synchronous or Asynchronous Serial Communications
  - 1, 1.5 or 2 Stop Bits in Asynchronous Mode or 1 or 2 Stop Bits in Synchronous Mode
  - Parity Generation and Error Detection
  - Framing Error Detection, Overrun Error Detection
  - MSB- or LSB-first
  - Optional Break Generation and Detection
  - By 8 or by 16 Over-sampling Receiver Frequency
  - Optional Hardware Handshaking RTS-CTS
  - Receiver Time-out and Transmitter Timeguard
  - Optional Multidrop Mode with Address Generation and Detection
- RS485 with Driver Control Signal
- ISO7816, T = 0 or T = 1 Protocols for Interfacing with Smart Cards
  - NACK Handling, Error Counter with Repetition and Iteration Limit
- IrDA Modulation and Demodulation
  - Communication at up to 115.2 Kbps
- SPI Mode
  - MASTER or Slave
  - Serial Clock Programmable Phase and Polarity
  - SPI Serial Clock (SCK) Frequency up to Internal Clock Frequency MCK/6
- Test Modes
  - Remote Loopback, Local Loopback, Automatic Echo
- Supports Connection of:
  - Two Peripheral DMA Controller Channels (PDC)
- Offers Buffer Transfer without Processor Intervention

30.3 Block Diagram

Figure 30-1. USART Block Diagram

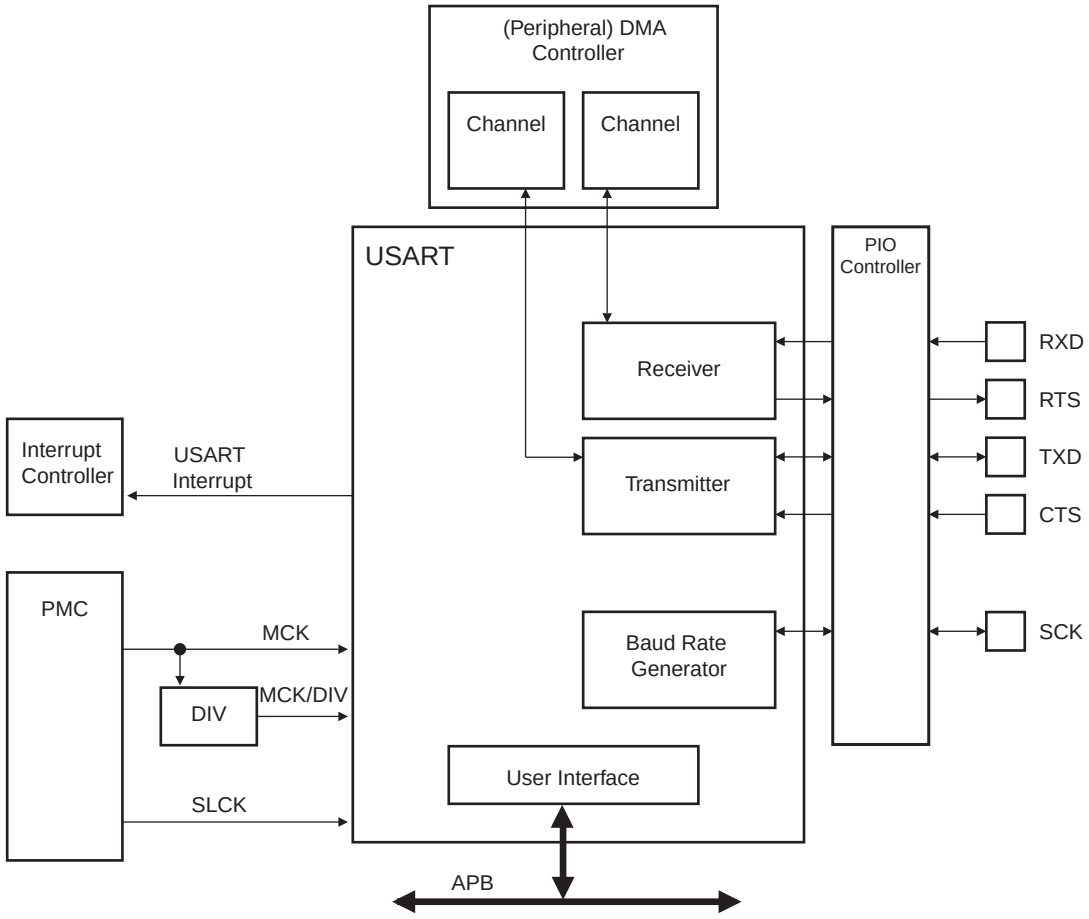
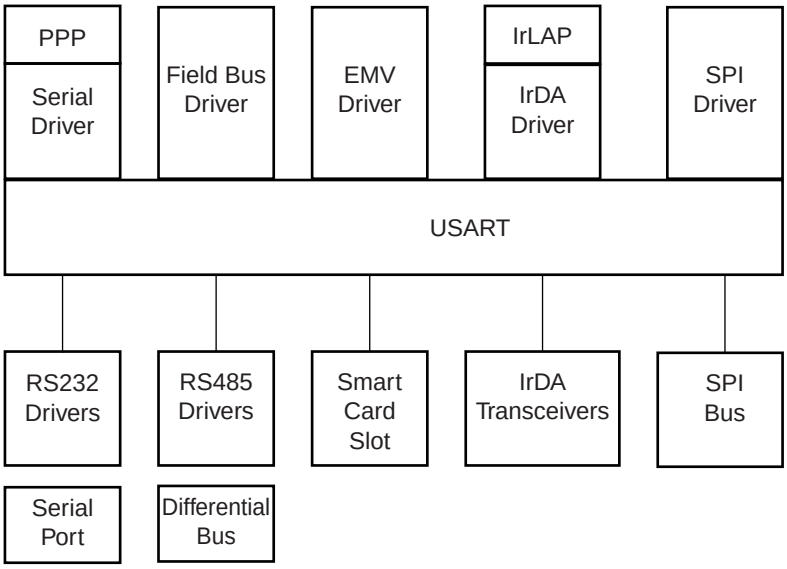


Table 30-1. SPI Operating Mode

Pin	USART	SPI Slave	SPI Master
RXD	RXD	MOSI	MISO
TXD	TXD	MISO	MOSI
RTS	RTS	–	CS
CTS	CTS	CS	–

30.4 Application Block Diagram

Figure 30-2. Application Block Diagram



30.5 I/O Lines Description

Table 30-2. I/O Line Description

Name	Description	Type	Active Level
SCK	Serial Clock	I/O	—
TXD	Transmit Serial Data or Master Out Slave In (MOSI) in SPI master mode or Master In Slave Out (MISO) in SPI slave mode	I/O	—
RXD	Receive Serial Data or Master In Slave Out (MISO) in SPI master mode or Master Out Slave In (MOSI) in SPI slave mode	I/O	—
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

30.6 Product Dependencies

30.6.1 I/O Lines

The pins used for interfacing the USART may be multiplexed with the PIO lines. The programmer must first program the PIO controller to assign the desired USART pins to their peripheral function. If I/O lines of the USART are not used by the application, they can be used for other purposes by the PIO Controller.

To prevent the TXD line from falling when the USART is disabled, the use of an internal pull up is mandatory. If the hardware handshaking feature is used, the internal pull up on TXD must also be enabled.

Table 30-3. I/O Lines

Instance	Signal	I/O Line	Peripheral
USART	CTS	PA16	A
USART	RTS	PA15	A
USART	RXD	PA5	A
USART	SCK	PA15	B
USART	TXD	PA6	A

30.6.2 Power Management

The USART is not continuously clocked. The programmer must first enable the USART Clock in the Power Management Controller (PMC) before using the USART. However, if the application does not require USART operations, the USART clock can be stopped when not needed and be restarted later. In this case, the USART will resume its operations where it left off.

Configuring the USART does not require the USART clock to be enabled.

30.6.3 Interrupt

The USART interrupt line is connected on one of the internal sources of the Interrupt Controller. Using the USART

Table 30-4. Peripheral IDs

Instance	ID
USART	14

interrupt requires the Interrupt Controller to be programmed first. Note that it is not recommended to use the USART interrupt line in edge sensitive mode.

30.7 Functional Description

30.7.1 Baud Rate Generator

The baud rate generator provides the bit period clock named the baud rate clock to both the receiver and the transmitter.

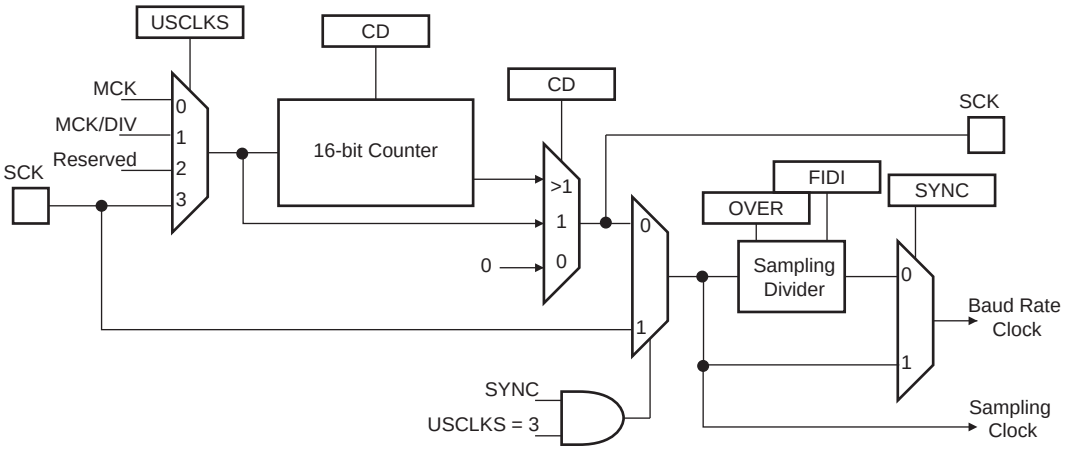
The baud rate generator clock source can be selected by setting the USCLKS field in US_MR between:

- The master clock MCK
- A division of the master clock, the divider being product dependent, but generally set to 8
- The external clock, available on the SCK pin

The baud rate generator is based upon a 16-bit divider, which is programmed with the CD field of the Baud Rate Generator register (US_BRGR). If a zero is written to CD, the baud rate generator does not generate any clock. If a one is written to CD, the divider is bypassed and becomes inactive.

If the external SCK clock is selected, the duration of the low and high levels of the signal provided on the SCK pin must be longer than a master clock (MCK) period. The frequency of the signal provided on SCK must be at least 3 times lower than MCK in USART mode, or 6 times lower in SPI mode.

Figure 30-3. Baud Rate Generator



30.7.1.1 Baud Rate in Asynchronous Mode

If the USART is programmed to operate in asynchronous mode, the selected clock is first divided by CD, which is field programmed in the US_BRGR. The resulting clock is provided to the receiver as a sampling clock and then divided by 16 or 8, depending on the programming of the OVER bit in the US_MR.

If OVER is set, the receiver sampling is eight times higher than the baud rate clock. If OVER is cleared, the sampling is performed at 16 times the baud rate clock.

The baud rate is calculated as per the following formula:

$$Baudrate = \frac{SelectedClock}{(8(2-Over)CD)}$$

This gives a maximum baud rate of MCK divided by 8, assuming that MCK is the highest possible clock and that the OVER bit is set.

Baud Rate Calculation Example

Table 30-5 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 30-5. Baud Rate Example (OVER = 0)

Source Clock (MHz)	Expected Baud Rate (Bit/s)	Calculation Result	CD	Actual Baud Rate (Bit/s)	Error
3,686,400	38,400	6.00	6	38,400.00	0.00%
4,915,200	38,400	8.00	8	38,400.00	0.00%
5,000,000	38,400	8.14	8	39,062.50	1.70%
7,372,800	38,400	12.00	12	38,400.00	0.00%
8,000,000	38,400	13.02	13	38,461.54	0.16%
12,000,000	38,400	19.53	20	37,500.00	2.40%
12,288,000	38,400	20.00	20	38,400.00	0.00%
14,318,180	38,400	23.30	23	38,908.10	1.31%
14,745,600	38,400	24.00	24	38,400.00	0.00%
18,432,000	38,400	30.00	30	38,400.00	0.00%
24,000,000	38,400	39.06	39	38,461.54	0.16%
24,576,000	38,400	40.00	40	38,400.00	0.00%
25,000,000	38,400	40.69	40	38,109.76	0.76%
32,000,000	38,400	52.08	52	38,461.54	0.16%
32,768,000	38,400	53.33	53	38,641.51	0.63%
33,000,000	38,400	53.71	54	38,194.44	0.54%
40,000,000	38,400	65.10	65	38,461.54	0.16%
50,000,000	38,400	81.38	81	38,580.25	0.47%

The baud rate is calculated with the following formula:

$$BaudRate = MCK/CD \times 16$$

The baud rate error is calculated with the following formula. It is not recommended to work with an error higher than 5%.

$$Error = 1 - \left(\frac{ExpectedBaudRate}{ActualBaudRate}\right)$$

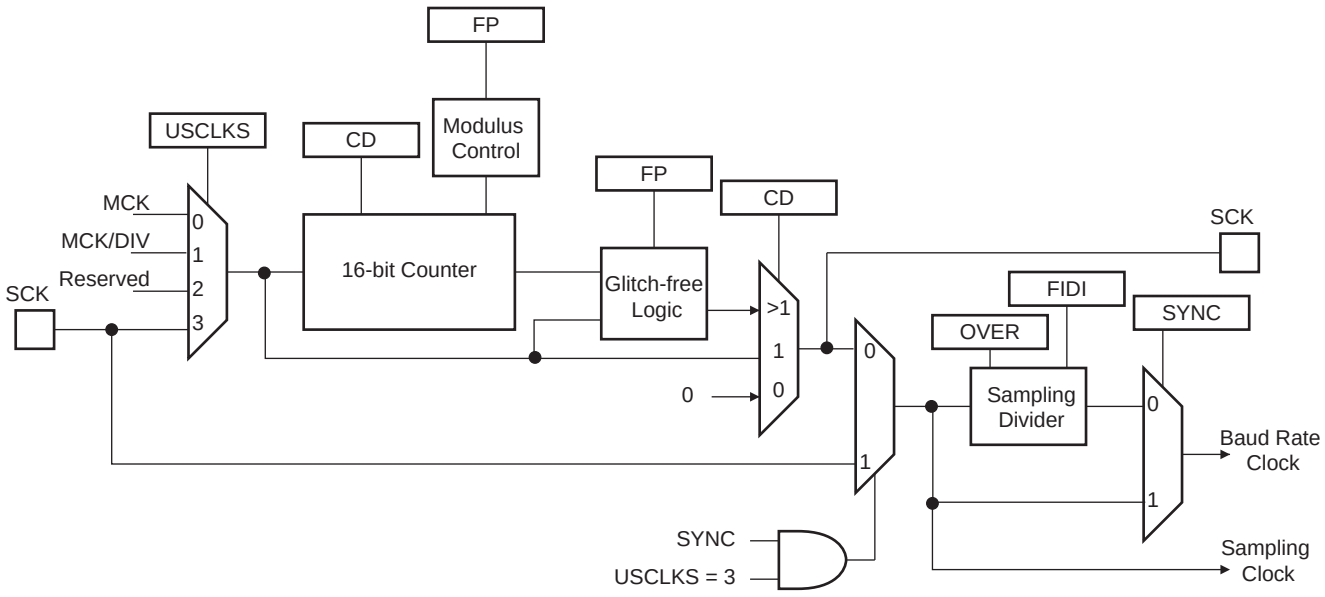
30.7.1.2 Fractional Baud Rate in Asynchronous Mode

The baud rate generator previously defined is subject to the following limitation: the output frequency changes by only integer multiples of the reference frequency. An approach to this problem is to integrate a fractional N clock generator that has a high resolution. The generator architecture is modified to obtain baud rate changes by a fraction of the reference source clock. This fractional part is programmed with the FP field in the US_BRGR. If FP is not 0, the fractional part is activated. The resolution is one eighth of the clock divider. This feature is only available when using USART normal mode. The fractional baud rate is calculated using the following formula:

$$Baudrate = \frac{SelectedClock}{\left(8(2 - Over)\left(CD + \frac{FP}{8}\right)\right)}$$

The modified architecture is presented below:

Figure 30-4. Fractional Baud Rate Generator



30.7.1.3 Baud Rate in Synchronous Mode or SPI Mode

If the USART is programmed to operate in synchronous mode, the selected clock is simply divided by the field CD in the US_BRGR.

$$BaudRate = \frac{SelectedClock}{CD}$$

In synchronous mode, if the external clock is selected (USCLKS = 3), the clock is provided directly by the signal on the USART SCK pin. No division is active. The value written in US_BRGR has no effect. The external clock frequency must be at least 3 times lower than the system clock. In synchronous mode master (USCLKS = 0 or 1, CLK0 set to 1), the receive part limits the SCK maximum frequency toMCK/3 in USART mode, or MCK/6 in SPI mode.

When either the external clock SCK or the internal clock divided (MCK/DIV) is selected, the value programmed in CD must be even if the user has to ensure a 50:50 mark/space ratio on the SCK pin. If the internal clock MCK is selected, the baud rate generator ensures a 50:50 duty cycle on the SCK pin, even if the value programmed in CD is odd.

30.7.1.4 Baud Rate in ISO 7816 Mode

The ISO7816 specification defines the bit rate with the following formula:

$$B = \frac{Di}{Fi} \times f$$

where:

- B is the bit rate
- Di is the bit-rate adjustment factor
- Fi is the clock frequency division factor
- f is the ISO7816 clock frequency (Hz)

Di is a binary value encoded on a 4-bit field, named DI, as represented in Table 30-6.

Table 30-6. Binary and Decimal Values for Di

DI field	0001	0010	0011	0100	0101	0110	1000	1001
Di (decimal)	1	2	4	8	16	32	12	20

Fi is a binary value encoded on a 4-bit field, named FI, as represented in Table 30-7.

Table 30-7. Binary and Decimal Values for Fi

FI field	0000	0001	0010	0011	0100	0101	0110	1001	1010	1011	1100	1101
Fi (decimal)	372	372	558	744	1116	1488	1860	512	768	1024	1536	2048

Table 30-8 shows the resulting Fi/Di Ratio, which is the ratio between the ISO7816 clock and the baud rate clock.

Table 30-8. Possible Values for the Fi/Di Ratio

Fi/Di	372	558	774	1116	1488	1806	512	768	1024	1536	2048
1	372	558	744	1116	1488	1860	512	768	1024	1536	2048
2	186	279	372	558	744	930	256	384	512	768	1024
4	93	139.5	186	279	372	465	128	192	256	384	512
8	46.5	69.75	93	139.5	186	232.5	64	96	128	192	256
16	23.25	34.87	46.5	69.75	93	116.2	32	48	64	96	128
32	11.62	17.43	23.25	34.87	46.5	58.13	16	24	32	48	64
12	31	46.5	62	93	124	155	42.66	64	85.33	128	170.6
20	18.6	27.9	37.2	55.8	74.4	93	25.6	38.4	51.2	76.8	102.4

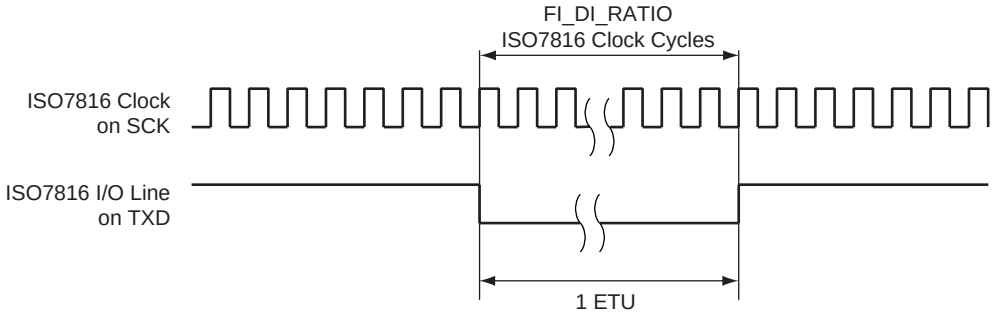
If the USART is configured in ISO7816 mode, the clock selected by the USCLKS field in the US_MR is first divided by the value programmed in the field CD in the US_BRGR. The resulting clock can be provided to the SCK pin to feed the smart card clock inputs. This means that the CLKO bit can be set in US_MR.

This clock is then divided by the value programmed in the FI_DI_RATIO field in the FI_DI_Ratio register (US_FIDI). This is performed by the Sampling Divider, which performs a division by up to 2047 in ISO7816 mode. The non-integer values of the Fi/Di Ratio are not supported and the user must program the FI_DI_RATIO field to a value as close as possible to the expected value.

The FI_DI_RATIO field resets to the value 0x174 (372 in decimal) and is the most common divider between the ISO7816 clock and the bit rate (Fi = 372, Di = 1).

Figure 30-5 shows the relation between the Elementary Time Unit, corresponding to a bit time, and the ISO 7816 clock.

Figure 30-5. Elementary Time Unit (ETU)



30.7.2 Receiver and Transmitter Control

After reset, the receiver is disabled. The user must enable the receiver by setting the RXEN bit in the Control register (US_CR). However, the receiver registers can be programmed before the receiver clock is enabled.

After reset, the transmitter is disabled. The user must enable it by setting the TXEN bit in the US_CR. However, the transmitter registers can be programmed before being enabled.

The receiver and the transmitter can be enabled together or independently.

At any time, the software can perform a reset on the receiver or the transmitter of the USART by setting the corresponding bit, RSTRX and RSTTX respectively, in the US_CR. The software resets clear the status flag and reset internal state machines but the user interface configuration registers hold the value configured prior to software reset. Regardless of what the receiver or the transmitter is performing, the communication is immediately stopped.

The user can also independently disable the receiver or the transmitter by setting RXDIS and TXDIS respectively in the US_CR. If the receiver is disabled during a character reception, the USART waits until the end of reception of the current character, then the reception is stopped. If the transmitter is disabled while it is operating, the USART waits the end of transmission of both the current character and character being stored in the Transmit Holding register (US_THR). If a timeguard is programmed, it is handled normally.

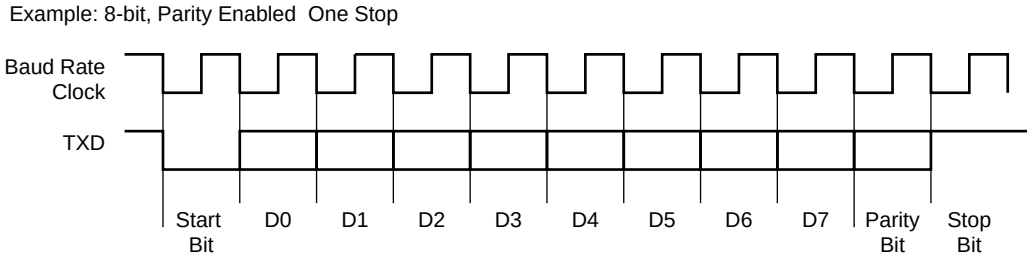
30.7.3 Synchronous and Asynchronous Modes

30.7.3.1 Transmitter Operations

The transmitter performs the same in both synchronous and asynchronous operating modes (SYNC = 0 or SYNC = 1). One start bit, up to 9 data bits, one optional parity bit and up to two stop bits are successively shifted out on the TXD pin at each falling edge of the programmed serial clock.

The number of data bits is selected by the CHRL field and the MODE 9 bit in US_MR. Nine bits are selected by setting the MODE 9 bit regardless of the CHRL field. The parity bit is set according to the PAR field in US_MR. The even, odd, space, marked or none parity bit can be configured. The MSBF field in the US_MR configures which data bit is sent first. If written to 1, the most significant bit is sent first. If written to 0, the less significant bit is sent first. The number of stop bits is selected by the NBSTOP field in the US_MR. The 1.5 stop bit is supported in asynchronous mode only.

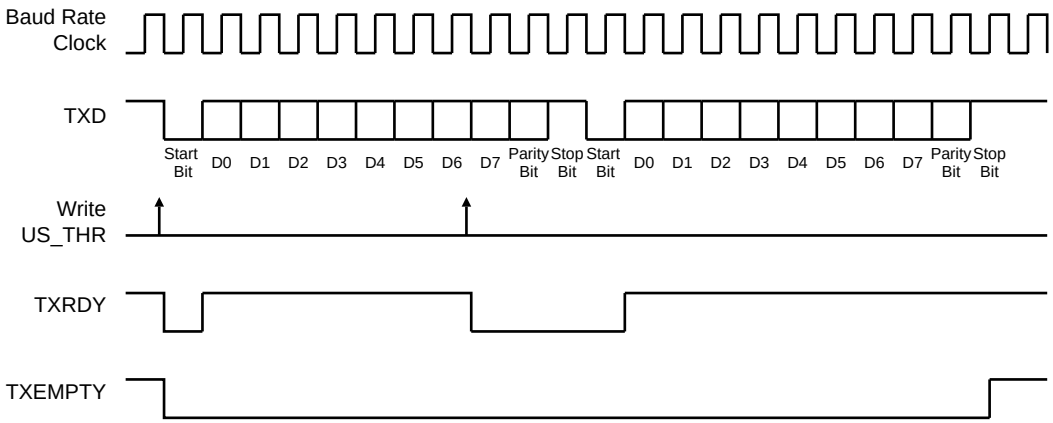
Figure 30-6. Character Transmit



The characters are sent by writing in the Transmit Holding register (US_THR). The transmitter reports two status bits in the Channel Status register (US_CSR): TXRDY (Transmitter Ready), which indicates that US_THR is empty and TXEMPTY, which indicates that all the characters written in US_THR have been processed. When the current character processing is completed, the last character written in US_THR is transferred into the Shift register of the transmitter and US_THR becomes empty, thus TXRDY rises.

Both TXRDY and TXEMPTY bits are low when the transmitter is disabled. Writing a character in US_THR while TXRDY is low has no effect and the written character is lost.

Figure 30-7. Transmitter Status



30.7.3.2 Asynchronous Receiver

If the USART is programmed in asynchronous operating mode (SYNC = 0), the receiver oversamples the RXD input line. The oversampling is either 16 or 8 times the baud rate clock, depending on the OVER bit in the US_MR.

The receiver samples the RXD line. If the line is sampled during one half of a bit time to 0, a start bit is detected and data, parity and stop bits are successively sampled on the bit rate clock.

If the oversampling is 16, (OVER to 0), a start is detected at the eighth sample to 0. Then, data bits, parity bit and stop bit are sampled on each 16 sampling clock cycle. If the oversampling is 8 (OVER to 1), a start bit is detected at the fourth sample to 0. Then, data bits, parity bit and stop bit are sampled on each 8 sampling clock cycle.

The number of data bits, first bit sent and parity mode are selected by the same fields and bits as the transmitter, i.e., respectively CHRL, MODE9, MSBF and PAR. For the synchronization mechanism **only**, the number of stop bits has no effect on the receiver as it considers only one stop bit, regardless of the field NBSTOP, so that resynchronization between the receiver and the transmitter can occur. Moreover, as soon as the stop bit is sampled, the receiver starts looking for a new start bit so that resynchronization can also be accomplished when the transmitter is operating with one stop bit.

Figure 30-8 and Figure 30-9 illustrate start detection and character reception when USART operates in asynchronous mode.

Figure 30-8. Asynchronous Start Detection

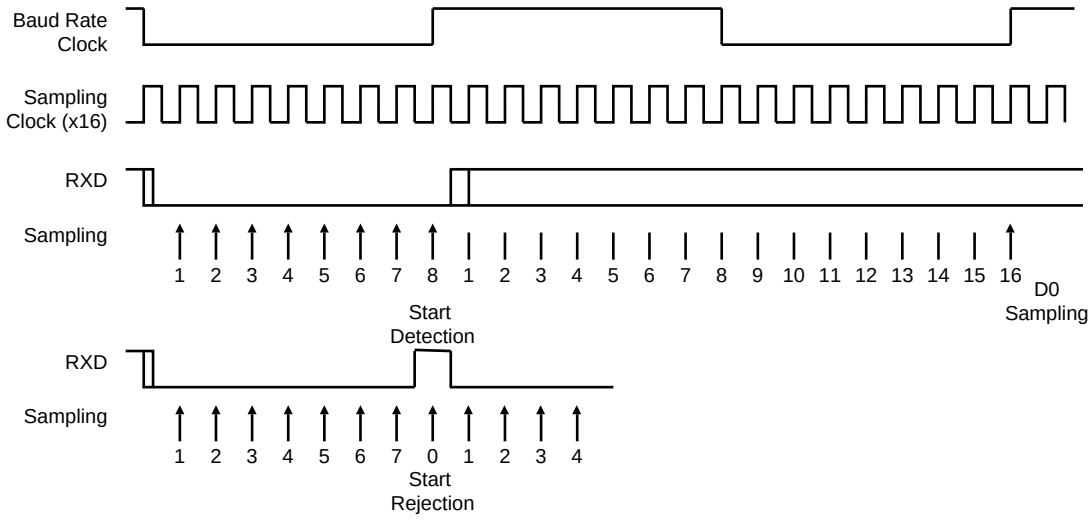
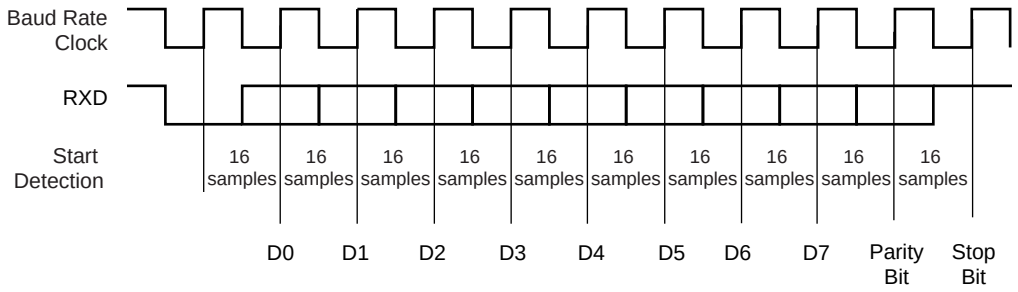


Figure 30-9. Asynchronous Character Reception

Example: 8-bit, Parity Enabled



30.7.3.3 Synchronous Receiver

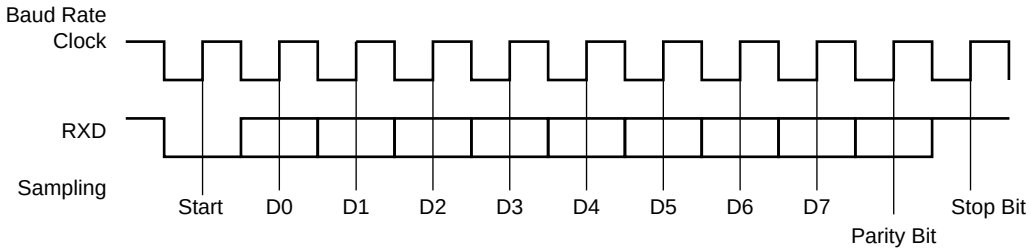
In synchronous mode (SYNC = 1), the receiver samples the RXD signal on each rising edge of the baud rate clock. If a low level is detected, it is considered as a start. All data bits, the parity bit and the stop bits are sampled and the receiver waits for the next start bit. Synchronous mode operations provide a high-speed transfer capability.

Configuration fields and bits are the same as in asynchronous mode.

Figure 30-10 illustrates a character reception in synchronous mode.

Figure 30-10. Synchronous Mode Character Reception

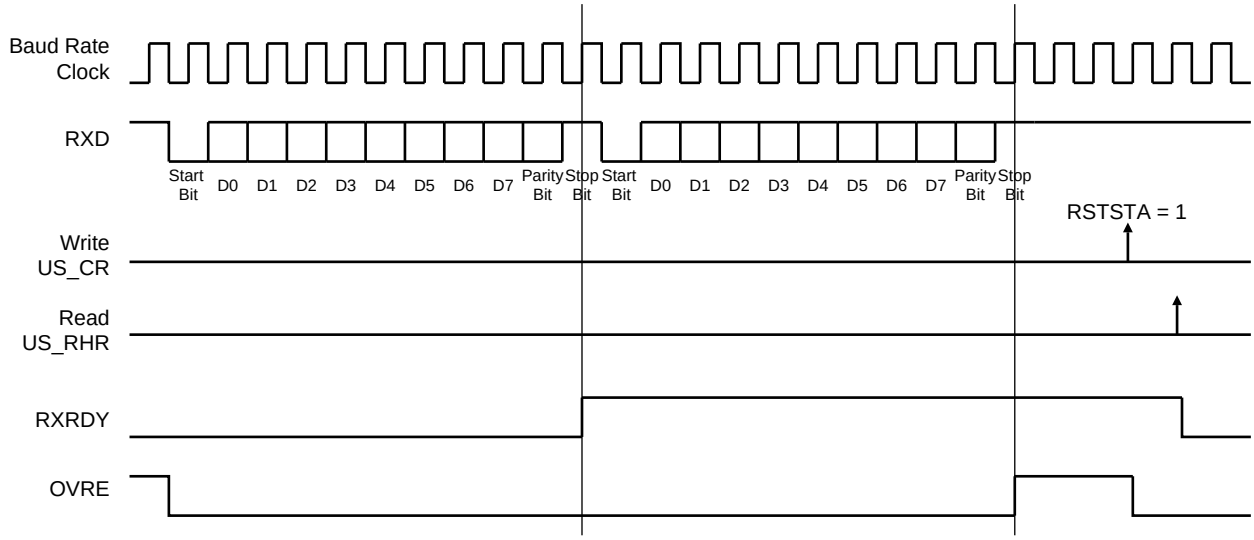
Example: 8-bit, Parity Enabled 1 Stop



30.7.3.4 Receiver Operations

When a character reception is completed, it is transferred to the Receive Holding register (US_RHR) and the RXRDY bit in US_CSR rises. If a character is completed while the RXRDY is set, the OVRE (Overrun Error) bit is set. The last character is transferred into US_RHR and overwrites the previous one. The OVRE bit is cleared by writing US_CR with the RSTSTA (Reset Status) bit to 1.

Figure 30-11. Receiver Status



30.7.3.5 Parity

The USART supports five parity modes that are selected by writing to the PAR field in the US_MR. The PAR field also enables the multidrop mode, see “Multidrop Mode” on page 695. Even and odd parity bit generation and error detection are supported.

If even parity is selected, the parity generator of the transmitter drives the parity bit to 0 if a number of 1s in the character data bit is even, and to 1 if the number of 1s is odd. Accordingly, the receiver parity checker counts the number of received 1s and reports a parity error if the sampled parity bit does not correspond. If odd parity is selected, the parity generator of the transmitter drives the parity bit to 1 if a number of 1s in the character data bit is even, and to 0 if the number of 1s is odd. Accordingly, the receiver parity checker counts the number of received 1s and reports a parity error if the sampled parity bit does not correspond. If the mark parity is used, the parity generator of the transmitter drives the parity bit to 1 for all characters. The receiver parity checker reports an error if the parity bit is sampled to 0. If the space parity is used, the parity generator of the transmitter drives the parity bit to 0 for all characters. The receiver parity checker reports an error if the parity bit is sampled to 1. If parity is disabled, the transmitter does not generate any parity bit and the receiver does not report any parity error.

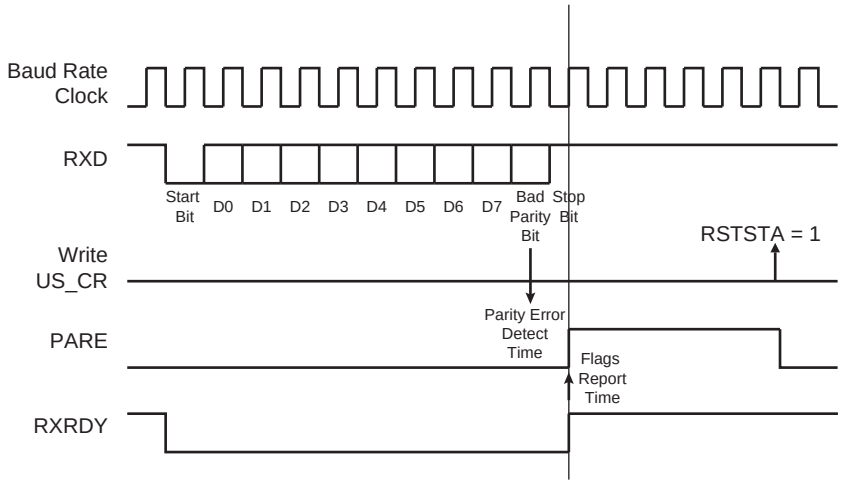
Table 30-9 shows an example of the parity bit for the character 0x41 (character ASCII “A”) depending on the configuration of the USART. Because there are two bits to 1, 1 bit is added when a parity is odd, or 0 is added when a parity is even.

Table 30-9. Parity Bit Examples

Character	Hexadecimal	Binary	Parity Bit	Parity Mode
A	0x41	0100 0001	1	Odd
A	0x41	0100 0001	0	Even
A	0x41	0100 0001	1	Mark
A	0x41	0100 0001	0	Space
A	0x41	0100 0001	None	None

When the receiver detects a parity error, it sets the PARE (Parity Error) bit in the US_CSR. The PARE bit can be cleared by writing the US_CR with the RSTSTA bit to 1. Figure 30-12 illustrates the parity bit status setting and clearing.

Figure 30-12. Parity Error



30.7.3.6 Multidrop Mode

If the value 0x6 or 0x07 is written to the PAR field in the US_MR, the USART runs in multidrop mode. This mode differentiates the data characters and the address characters. Data is transmitted with the parity bit to 0 and addresses are transmitted with the parity bit to 1.

If the USART is configured in multidrop mode, the receiver sets the PARE parity error bit when the parity bit is high and the transmitter is able to send a character with the parity bit high when a one is written to the SENTA bit in the US_CR.

To handle parity error, the PARE bit is cleared when a one is written to the RSTSTA bit in the US_CR.

The transmitter sends an address byte (parity bit set) when SENDA is written to in the US_CR. In this case, the next byte written to the US_THR is transmitted as an address. Any character written in the US_THR without having written the command SENDA is transmitted normally with the parity to 0.

30.7.3.7 Transmitter Timeguard

The timeguard feature enables the USART interface with slow remote devices.

The timeguard function enables the transmitter to insert an idle state on the TXD line between two characters. This idle state actually acts as a long stop bit.

The duration of the idle state is programmed in the TG field of the Transmitter Timeguard register (US_TTGR). When this field is written to zero no timeguard is generated. Otherwise, the transmitter holds a high level on TXD after each transmitted byte during the number of bit periods programmed in TG in addition to the number of stop bits.

As illustrated in Figure 30-13, the behavior of TXRDY and TXEMPTY status bits is modified by the programming of a timeguard. TXRDY rises only when the start bit of the next character is sent, and thus remains to 0 during the timeguard transmission if a character has been written in US_THR. TXEMPTY remains low until the timeguard transmission is completed as the timeguard is part of the current character being transmitted.

Figure 30-13. Timeguard Operations

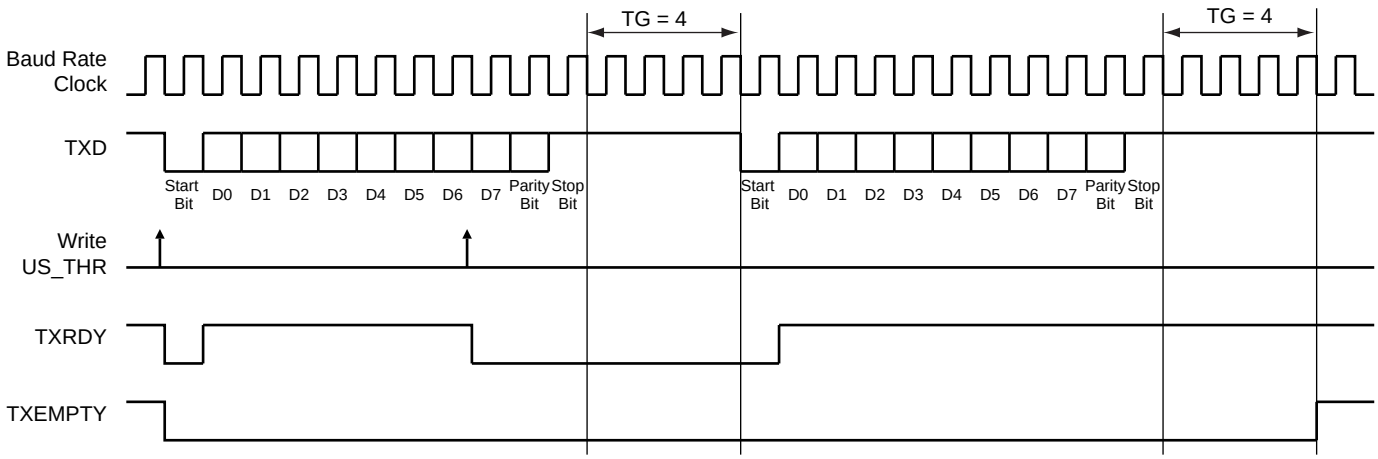


Table 30-10 indicates the maximum length of a timeguard period that the transmitter can handle in relation to the function of the baud rate.

Table 30-10. 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

30.7.3.8 Receiver Time-out

The Receiver Time-out provides support in handling variable-length frames. This feature detects an idle condition on the RXD line. When a time-out is detected, the bit TIMEOUT in the US_CSR rises and can generate an interrupt, thus indicating to the driver an end of frame.

The time-out delay period (during which the receiver waits for a new character) is programmed in the TO field of the Receiver Time-out register (US_RTOR). If the TO field is written to 0, the Receiver Time-out is disabled and no time-out is detected. The TIMEOUT bit in the US_CSR remains at 0. Otherwise, the receiver loads a 16-bit counter with the value programmed in TO. This counter is decremented at each bit period and reloaded each time a new character is received. If the counter reaches 0, the TIMEOUT bit in US_CSR rises. Then, the user can either:

- Stop the counter clock until a new character is received. This is performed by writing a one to the STTTO (Start Time-out) bit the US_CR. In this case, the idle state on RXD before a new character is received will not provide a time-out. This prevents having to handle an interrupt before a character is received and allows waiting for the next idle state on RXD after a frame is received.
- Obtain an interrupt while no character is received. This is performed by writing a one to the RETTO (Reload and Start Time-out) bit in the US_CR. If RETTO is performed, the counter starts counting down immediately from the value TO. This enables generation of a periodic interrupt so that a user time-out can be handled, for example when no key is pressed on a keyboard.

If STTTO is performed, the counter clock is stopped until a first character is received. The idle state on RXD before the start of the frame does not provide a time-out. This prevents having to obtain a periodic interrupt and enables a wait of the end of frame when the idle state on RXD is detected.

If RETTO is performed, the counter starts counting down immediately from the value TO. This enables generation of a periodic interrupt so that a user time-out can be handled, for example when no key is pressed on a keyboard.

Figure 30-14 shows the block diagram of the Receiver Time-out feature.

Figure 30-14. Receiver Time-out Block Diagram

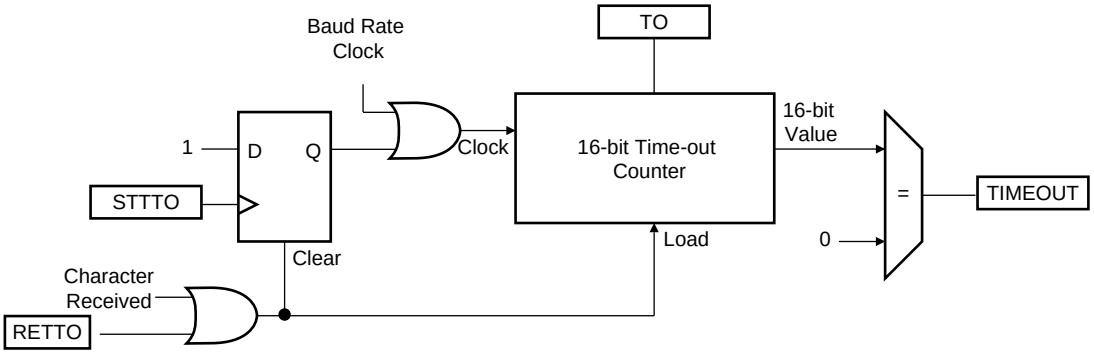


Table 30-11 gives the maximum time-out period for some standard baud rates.

Table 30-11. Maximum Time-out Period

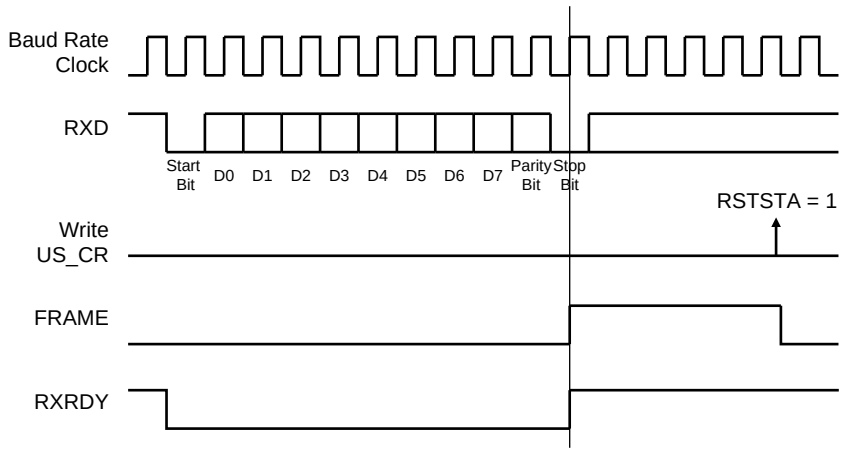
Baud Rate (Bit/s)	Bit Time (µs)	Time-out (ms)
600	1,667	109,225
1,200	833	54,613
2,400	417	27,306
4,800	208	13,653
9,600	104	6,827
14,400	69	4,551
19,200	52	3,413
28,800	35	2,276
38,400	26	1,704
56,000	18	1,170
57,600	17	1,138
200,000	5	328

30.7.3.9 Framing Error

The receiver is capable of detecting framing errors. A framing error happens when the stop bit of a received character is detected at level 0. This can occur if the receiver and the transmitter are fully desynchronized.

A framing error is reported on the FRAME bit of US_CSR. The FRAME bit is asserted in the middle of the stop bit as soon as the framing error is detected. It is cleared by writing US_CR with the RSTSTA bit to 1.

Figure 30-15. Framing Error Status



30.7.3.10 Transmit Break

The user can request the transmitter to generate a break condition on the TXD line. A break condition drives the TXD line low during at least one complete character. It appears the same as a 0x00 character sent with the parity and the stop bits to 0. However, the transmitter holds the TXD line at least during one character until the user requests the break condition to be removed.

A break is transmitted by writing US_CR with the STTBRK bit to 1. This can be performed at any time, either while the transmitter is empty (no character in either the Shift register or in US_THR) or when a character is being transmitted. If a break is requested while a character is being shifted out, the character is first completed before the TXD line is held low.

Once STTBRK command is requested further STTBRK commands are ignored until the end of the break is completed.

The break condition is removed by writing US_CR with the STPBRK bit to 1. If the STPBRK is requested before the end of the minimum break duration (one character, including start, data, parity and stop bits), the transmitter ensures that the break condition completes.

The transmitter considers the break as though it is a character, i.e., the STTBRK and STPBRK commands are taken into account only if the TXRDY bit in US_CSR is to 1 and the start of the break condition clears the TXRDY and TXEMPTY bits as if a character is processed.

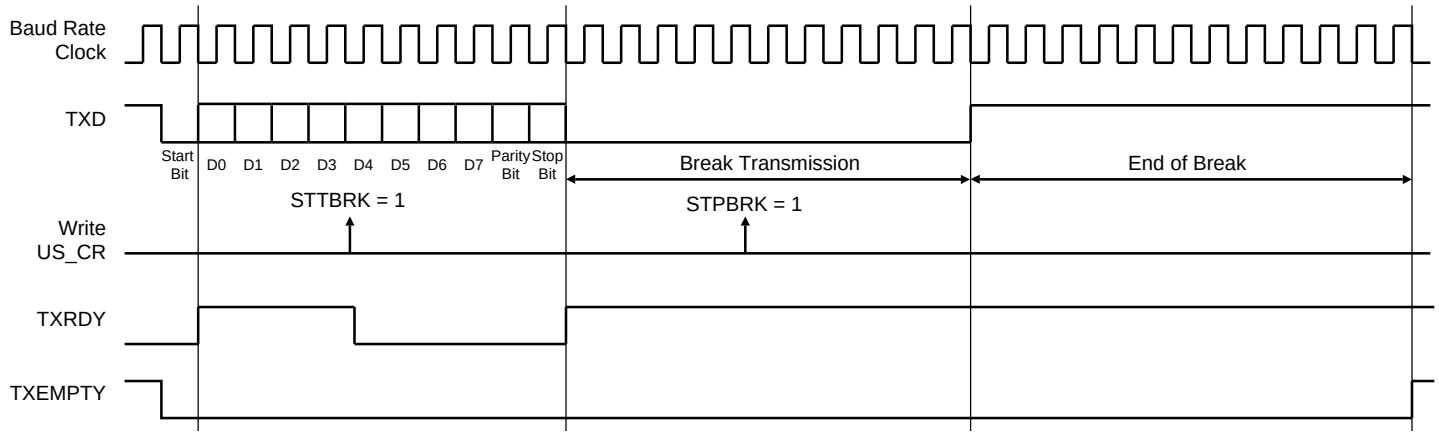
Writing US_CR with both STTBRK and STPBRK bits to 1 can lead to an unpredictable result. All STPBRK commands requested without a previous STTBRK command are ignored. A byte written into the Transmit Holding register while a break is pending, but not started, is ignored.

After the break condition, the transmitter returns the TXD line to 1 for a minimum of 12 bit times. Thus, the transmitter ensures that the remote receiver detects correctly the end of break and the start of the next character. If the timeguard is programmed with a value higher than 12, the TXD line is held high for the timeguard period.

After holding the TXD line for this period, the transmitter resumes normal operations.

Figure 30-16 illustrates the effect of both the Start Break (STTBRK) and Stop Break (STPBRK) commands on the TXD line.

Figure 30-16. Break Transmission



30.7.3.11 Receive Break

The receiver detects a break condition when all data, parity and stop bits are low. This corresponds to detecting a framing error with data to 0x00, but FRAME remains low.

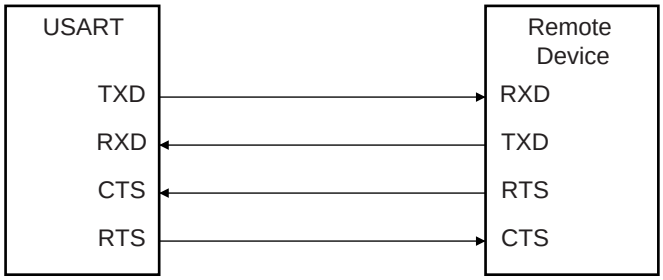
When the low stop bit is detected, the receiver asserts the RXBRK bit in US_CSR. This bit may be cleared by writing US_CR with the bit RSTSTA to 1.

An end of receive break is detected by a high level for at least 2/16 of a bit period in asynchronous operating mode or one sample at high level in synchronous operating mode. The end of break detection also asserts the RXBRK bit.

30.7.3.12 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 30-17.

Figure 30-17. Connection with a Remote Device for Hardware Handshaking



Setting the USART to operate with hardware handshaking is performed by writing the USART_MODE field in US_MR to the value 0x2.

The USART behavior when hardware handshaking is enabled is the same as the behavior in standard synchronous or asynchronous mode, except that the receiver drives the RTS pin as described below and the level on the CTS pin modifies the behavior of the transmitter as described below. Using this mode requires using the PDC channel for reception. The transmitter can handle hardware handshaking in any case.

Figure 30-18 shows how the receiver operates if hardware handshaking is enabled. The RTS pin is driven high if the receiver is disabled and if the status RXBUFF (Receive Buffer Full) coming from the PDC channel is high. Normally, the remote device does not start transmitting while its CTS pin (driven by RTS) is high. As soon as the receiver is enabled, the RTS falls, indicating to the remote device that it can start transmitting. Defining a new buffer to the PDC clears the status bit RXBUFF and, as a result, asserts the pin RTS low.

Figure 30-18. Receiver Behavior when Operating with Hardware Handshaking

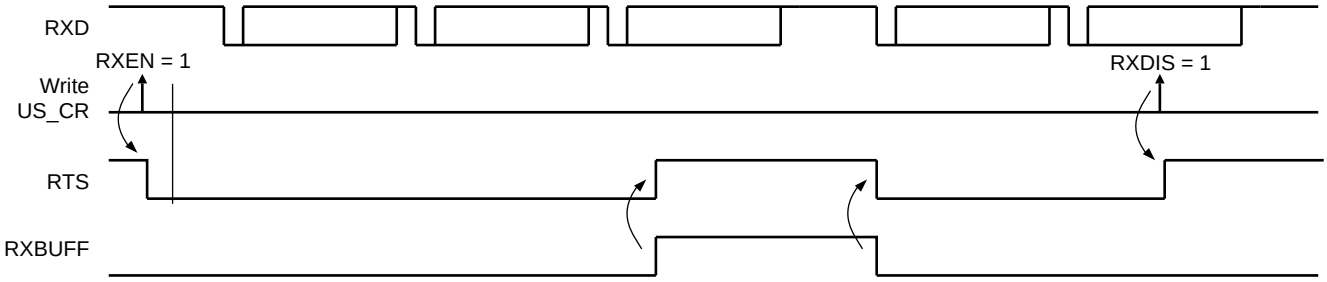
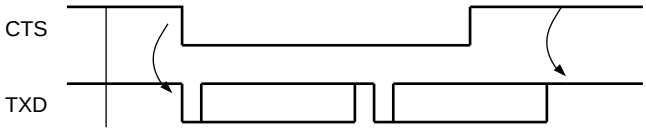


Figure 30-19 shows how the transmitter operates if hardware handshaking is enabled. The CTS pin disables the transmitter. If a character is being processing, the transmitter is disabled only after the completion of the current character and transmission of the next character happens as soon as the pin CTS falls.

Figure 30-19. Transmitter Behavior when Operating with Hardware Handshaking



30.7.4 ISO7816 Mode

The USART features an ISO7816-compatible operating mode. This mode permits interfacing with smart cards and Security Access Modules (SAM) communicating through an ISO7816 link. Both T = 0 and T = 1 protocols defined by the ISO7816 specification are supported.

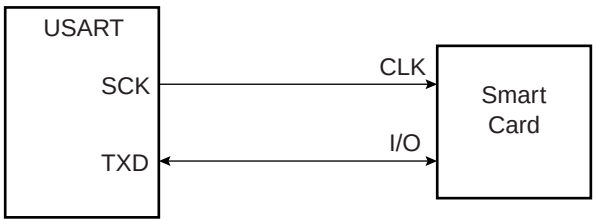
Setting the USART in ISO7816 mode is performed by writing the USART_MODE field in US_MR to the value 0x4 for protocol T = 0 and to the value 0x5 for protocol T = 1.

30.7.4.1 ISO7816 Mode Overview

The ISO7816 is a half duplex communication on only one bidirectional line. The baud rate is determined by a division of the clock provided to the remote device (see “Baud Rate Generator” on page 687).

The USART connects to a smart card as shown in Figure 30-20. 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 30-20. Connection of a Smart Card to the USART



When operating in ISO7816, either in T = 0 or T = 1 modes, the character format is fixed. The configuration is 8 data bits, even parity and 1 or 2 stop bits, regardless of the values programmed in the CHRL, MODE9, PAR and CHMODE fields. MSBF can be used to transmit LSB or MSB first. Parity Bit (PAR) can be used to transmit in normal or inverse mode. Refer to “USART Mode Register” on page 718 and “PAR: Parity Type” on page 719.

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.

30.7.4.2 Protocol T = 0

In T = 0 protocol, a character is made up of one start bit, eight data bits, one parity bit and one guard time, which lasts two bit times. The transmitter shifts out the bits and does not drive the I/O line during the guard time.

If no parity error is detected, the I/O line remains to 1 during the guard time and the transmitter can continue with the transmission of the next character, as shown in Figure 30-21.

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 30-22. This error bit is also named NACK, for Non Acknowledge. In this case, the character lasts 1 bit time more, as the guard time length is the same and is added to the error bit time which lasts 1 bit time.

When the USART is the receiver and it detects an error, it does not load the erroneous character in the Receive Holding register (US_RHR). It appropriately sets the PARE bit in the Status register (US_SR) so that the software can handle the error.

Figure 30-21. T = 0 Protocol without Parity Error

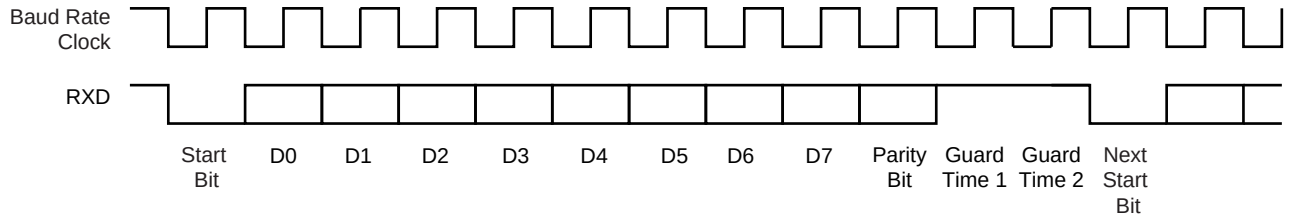
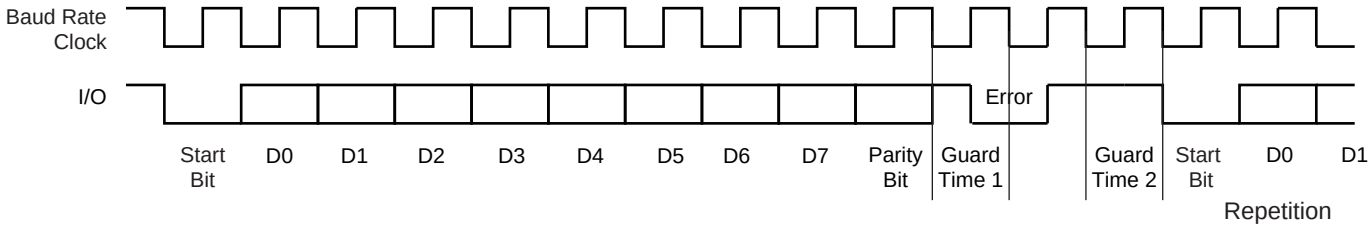


Figure 30-22. T = 0 Protocol with Parity Error



Receive Error Counter

The USART receiver also records the total number of errors. This can be read in the Number of Error (US_NER) register. The NB_ERRORS field can record up to 255 errors. Reading US_NER automatically clears the NB_ERRORS field.

Receive NACK Inhibit

The USART can also be configured to inhibit an error. This can be achieved by setting the INACK bit in US_MR. If INACK is to 1, no error signal is driven on the I/O line even if a parity bit is detected.

Moreover, if INACK is set, the erroneous received character is stored in the Receive Holding register, as if no error occurred and the RXRDY bit does rise.

Transmit Character Repetition

When the USART is transmitting a character and gets a NACK, it can automatically repeat the character before moving on to the next one. Repetition is enabled by writing the MAX_ITERATION field in the US_MR at a value higher than 0. Each character can be transmitted up to eight times; the first transmission plus seven repetitions.

If MAX_ITERATION does not equal zero, the USART repeats the character as many times as the value loaded in MAX_ITERATION.

When the USART repetition number reaches MAX_ITERATION, the ITERATION bit is set in US_CSR. If the repetition of the character is acknowledged by the receiver, the repetitions are stopped and the iteration counter is cleared.

The ITERATION bit in US_CSR can be cleared by writing US_CR with the RSTIT bit to 1.

Disable Successive Receive NACK

The receiver can limit the number of successive NACKs sent back to the remote transmitter. This is programmed by setting the bit DSNACK in the US_MR. The maximum number of NACKs transmitted is programmed in the MAX_ITERATION field. As soon as MAX_ITERATION is reached, the character is considered as correct, an acknowledge is sent on the line and the ITERATION bit in the US_CSR is set.

30.7.4.3 Protocol T = 1

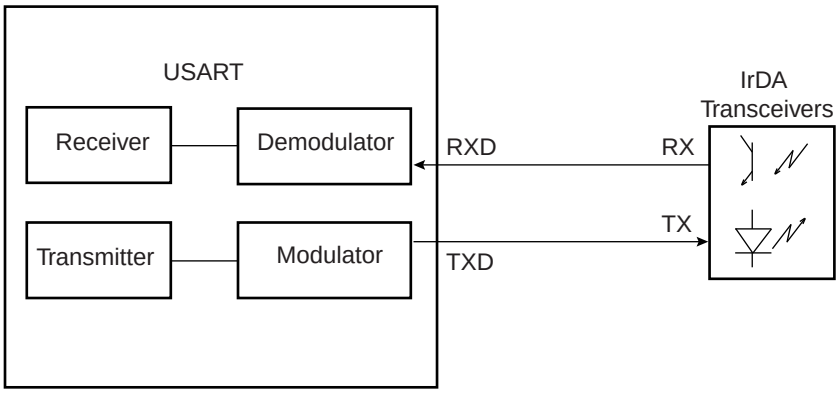
When operating in ISO7816 protocol T = 1, the transmission is similar to an asynchronous format with only one stop bit. The parity is generated when transmitting and checked when receiving. Parity error detection sets the PARE bit in the US_CSR.

30.7.5 IrDA Mode

The USART features an IrDA mode supplying half-duplex point-to-point wireless communication. It embeds the modulator and demodulator which allows a glueless connection to the infrared transceivers, as shown in Figure 30-23. The modulator and demodulator are compliant with the IrDA specification version 1.1 and support data transfer speeds ranging from 2.4 Kb/s to 115.2 Kb/s.

The USART IrDA mode is enabled by setting the USART_MODE field in US_MR to the value 0x8. The IrDA Filter register (US_IF) allows configuring the demodulator filter. The USART transmitter and receiver operate in a normal asynchronous mode and all parameters are accessible. Note that the modulator and the demodulator are activated.

Figure 30-23. Connection to IrDA Transceivers



The receiver and the transmitter must be enabled or disabled according to the direction of the transmission to be managed.

To receive IrDA signals, the following needs to be done:

- Disable TX and Enable RX
- Configure the TXD pin as PIO and set it as an output to 0 (to avoid LED emission). Disable the internal pull-up (better for power consumption).
- Receive data

30.7.5.1 IrDA Modulation

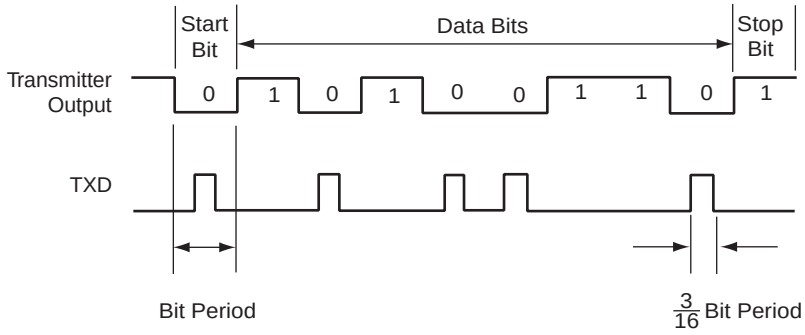
For baud rates up to and including 115.2 Kb/s, the RZI modulation scheme is used. “0” is represented by a light pulse of 3/16th of a bit time. Some examples of signal pulse duration are shown in Table 30-12.

Table 30-12. IrDA Pulse Duration

Baud Rate	Pulse Duration (3/16)
2.4 Kb/s	78.13 µs
9.6 Kb/s	19.53 µs
19.2 Kb/s	9.77 µs
38.4 Kb/s	4.88 µs
57.6 Kb/s	3.26 µs
115.2 Kb/s	1.63 µs

Figure 30-24 shows an example of character transmission.

Figure 30-24. IrDA Modulation



30.7.5.2 IrDA Baud Rate

Table 30-13 gives some examples of CD values, baud rate error and pulse duration. Note that the requirement on the maximum acceptable error of ±1.87% must be met.

Table 30-13. IrDA Baud Rate Error

Peripheral Clock	Baud Rate (Bit/s)	CD	Baud Rate Error	Pulse Time (µs)
3,686,400	115,200	2	0.00%	1.63
20,000,000	115,200	11	1.38%	1.63
32,768,000	115,200	18	1.25%	1.63
40,000,000	115,200	22	1.38%	1.63
3,686,400	57,600	4	0.00%	3.26
20,000,000	57,600	22	1.38%	3.26
32,768,000	57,600	36	1.25%	3.26
40,000,000	57,600	43	0.93%	3.26
3,686,400	38,400	6	0.00%	4.88
20,000,000	38,400	33	1.38%	4.88
32,768,000	38,400	53	0.63%	4.88
40,000,000	38,400	65	0.16%	4.88
3,686,400	19,200	12	0.00%	9.77
20,000,000	19,200	65	0.16%	9.77

Table 30-13. IrDA Baud Rate Error (Continued)

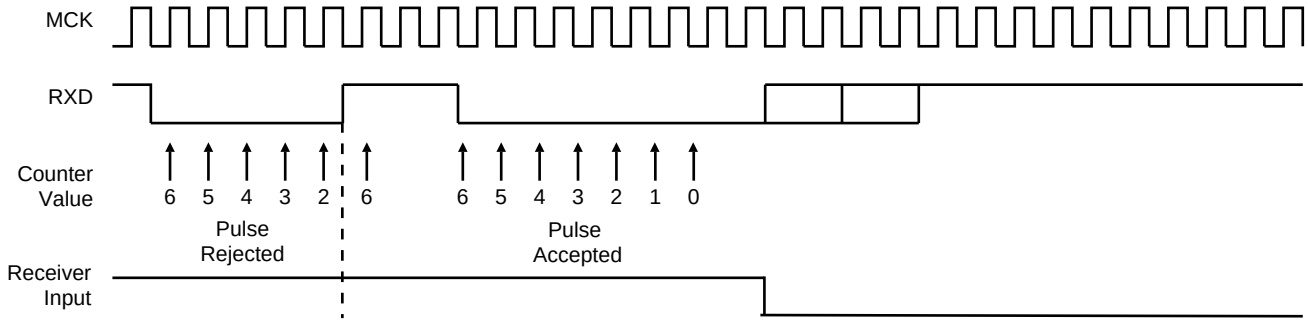
Peripheral Clock	Baud Rate (Bit/s)	CD	Baud Rate Error	Pulse Time (µs)
32,768,000	19,200	107	0.31%	9.77
40,000,000	19,200	130	0.16%	9.77
3,686,400	9,600	24	0.00%	19.53
20,000,000	9,600	130	0.16%	19.53
32,768,000	9,600	213	0.16%	19.53
40,000,000	9,600	260	0.16%	19.53
3,686,400	2,400	96	0.00%	78.13
20,000,000	2,400	521	0.03%	78.13
32,768,000	2,400	853	0.04%	78.13

30.7.5.3 IrDA Demodulator

The demodulator is based on the IrDA Receive filter comprised of an 8-bit down counter which is loaded with the value programmed in US_IF. When a falling edge is detected on the RXD pin, the Filter Counter starts counting down at the master clock (MCK) speed. If a rising edge is detected on the RXD pin, the counter stops and is reloaded with US_IF. If no rising edge is detected when the counter reaches 0, the input of the receiver is driven low during one bit time.

Figure 30-25 illustrates the operations of the IrDA demodulator.

Figure 30-25. IrDA Demodulator Operations



The programmed value in the US_IF register must always meet the following criteria:

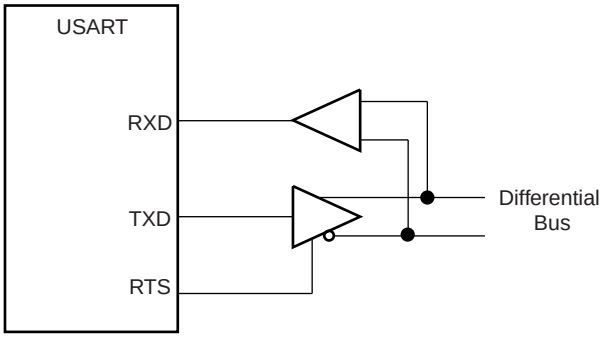
t_{MCK} * (IRDA_FILTER + 3) < 1.41 µs

As the IrDA mode uses the same logic as the ISO7816, note that the FI_DI_RATIO field in US_FIDI must be set to a value higher than 0 in order to assure IrDA communications operate correctly.

30.7.6 RS485 Mode

The USART features the RS485 mode to enable line driver control. While operating in RS485 mode, the USART behaves as though in asynchronous or synchronous mode and configuration of all the parameters is possible. The difference is that the RTS pin is driven high when the transmitter is operating. The behavior of the RTS pin is controlled by the TXEMPTY bit. A typical connection of the USART to an RS485 bus is shown in Figure 30-26.

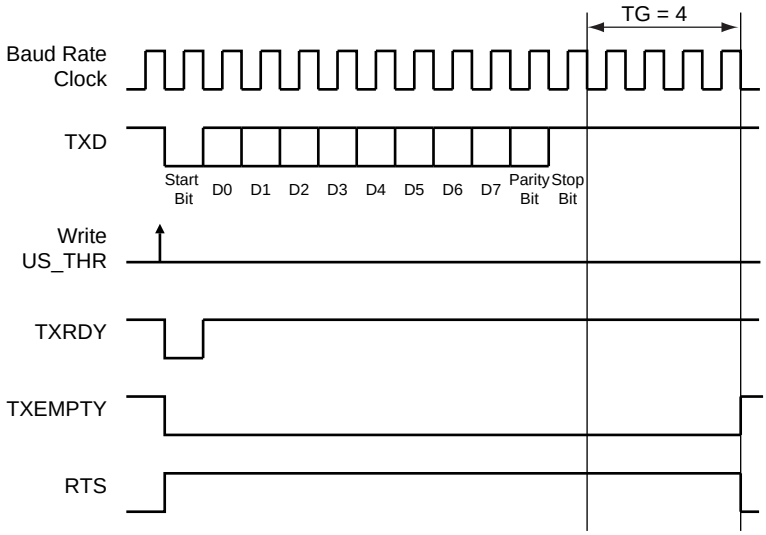
Figure 30-26. Typical Connection to a RS485 Bus



The USART is set in RS485 mode by writing the value 0x1 to the USART_MODE field in US_MR.

The RTS pin is at a level inverse to the TXEMPTY bit. Significantly, the RTS pin remains high when a timeguard is programmed so that the line can remain driven after the last character completion. Figure 30-27 gives an example of the RTS waveform during a character transmission when the timeguard is enabled.

Figure 30-27. Example of RTS Drive with Timeguard



30.7.7 SPI Mode

The Serial Peripheral Interface (SPI) mode is a synchronous serial data link that provides communication with external devices in master or slave mode. It also enables communication between processors if an external processor is connected to the system.

The Serial Peripheral Interface is essentially a shift register that serially transmits data bits to other SPIs. During a data transfer, one SPI system acts as the “master” which controls the data flow, while the other devices act as “slaves" which have data shifted into and out by the master. Different CPUs can take turns being masters and one master may simultaneously shift data into multiple slaves. (Multiple master protocol is the opposite of single master protocol, where one CPU is always the master while all of the others are always slaves.) However, only one slave may drive its output to write data back to the master at any given time.

A slave device is selected when its NSS signal is asserted by the master. The USART in SPI master mode can address only one SPI slave because it can generate only one NSS signal.

The SPI system consists of two data lines and two control lines:

- master Out Slave In (MOSI): This data line supplies the output data from the master shifted into the input of the slave.
- master In Slave Out (MISO): This data line supplies the output data from a slave to the input of the master.
- Serial Clock (SCK): This control line is driven by the master and regulates the flow of the data bits. The master may transmit data at a variety of baud rates. The SCK line cycles once for each bit that is transmitted.
- Slave Select (NSS): This control line allows the master to select or deselect the slave.

30.7.7.1 Modes of Operation

The USART can operate in SPI master mode or in SPI slave mode.

Operation in SPI master mode is programmed by writing 0xE to the USART_MODE field in US_MR. In this case the SPI lines must be connected as described below:

- The MOSI line is driven by the output pin TXD
- The MISO line drives the input pin RXD
- The SCK line is driven by the output pin SCK
- The NSS line is driven by the output pin RTS

Operation in SPI slave mode is programmed by writing to 0xF the USART_MODE field in US_MR. In this case the SPI lines must be connected as described below:

- The MOSI line drives the input pin RXD
- The MISO line is driven by the output pin TXD
- The SCK line drives the input pin SCK
- The NSS line drives the input pin CTS

In order to avoid unpredicted behavior, any change of the SPI mode must be followed by a software reset of the transmitter and of the receiver (except the initial configuration after a hardware reset). (See Section 30.7.7.4).

30.7.7.2 Baud Rate

In SPI mode, the baud rate generator operates in the same way as in USART synchronous mode: See “Baud Rate in Synchronous Mode or SPI Mode” on page 689. However, there are some restrictions:

In SPI master mode:

- The external clock SCK must not be selected (USCLKS ≠ 0x3), and the bit CLKO must be set to ‘1’ in the US_MR, in order to generate correctly the serial clock on the SCK pin.
- To obtain correct behavior of the receiver and the transmitter, the value programmed in CD must be superior or equal to 6.
- If the internal clock divided (MCK/DIV) is selected, the value programmed in CD must be even to ensure a 50:50 mark/space ratio on the SCK pin, this value can be odd if the internal clock is selected (MCK).

In SPI slave mode:

- The external clock (SCK) selection is forced regardless of the value of the USCLKS field in the US_MR. Likewise, the value written in US_BRGR has no effect, because the clock is provided directly by the signal on the USART SCK pin.
- To obtain correct behavior of the receiver and the transmitter, the external clock (SCK) frequency must be at least 6 times lower than the system clock.

30.7.7.3 Data Transfer

Up to nine data bits are successively shifted out on the TXD pin at each rising or falling edge (depending of CPOL and CPHA) of the programmed serial clock. There is no Start bit, no Parity bit and no Stop bit.

The number of data bits is selected by the CHRL field and the MODE 9 bit in the US_MR. The nine bits are selected by setting the MODE 9 bit regardless of the CHRL field. The MSB data bit is always sent first in SPI mode (master or slave).

Four combinations of polarity and phase are available for data transfers. The clock polarity is programmed with the CPOL bit in the US_MR. The clock phase is programmed with the CPHA bit. These two parameters determine the edges of the clock signal upon which data is driven and sampled. Each of the two parameters has two possible states, resulting in four possible combinations that are incompatible with one another. Thus, a master/slave pair must use the same parameter pair values to communicate. If multiple slaves are used and fixed in different configurations, the master must reconfigure itself each time it needs to communicate with a different slave.

Table 30-14. SPI Bus Protocol Mode

SPI Bus Protocol Mode	CPOL	CPHA
0	0	1
1	0	0
2	1	1
3	1	0

Figure 30-28. SPI Transfer Format (CPHA = 1, 8 bits per transfer)

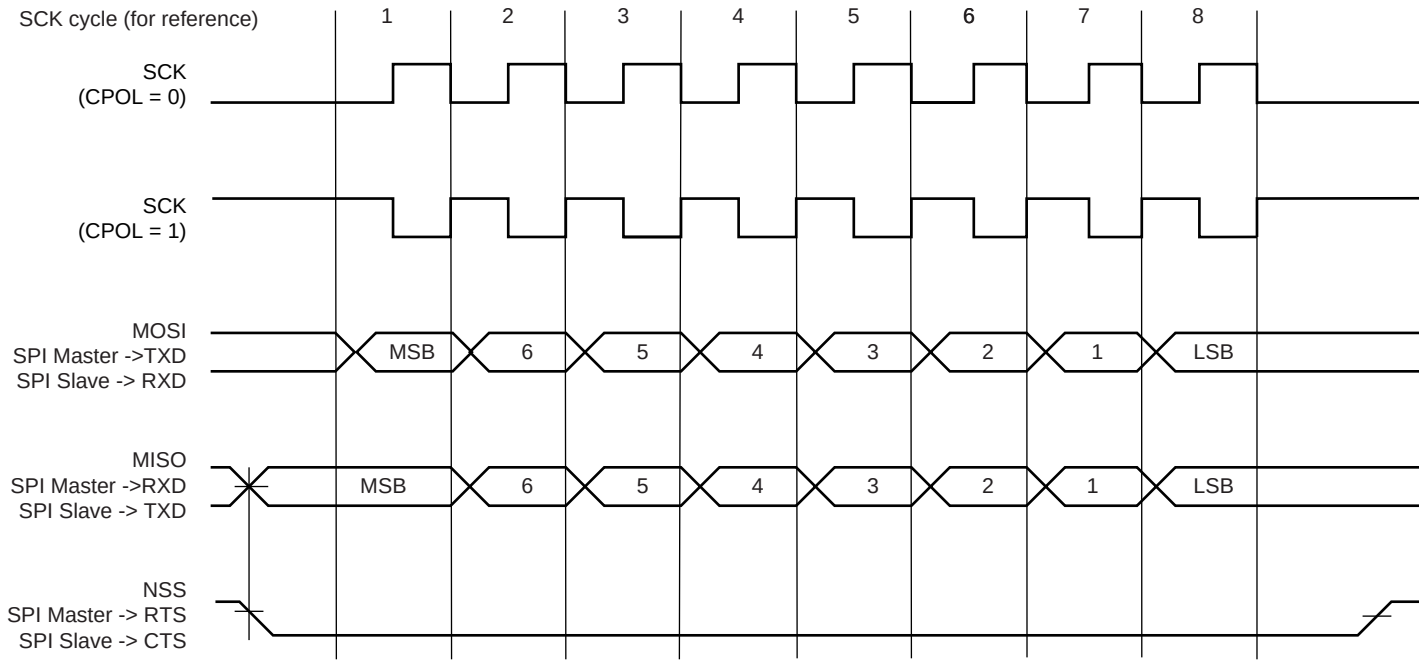
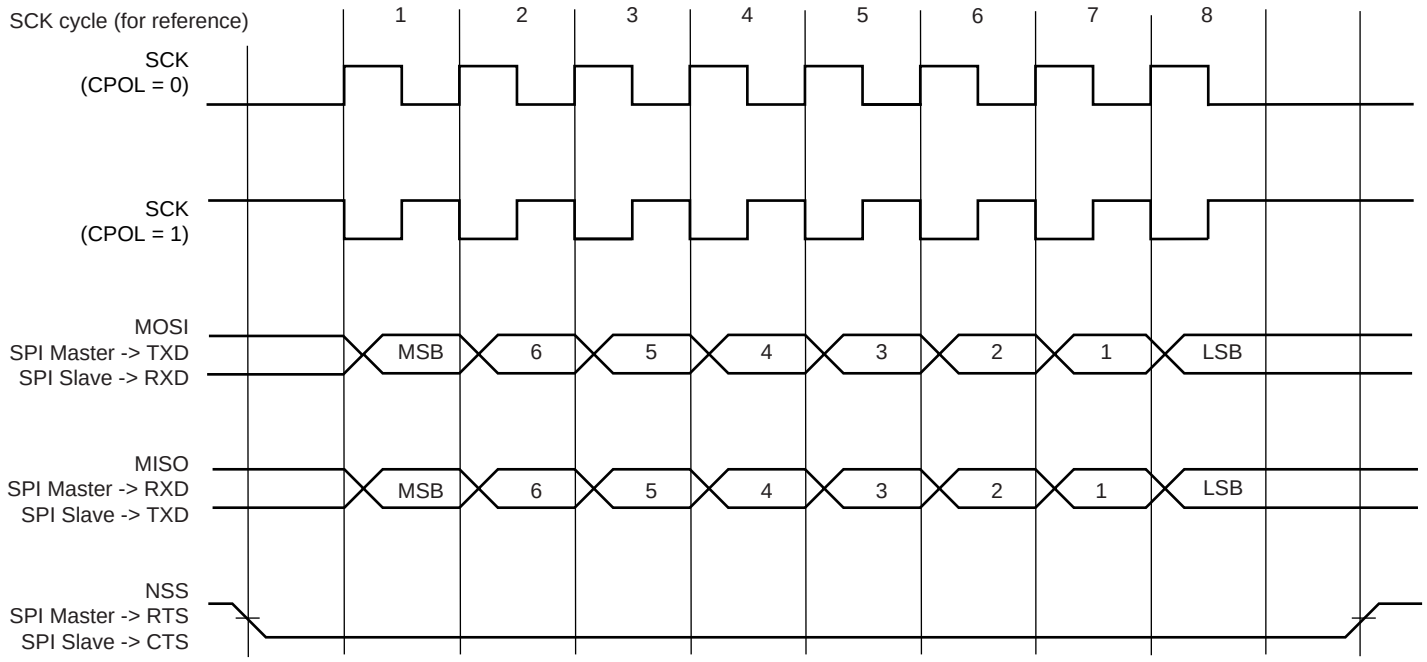


Figure 30-29. SPI Transfer Format (CPHA = 0, 8 bits per transfer)



30.7.7.4 Receiver and Transmitter Control

See “Receiver and Transmitter Control” on page 690.

30.7.7.5 Character Transmission

The characters are sent by writing in the Transmit Holding register (US_THR). An additional condition for transmitting a character can be added when the USART is configured in SPI master mode. In the USART_MR, the value configured on INACK field can prevent any character transmission (even if US_THR has been written) while the receiver side is not ready (character not read). When WRDBT equals 0, the character is transmitted whatever the receiver status. If WRDBT is set to 1, the transmitter waits for the Receive Holding register (US_RHR) to be read before transmitting the character (RXRDY flag cleared), thus preventing any overflow (character loss) on the receiver side.

The 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 one to the RSTSTA (Reset Status) bit in US_CR.

In SPI master mode, the slave select line (NSS) is asserted at low level 1 Tbit (Time bit) before the transmission of the MSB bit and released at high level 1 Tbit after the transmission of the LSB bit. So, the slave select line (NSS) is always released between each character transmission and a minimum delay of 3 Tbits always inserted. However, in order to address slave devices supporting the CSAAT mode (Chip Select Active After Transfer), the slave select line (NSS) can be forced at low level by writing a one to the RTSEN bit in the US_CR. The slave select line (NSS) can be released at high level only by writing a one to the RTSDIS bit in the US_CR (for example, when all data have been transferred to the slave device).

In SPI slave mode, the transmitter does not require a falling edge of the slave select line (NSS) to initiate a character transmission but only a low level. However, this low level must be present on the slave select line (NSS) at least 1 Tbit before the first serial clock cycle corresponding to the MSB bit.

30.7.7.6 Character Reception

When a character reception is completed, it is transferred to the Receive Holding register (US_RHR) and the RXRDY bit in the Status register (US_CSR) rises. If a character is completed while RXRDY is set, the OVRE (Overrun Error) bit is set. The last character is transferred into US_RHR and overwrites the previous one. The OVRE bit is cleared by writing a one to the RSTSTA (Reset Status) bit the US_CR.

To ensure correct behavior of the receiver in SPI slave mode, the master device sending the frame must ensure a minimum delay of 1 Tbit between each character transmission. The receiver does not require a falling edge of the slave select line (NSS) to initiate a character reception but only a low level. However, this low level must be present on the slave select line (NSS) at least 1 Tbit before the first serial clock cycle corresponding to the MSB bit.

30.7.7.7 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 time-out value is (field TO) in the US_RTOR.

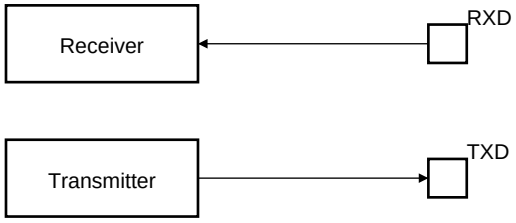
30.7.8 Test Modes

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.

30.7.8.1 Normal Mode

Normal mode connects the RXD pin on the receiver input and the transmitter output on the TXD pin.

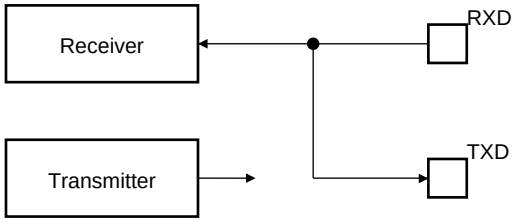
Figure 30-30. Normal Mode Configuration



30.7.8.2 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 30-31. 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.

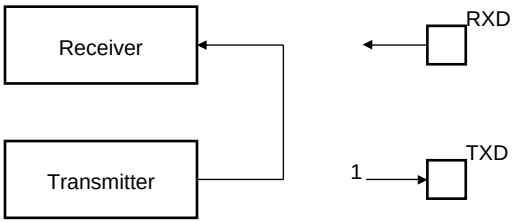
Figure 30-31. Automatic Echo Mode Configuration



30.7.8.3 Local Loopback Mode

Local loopback mode connects the output of the transmitter directly to the input of the receiver, as shown in Figure 30-32. 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.

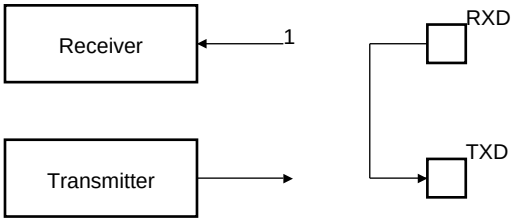
Figure 30-32. Local Loopback Mode Configuration



30.7.8.4 Remote Loopback Mode

Remote loopback mode directly connects the RXD pin to the TXD pin, as shown in Figure 30-33. The transmitter and the receiver are disabled and have no effect. This mode allows bit-by-bit retransmission.

Figure 30-33. Remote Loopback Mode Configuration



30.7.9 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 FI DI RATIO Register"
- "USART IrDA FILTER Register"

30.8 Universal Synchronous Asynchronous Receiver Transmitter (USART) User Interface

Table 30-15. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	Control Register	US_CR	Write-only	–
0x0004	Mode Register	US_MR	Read/Write	–
0x0008	Interrupt Enable Register	US_IER	Write-only	–
0x000C	Interrupt Disable Register	US_IDR	Write-only	–
0x0010	Interrupt Mask Register	US_IMR	Read-only	0x0
0x0014	Channel Status Register	US_CSR	Read-only	–
0x0018	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
0x2C–0x3C	Reserved	–	–	–
0x0040	FI DI Ratio Register	US_FIDI	Read/Write	0x174
0x0044	Number of Errors Register	US_NER	Read-only	–
0x0048	Reserved	–	–	–
0x004C	IrDA Filter Register	US_IF	Read/Write	0x0
0x0050	Reserved	–	–	–
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	–	–	–
0x100–0x128	Reserved for PDC Registers	–	–	–

30.8.1 USART Control Register

Name: US_CR

Address: 0x40024000

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 “USART Control Register (SPI_MODE)” on page 716.

- **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 and RXBRK in US_CSR.

- **STTBRK: Start Break**

0: No effect.

1: Starts transmission of a break after the characters present in US_THR and the Transmit Shift Register have been transmitted. No effect if a break is already being transmitted.

- **STPBRK: Stop Break**

0: No effect.

1: Stops transmission of the break after a minimum of one character length and transmits a high level during 12-bit periods. No effect if no break is being transmitted.

- **STTTO: Start Time-out**

0: No effect.

1: Starts waiting for a character before clocking the time-out counter. Resets the status bit TIMEOUT in US_CSR.

- **SENDA: Send Address**

0: No effect.

1: In multidrop mode only, the next character written to the US_THR is sent with the address bit set.

- **RSTIT: Reset Iterations**

0: No effect.

1: Resets ITERATION in US_CSR. No effect if the ISO7816 is not enabled.

- **RSTNACK: Reset Non Acknowledge**

0: No effect

1: Resets NACK in US_CSR.

- **RETTO: Rearm Time-out**

0: No effect

1: Restart Time-out

- **RTSEN: Request to Send Enable**

0: No effect.

1: Drives the pin RTS to 0.

- **RTSDIS: Request to Send Disable**

0: No effect.

1: Drives the pin RTS to 1.

30.8.2 USART Control Register (SPI_MODE)

Name: US_CR (SPI_MODE)

Address: 0x40024000

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	RCS	FCS	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	RSTSTA
7	6	5	4	3	2	1	0
TXDIS	TXEN	RXDIS	RXEN	RSTTX	RSTRX	–	–

This configuration is relevant only if USART_MODE=0xE or 0xF in “USART Mode Register” on page 718.

- **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).

30.8.3 USART Mode Register

Name: US_MR

Address: 0x40024004

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	FILTER	–	MAX_ITERATION		

23	22	21	20	19	18	17	16
INVDATA	–	DSNACK	INACK	OVER	CLKO	MODE9	MSBF

15	14	13	12	11	10	9	8
CHMODE		NBSTOP		PAR			SYNC

7	6	5	4	3	2	1	0
CHRL		USCLKS		USART_MODE			

This register can only be written if the WPEN bit is cleared in “USART Write Protection Mode Register” on page 740.

For SPI configuration, see “USART Mode Register (SPI_MODE)” on page 721.

- **USART_MODE: USART Mode of Operation**

Value	Name	Description
0x0	NORMAL	Normal mode
0x1	RS485	RS485
0x2	HW_HANDSHAKING	Hardware Handshaking
0x4	IS07816_T_0	IS07816 Protocol: T = 0
0x6	IS07816_T_1	IS07816 Protocol: T = 1
0x8	IRDA	IrDA
0xE	SPI_MASTER	SPI master
0xF	SPI_SLAVE	SPI Slave

The PDC transfers are supported in all USART modes of operation.

- **USCLKS: Clock Selection**

Value	Name	Description
0	MCK	master Clock MCK is selected
1	DIV	Internal Clock Divided MCK/DIV (DIV=8) is selected
3	SCK	Serial Clock SLK is selected

- **CHRL: Character Length**

Value	Name	Description
0	5_BIT	Character length is 5 bits

1	6_BIT	Character length is 6 bits
2	7_BIT	Character length is 7 bits
3	8_BIT	Character length is 8 bits

- **SYNC: Synchronous Mode Select**

0: USART operates in asynchronous mode.

1: USART operates in synchronous mode.

- **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
5	NO	No parity
6	MULTIDROP	Multidrop mode

- **NBSTOP: Number of Stop Bits**

Value	Name	Description
0	1_BIT	1 stop bit
1	1_5_BIT	1.5 stop bit (SYNC = 0) or reserved (SYNC = 1)
2	2_BIT	2 stop bits

- **CHMODE: Channel Mode**

Value	Name	Description
0	NORMAL	Normal mode
1	AUTOMATIC	Automatic Echo. Receiver input is connected to the TXD pin.
2	LOCAL_LOOPBACK	Local Loopback. Transmitter output is connected to the Receiver Input.
3	REMOTE_LOOPBACK	Remote Loopback. RXD pin is internally connected to the TXD pin.

- **MSBF: Bit Order**

0: Least significant bit is sent/received first.

1: Most significant bit is sent/received first.

- **MODE9: 9-bit Character Length**

0: CHRL defines character length.

1: 9-bit character length.

- **CLKO: Clock Output Select**

0: The USART does not drive the SCK pin.

1: The USART drives the SCK pin if USCLKS does not select the external clock SCK.

- **OVER: Oversampling Mode**

0: 16x Oversampling.

1: 8x Oversampling.

- **INACK: Inhibit Non Acknowledge**

0: The NACK is generated.

1: The NACK is not generated.

- **DSNACK: Disable Successive NACK**

0: NACK is sent on the ISO line as soon as a parity error occurs in the received character (unless INACK is set).

1: Successive parity errors are counted up to the value specified in the MAX_ITERATION field. These parity errors generate a NACK on the ISO line. As soon as this value is reached, no additional NACK is sent on the ISO line. The flag ITERATION is asserted.

- **INVDATA: Inverted Data**

0: The data field transmitted on TXD line is the same as the one written in US_THR register or the content read in US_RHR is the same as RXD line. Normal mode of operation.

1: The data field transmitted on TXD line is inverted (voltage polarity only) compared to the value written on US_THR register or the content read in US_RHR is inverted compared to what is received on RXD line (or ISO7816 IO line). Inverted mode of operation, useful for contactless card application. To be used with configuration bit MSBF.

- **MAX_ITERATION: Maximum Number of Automatic Iteration**

0–7: Defines the maximum number of iterations in mode ISO7816, protocol T = 0.

- **FILTER: Infrared Receive Line Filter**

0: The USART does not filter the receive line.

1: The USART filters the receive line using a three-sample filter (1/16-bit clock) (2 over 3 majority).

30.8.4 USART Mode Register (SPI_MODE)

Name: US_MR (SPI_MODE)

Address: 0x40024004

Access: Read/Write

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

This configuration is relevant only if USART_MODE = 0xE or 0xF in “USART Mode Register” on page 718.

This register can only be written if the WPEN bit is cleared in “USART Write Protection Mode Register” on page 740.

- **USART_MODE: USART Mode of Operation**

Value	Name	Description
0xE	SPI_MASTER	SPI master
0xF	SPI_SLAVE	SPI Slave

- **USCLKS: Clock Selection**

Value	Name	Description
0	MCK	master Clock MCK is selected
1	DIV	Internal Clock Divided MCK/DIV (DIV=8) is selected
3	SCK	Serial Clock SLK is selected

- **CHRL: Character Length**

Value	Name	Description
3	8_BIT	Character length is 8 bits

- **CPHA: SPI Clock Phase**
  - Applicable if USART operates in SPI mode (USART_MODE = 0xE or 0xF):

0: Data is changed on the leading edge of SPCK and captured on the following edge of SPCK.

1: Data is captured on the leading edge of SPCK and changed on the following edge of SPCK.

CPHA determines which edge of SPCK causes data to change and which edge causes data to be captured. CPHA is used with CPOL to produce the required clock/data relationship between master and slave devices.

- **CHMODE: Channel Mode**

Value	Name	Description
0	NORMAL	Normal mode
1	AUTOMATIC	Automatic echo mode. Receiver input is connected to the TXD pin.
2	LOCAL_LOOPBACK	Local loopback mode. Transmitter output is connected to the Receiver Input.
3	REMOTE_LOOPBACK	Remote loopback mode. RXD pin is internally connected to the TXD pin.

- **CPOL: SPI Clock Polarity**

Applicable if USART operates in SPI mode (slave or master, USART_MODE = 0xE or 0xF):

0: The inactive state value of SPCK is logic level zero.

1: The inactive state value of SPCK is logic level one.

CPOL is used to determine the inactive state value of the serial clock (SPCK). It is used with CPHA to produce the required clock/data relationship between master and slave devices.

- **WRDBT: Wait Read Data Before Transfer**

0: The character transmission starts as soon as a character is written into US_THR register (assuming TXRDY was set).

1: The character transmission starts when a character is written and only if RXRDY flag is cleared (Receive Holding Register has been read).

30.8.5 USART Interrupt Enable Register

Name: US_IER

Address: 0x40024008

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
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 “USART Interrupt Enable Register (SPI_MODE)” on page 724.

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 Transfer Interrupt Enable (available in all USART modes of operation)**
- **ENDTX: End of Transmit Interrupt Enable (available in all USART modes of operation)**
- **OVRE: Overrun Error Interrupt Enable**
- **FRAME: Framing Error Interrupt Enable**
- **PARE: Parity Error Interrupt Enable**
- **TIMEOUT: Time-out Interrupt Enable**
- **TXEMPTY: TXEMPTY Interrupt Enable**
- **ITER: Max number of Repetitions Reached Interrupt Enable**
- **TXBUFE: Buffer Empty Interrupt Enable (available in all USART modes of operation)**
- **RXBUFF: Buffer Full Interrupt Enable (available in all USART modes of operation)**
- **NACK: Non AcknowledgeInterrupt Enable**
- **CTSIC: Clear to Send Input Change Interrupt Enable**

30.8.6 USART Interrupt Enable Register (SPI_MODE)

Name: US_IER (SPI_MODE)

Address: 0x40024008

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	UNRE	TXEMPTY	–

7	6	5	4	3	2	1	0
–	–	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

This configuration is relevant only if USART_MODE = 0xE or 0xF in “USART Mode Register” on page 718.

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**
- **OVRE: Overrun Error Interrupt Enable**
- **TXEMPTY: TXEMPTY Interrupt Enable**
- **UNRE: SPI Underrun Error Interrupt Enable**

30.8.7 USART Interrupt Disable Register

Name: US_IDR

Address: 0x4002400C

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

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 “USART Interrupt Disable Register (SPI_MODE)” on page 726.

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 Transfer Interrupt Disable (available in all USART modes of operation)**
- **ENDTX: End of Transmit Interrupt Disable (available in all USART modes of operation)**
- **OVRE: Overrun Error Interrupt Enable**
- **FRAME: Framing Error Interrupt Disable**
- **PARE: Parity Error Interrupt Disable**
- **TIMEOUT: Time-out Interrupt Disable**
- **TXEMPTY: TXEMPTY Interrupt Disable**
- **ITER: Max Number of Repetitions Reached Interrupt Disable**
- **TXBUFE: Buffer Empty Interrupt Disable (available in all USART modes of operation)**
- **RXBUFF: Buffer Full Interrupt Disable (available in all USART modes of operation)**
- **NACK: Non AcknowledgeInterrupt Disable**
- **CTSIC: Clear to Send Input Change Interrupt Disable**

30.8.8 USART Interrupt Disable Register (SPI_MODE)

Name: US_IDR (SPI_MODE)

Address: 0x4002400C

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	UNRE	TXEMPTY	–
7	6	5	4	3	2	1	0
–	–	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

This configuration is relevant only if USART_MODE = 0xE or 0xF in “USART Mode Register” on page 718.

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**
- **OVRE: Overrun Error Interrupt Disable**
- **TXEMPTY: TXEMPTY Interrupt Disable**
- **UNRE: SPI Underrun Error Interrupt Disable**

30.8.9 USART Interrupt Mask Register

Name: US_IMR

Address: 0x40024010

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

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 “USART Interrupt Mask Register (SPI_MODE)” on page 728.

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 Transfer Interrupt Mask (available in all USART modes of operation)**
- **ENDTX: End of Transmit Interrupt Mask (available in all USART modes of operation)**
- **OVRE: Overrun Error Interrupt Mask**
- **FRAME: Framing Error Interrupt Mask**
- **PARE: Parity Error Interrupt Mask**
- **TIMEOUT: Time-out Interrupt Mask**
- **TXEMPTY: TXEMPTY Interrupt Mask**
- **ITER: Max Number of Repetitions Reached Interrupt Mask**
- **TXBUFE: Buffer Empty Interrupt Mask (available in all USART modes of operation)**
- **RXBUFF: Buffer Full Interrupt Mask (available in all USART modes of operation)**
- **NACK: Non AcknowledgeInterrupt Mask**
- **CTSIC: Clear to Send Input Change Interrupt Mask**

30.8.10 USART Interrupt Mask Register (SPI_MODE)

Name: US_IMR (SPI_MODE)

Address: 0x40024010

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	UNRE	TXEMPTY	–
7	6	5	4	3	2	1	0
–	–	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

This configuration is relevant only if USART_MODE = 0xE or 0xF in “USART Mode Register” on page 718.

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**
- **OVRE: Overrun Error Interrupt Mask**
- **TXEMPTY: TXEMPTY Interrupt Mask**
- **UNRE: SPI Underrun Error Interrupt Mask**

30.8.11 USART Channel Status Register

Name: US_CSR

Address: 0x40024014

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

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 “USART Channel Status Register (SPI_MODE)” on page 731.

- **RXRDY: Receiver Ready**

0: No complete character has been received since the last read of US_RHR or the receiver is disabled. If characters were being received when the receiver was disabled, RXRDY changes to 1 when the receiver is enabled.

1: At least one complete character has been received and US_RHR has not yet been read.

- **TXRDY: Transmitter Ready**

0: A character is in the US_THR waiting to be transferred to the Transmit Shift Register, or an STTBRK command has been requested, or the transmitter is disabled. As soon as the transmitter is enabled, TXRDY becomes 1.

1: There is no character in the US_THR.

- **RXBRK: Break Received/End of Break**

0: No break received or end of break detected since the last RSTSTA.

1: Break received or end of break detected since the last RSTSTA.

- **ENDRX: End of Receiver Transfer**

0: The end of transfer signal from the receive PDC channel is inactive.

1: The end of transfer signal from the receive PDC channel is active.

- **ENDTX: End of Transmitter Transfer**

0: The end of transfer signal from the transmit PDC channel is inactive.

1: The end of transfer signal from the transmit PDC channel is active.

- **OVRE: Overrun Error**

0: No overrun error has occurred since the last RSTSTA.

1: At least one overrun error has occurred since the last RSTSTA.

- **FRAME: Framing Error**

0: No stop bit has been detected low since the last RSTSTA.

1: At least one stop bit has been detected low since the last RSTSTA.

- **PARE: Parity Error**

0: No parity error has been detected since the last RSTSTA.

1: At least one parity error has been detected since the last RSTSTA.

- **TIMEOUT: Receiver Time-out**

0: There has not been a time-out since the last Start Time-out command (STTTO in US_CR) or the Time-out Register is 0.

1: There has been a time-out since the last Start Time-out command (STTTO in US_CR).

- **TXEMPTY: Transmitter Empty**

0: There are characters in either US_THR or the Transmit Shift Register, or the transmitter is disabled.

1: There are no characters in US_THR, nor in the Transmit Shift Register.

- **ITER: Max Number of Repetitions Reached**

0: Maximum number of repetitions has not been reached since the last RSTSTA.

1: Maximum number of repetitions has been reached since the last RSTSTA.

- **TXBUFE: Transmission Buffer Empty**

0: The signal buffer empty from the transmit PDC channel is inactive.

1: The signal buffer empty from the transmit PDC channel is active.

- **RXBUFF: Reception Buffer Full**

0: The signal Buffer Full from the Receive PDC channel is inactive.

1: The signal Buffer Full from the Receive PDC channel is active.

- **NACK: Non AcknowledgeInterrupt**

0: Non acknowledge has not been detected since the last RSTNACK.

1: At least one non acknowledge has been detected since the last RSTNACK.

- **CTSIC: Clear to Send Input Change Flag**

0: No input change has been detected on the CTS pin since the last read of US_CSR.

1: At least one input change has been detected on the CTS pin since the last read of US_CSR.

- **CTS: Image of CTS Input**

0: CTS is set to 0.

1: CTS is set to 1.

30.8.12 USART Channel Status Register (SPI_MODE)

Name: US_CSR (SPI_MODE)

Address: 0x40024014

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	UNRE	TXEMPTY	–
7	6	5	4	3	2	1	0
–	–	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

This configuration is relevant only if USART_MODE = 0xE or 0xF in “USART Mode Register” on page 718.

- **RXRDY: Receiver Ready**

0: No complete character has been received since the last read of US_RHR or the receiver is disabled. If characters were being received when the receiver was disabled, RXRDY changes to 1 when the receiver is enabled.

1: At least one complete character has been received and US_RHR has not yet been read.

- **TXRDY: Transmitter Ready**

0: A character is in the US_THR waiting to be transferred to the Transmit Shift Register or the transmitter is disabled. As soon as the transmitter is enabled, TXRDY becomes 1.

1: There is no character in the US_THR.

- **OVRE: Overrun Error**

0: No overrun error has occurred since the last RSTSTA.

1: At least one overrun error has occurred since the last RSTSTA.

- **TXEMPTY: Transmitter Empty**

0: There are characters in either US_THR or the Transmit Shift Register, or the transmitter is disabled.

1: There are no characters in US_THR, nor in the Transmit Shift Register.

- **UNRE: Underrun Error**

0: No SPI underrun error has occurred since the last RSTSTA.

1: At least one SPI underrun error has occurred since the last RSTSTA.

30.8.13 USART Receive Holding Register

Name: US_RHR

Address: 0x40024018

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.

30.8.14 USART Transmit Holding Register

Name: US_THR

Address: 0x4002401C

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.

30.8.15 USART Baud Rate Generator Register

Name: US_BRGR

Address: 0x40024020

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	FP		

15	14	13	12	11	10	9	8
CD							

7	6	5	4	3	2	1	0
CD							

This register can only be written if the WPEN bit is cleared in “USART Write Protection Mode Register” on page 740.

- **CD: Clock Divider**

	USART_MODE ≠ ISO7816			
	SYNC = 0		SYNC = 1 or USART_MODE = SPI (master or Slave)	USART_MODE = ISO7816
CD	OVER = 0	OVER = 1		
0	Baud Rate Clock Disabled			
1 to 65535	Baud Rate = Selected Clock/(16*CD)	Baud Rate = Selected Clock/(8*CD)	Baud Rate = Selected Clock/CD	Baud Rate = Selected Clock/(FI_DI_RATIO*CD)

- **FP: Fractional Part**

0: Fractional divider is disabled.

1–7: Baud rate resolution, defined by FP x 1/8.

30.8.16 USART Receiver Time-out Register

Name: US_RTOR

Address: 0x40024024

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–

15	14	13	12	11	10	9	8
TO							

7	6	5	4	3	2	1	0
TO							

This register can only be written if the WPEN bit is cleared in “USART Write Protection Mode Register” on page 740.

- **TO: Time-out Value**

0: The receiver time-out is disabled.

1–65535: The receiver time-out is enabled and the time-out delay is TO x bit period.

30.8.17 USART Transmitter Timeguard Register

Name: US_TTGR

Address: 0x40024028

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in “USART Write Protection Mode Register” on page 740.

- **TG: Timeguard Value**

0: The transmitter timeguard is disabled.

1–255: The transmitter timeguard is enabled and the timeguard delay is TG x bit period.

30.8.18 USART FI DI RATIO Register

Name: US_FIDI

Address: 0x40024040

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 “USART Write Protection Mode Register” on page 740.

- **FI_DI_RATIO: FI Over DI Ratio Value**

0: If ISO7816 mode is selected, the baud rate generator generates no signal.

1– 2047: If ISO7816 mode is selected, the baud rate is the clock provided on SCK divided by FI_DI_RATIO.

30.8.19 USART Number of Errors Register

Name: US_NER

Address: 0x40024044

Access: Read-only

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

This register is relevant only if USART_MODE = 0x4 or 0x6 in “USART Mode Register” on page 718.

- **NB_ERRORS: Number of Errors**

Total number of errors that occurred during an ISO7816 transfer. This register automatically clears when read.

30.8.20 USART IrDA FILTER Register

Name: US_IF

Address: 0x4002404C

Access: Read/Write

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

This register is relevant only if USART_MODE = 0x8 in “USART Mode Register” on page 718.

This register can only be written if the WPEN bit is cleared in “USART Write Protection Mode Register” on page 740.

- **IRDA_FILTER: IrDA Filter**

The IRDA_FILTER value must be defined to meet the following criteria:

t_{MCK} * (IRDA_FILTER + 3) < 1.41 µs

30.8.21 USART Write Protection Mode Register

Name: US_WPMR

Address: 0x400240E4

Access: Read/Write

Reset: See Table 30-15

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 30.7.9 ”Register Write Protection” for the list of registers that can be wirte-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.

30.8.22 USART Write Protection Status Register

Name: US_WPSR

Address: 0x400240E8

Access: Read-only

Reset: See Table 30-15

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.

31. Timer Counter (TC)

31.1 Description

The Timer Counter (TC) includes 3 identical 16-bit Timer Counter channels.

Each channel can be independently programmed to perform a wide range of functions including frequency measurement, event counting, interval measurement, pulse generation, delay timing and pulse width modulation.

Each channel has three external clock inputs, five internal clock inputs and two multi-purpose input/output signals which can be configured by the user. Each channel drives an internal interrupt signal which can be programmed to generate processor interrupts.

The Timer Counter 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

Table 31-1 gives the assignment of the device Timer Counter clock inputs common to Timer Counter 0 to 2.

Table 31-1. 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 Slow Clock is selected for Master Clock (CSS = 0 in PMC Master Clock Register), TIMER_CLOCK5 input is equivalent to Master Clock.

31.2 Embedded Characteristics

- Provides 3 16-bit Timer Counter channels
- Wide range of functions including:
  - Frequency measurement
  - Event counting
  - Interval measurement
  - Pulse generation
  - Delay timing
  - Pulse Width Modulation
  - Up/down capabilities
  - 2-bit gray up/down count for stepper motor
- Each channel is user-configurable and contains:
  - Three external clock inputs
  - Five Internal clock inputs
  - Two multi-purpose input/output signals acting as trigger event
- Internal interrupt signal
- Two global registers that act on all TC channels
- Read of the Capture registers by the PDC
- Register Write Protection

31.3 Block Diagram

Figure 31-1. Timer Counter Block Diagram

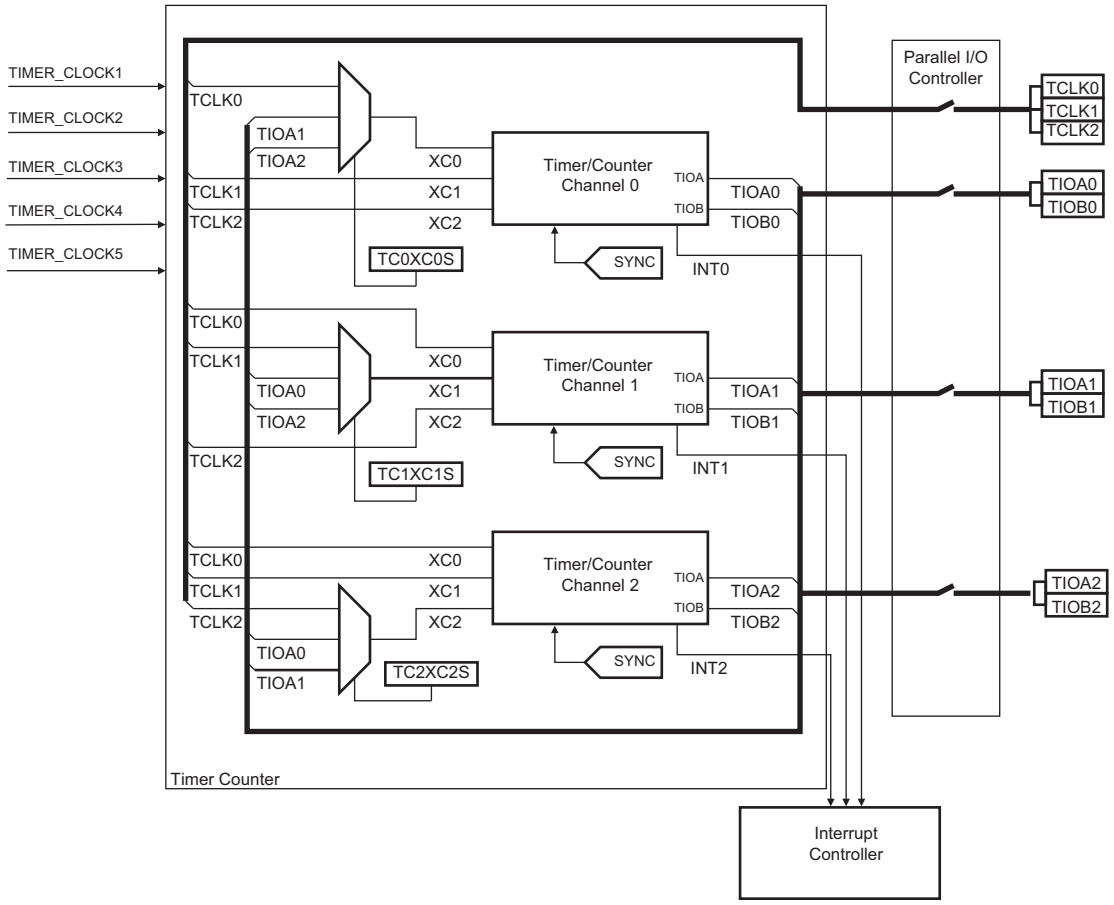


Table 31-2. Signal Name Description

Block/Channel	Signal Name	Description
Channel Signal	XC0, XC1, XC2	External Clock Inputs
	TIOA	Capture Mode: Timer Counter Input Waveform Mode: Timer Counter Output
	TIOB	Capture Mode: Timer Counter Input Waveform Mode: Timer Counter Input/Output
	INT	Interrupt Signal Output (internal signal)
	SYNC	Synchronization Input Signal (from configuration register)

31.4 Pin Name List

Table 31-3. 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

31.5 Product Dependencies

31.5.1 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.

31.5.2 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.

31.5.3 Interrupt

The TC has an interrupt line connected to the Interrupt Controller (IC). Handling the TC interrupt requires programming the IC before configuring the TC.

31.6 Functional Description

31.6.1 TC Description

The 3 channels of the Timer Counter are independent and identical in operation. The registers for channel programming are listed in Table 31-4 “Register Mapping”.

31.6.2 16-bit Counter

Each 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.

31.6.3 Clock Selection

At block level, input clock signals of each channel can either be connected to the external inputs TCLK0, TCLK1 or TCLK2, or be connected to the internal I/O signals TIOA0, TIOA1 or TIOA2 for chaining by programming the TC Block Mode Register (TC_BMR). See Figure 31-2 “Clock Chaining Selection”.

Each channel can independently select an internal or external clock source for its counter:

- Internal clock signals: TIMER_CLOCK1, TIMER_CLOCK2, TIMER_CLOCK3, TIMER_CLOCK4, TIMER_CLOCK5
- External clock signals: XC0, XC1 or XC2

This selection is made by the TCCLKS bits in the TC Channel Mode Register (TC_CMR).

The selected clock can be inverted with the CLKI bit in the TC_CMR. This allows counting on the opposite edges of the clock.

The burst function allows the clock to be validated when an external signal is high. The BURST parameter in the TC_CMR defines this signal (none, XC0, XC1, XC2). See Figure 31-3 “Clock Selection”.

Note: In all cases, if an external clock is used, the duration of each of its levels must be longer than the master clock period. The external clock frequency must be at least 2.5 times lower than the master clock

Figure 31-2. Clock Chaining Selection

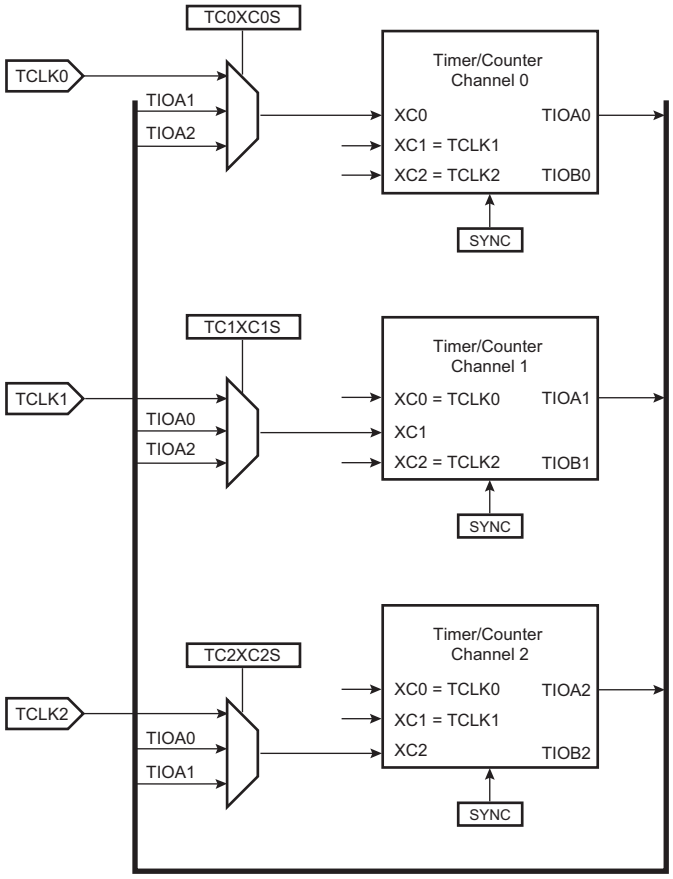
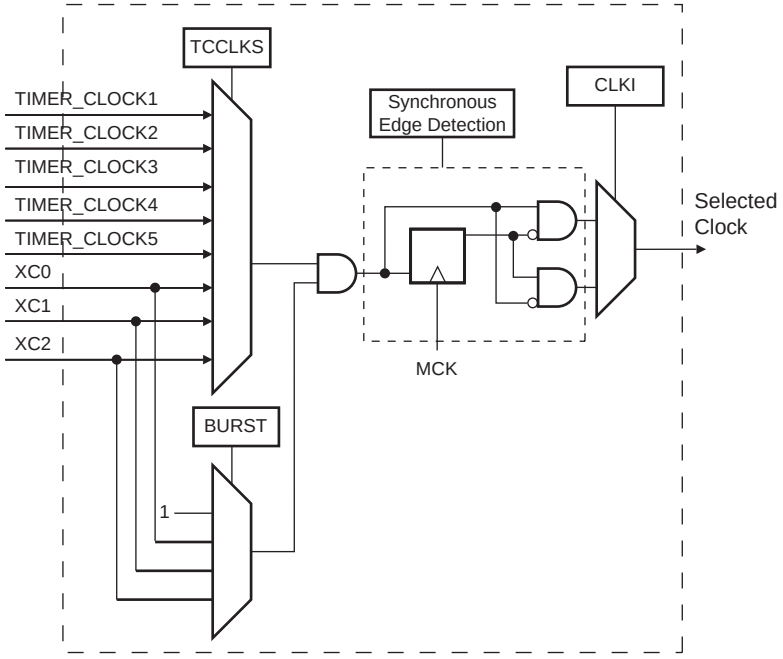


Figure 31-3. Clock Selection

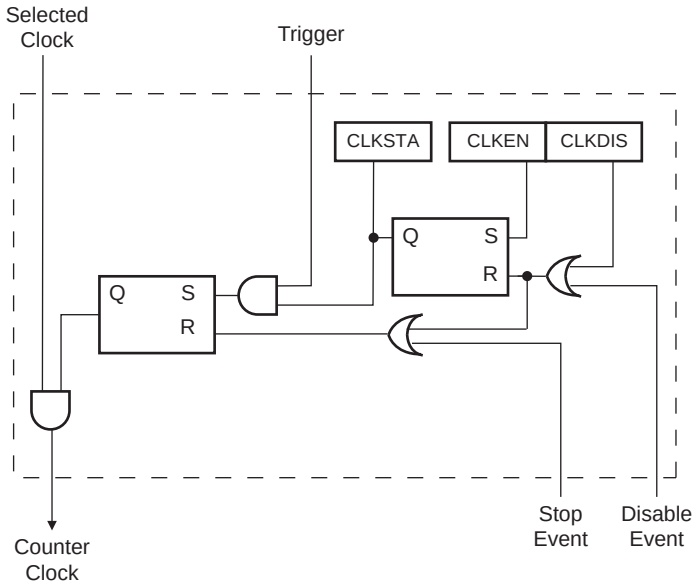


31.6.4 Clock Control

The clock of each counter can be controlled in two different ways: it can be enabled/disabled and started/stopped. See Figure 31-4.

- The clock can be enabled or disabled by the user with the CLKEN and the CLKDIS commands in the TC Channel Control Register (TC_CCR). In Capture Mode it can be disabled by an RB load event if LDBDIS is set to 1 in the TC_CMR. In Waveform Mode, it can be disabled by an RC Compare event if CPCDIS is set to 1 in TC_CMR. When disabled, the start or the stop actions have no effect: only a CLKEN command in the TC_CCR can re-enable the clock. When the clock is enabled, the CLKSTA bit is set in the TC_SR.
- The clock can also be started or stopped: a trigger (software, synchro, external or compare) always starts the clock. The clock can be stopped by an RB load event in Capture Mode (LDBSTOP = 1 in TC_CMR) or a RC compare event in Waveform Mode (CPCSTOP = 1 in TC_CMR). The start and the stop commands have effect only if the clock is enabled.

Figure 31-4. Clock Control



31.6.5 TC Operating Modes

Each channel can independently operate in two different modes:

- Capture Mode provides measurement on signals.
- Waveform Mode provides wave generation.

The TC Operating Mode is programmed with the WAVE bit in the TC Channel Mode Register.

In Capture Mode, TIOA and TIOB are configured as inputs.

In Waveform Mode, TIOA is always configured to be an output and TIOB is an output if it is not selected to be the external trigger.

31.6.6 Trigger

A trigger resets the counter and starts the counter clock. Three types of triggers are common to both modes, and a fourth external trigger is available to each mode.

Regardless of the trigger used, it will be taken into account at the following active edge of the selected clock. This means that the counter value can be read differently from zero just after a trigger, especially when a low frequency signal is selected as the clock.

The following triggers are common to both modes:

- Software Trigger: Each channel has a software trigger, available by setting SWTRG in TC_CCR.
- SYNC: Each channel has a synchronization signal SYNC. When asserted, this signal has the same effect as a software trigger. The SYNC signals of all channels are asserted simultaneously by writing TC_BCR (Block Control) with SYNC set.
- Compare RC Trigger: RC is implemented in each channel and can provide a trigger when the counter value matches the RC value if CPCTRG is set in the TC_CMR.

The channel can also be configured to have an external trigger. In Capture Mode, the external trigger signal can be selected between TIOA and TIOB. In Waveform Mode, an external event can be programmed on one of the following signals: TIOB, XC0, XC1 or XC2. This external event can then be programmed to perform a trigger by setting bit ENETRG in the TC_CMR.

If an external trigger is used, the duration of the pulses must be longer than the master clock period in order to be detected.

31.6.7 Capture Operating Mode

This mode is entered by clearing the WAVE bit in the TC_CMR.

Capture Mode allows the TC channel to perform measurements such as pulse timing, frequency, period, duty cycle and phase on TIOA and TIOB signals which are considered as inputs.

Figure 31-6 shows the configuration of the TC channel when programmed in Capture Mode.

31.6.8 Capture Registers A and B

Registers A and B (RA and RB) are used as capture registers. This means that they can be loaded with the counter value when a programmable event occurs on the signal TIOA.

The LDRA field in the TC_CMR defines the TIOA selected edge for the loading of register A, and the LDRB field defines the TIOA selected edge for the loading of Register B.

The subsampling ratio defined by the SBSMPLR field in TC_CMR is applied to these selected edges, so that the loading of Register A and Register B occurs once every 1, 2, 4, 8 or 16 selected edges.

RA is loaded only if it has not been loaded since the last trigger or if RB has been loaded since the last loading of RA.

RB is loaded only if RA has been loaded since the last trigger or the last loading of RB.

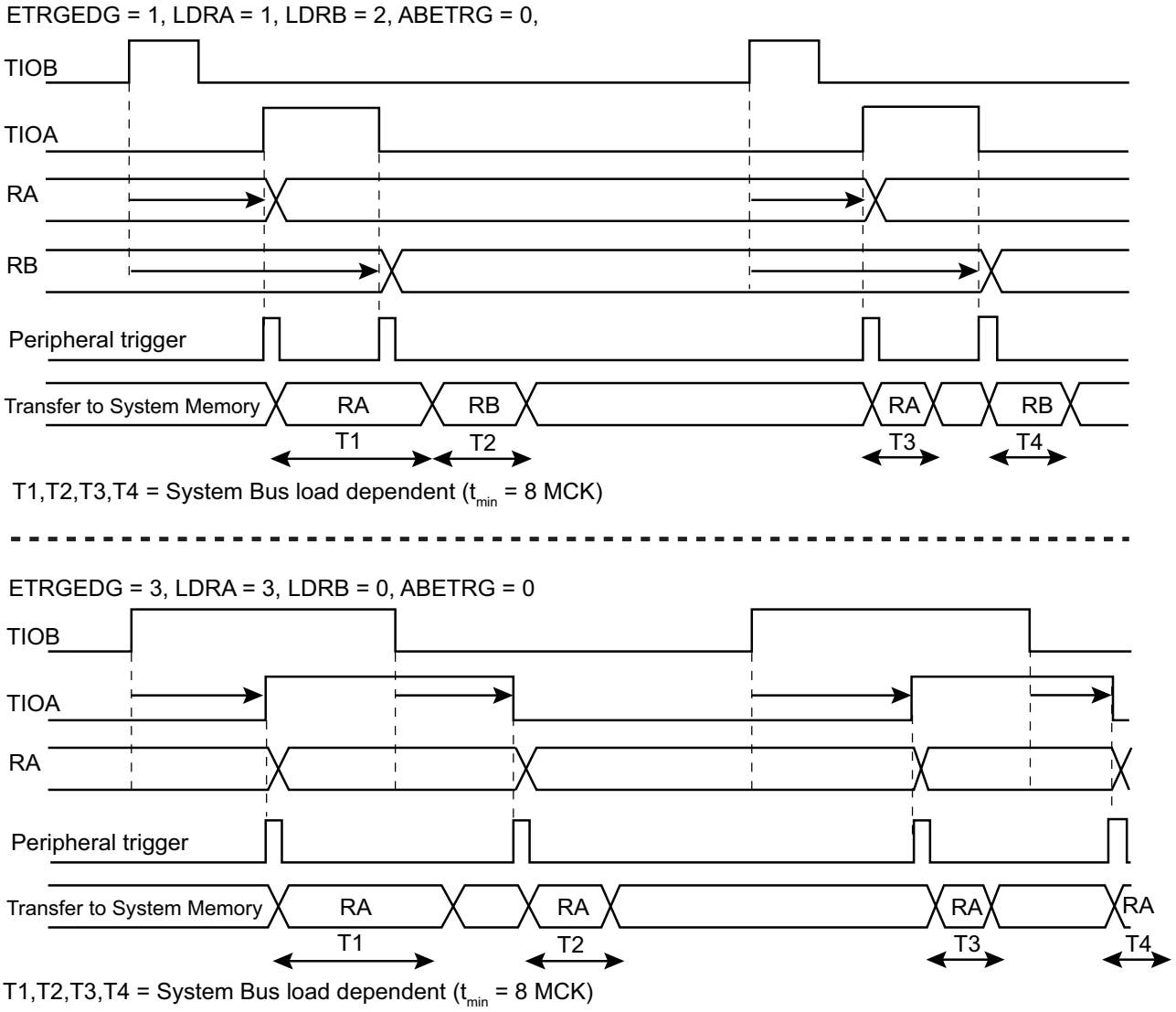
Loading RA or RB before the read of the last value loaded sets the Overrun Error Flag (LOVRS bit) in the TC_SR. In this case, the old value is overwritten.

31.6.9 Transfer with PDC

The PDC can only perform access from timer to system memory.

Figure 31-5 “Example of Transfer with PDC” illustrates how TC_RA and TC_RB can be loaded in the system memory without CPU intervention.

Figure 31-5. Example of Transfer with PDC

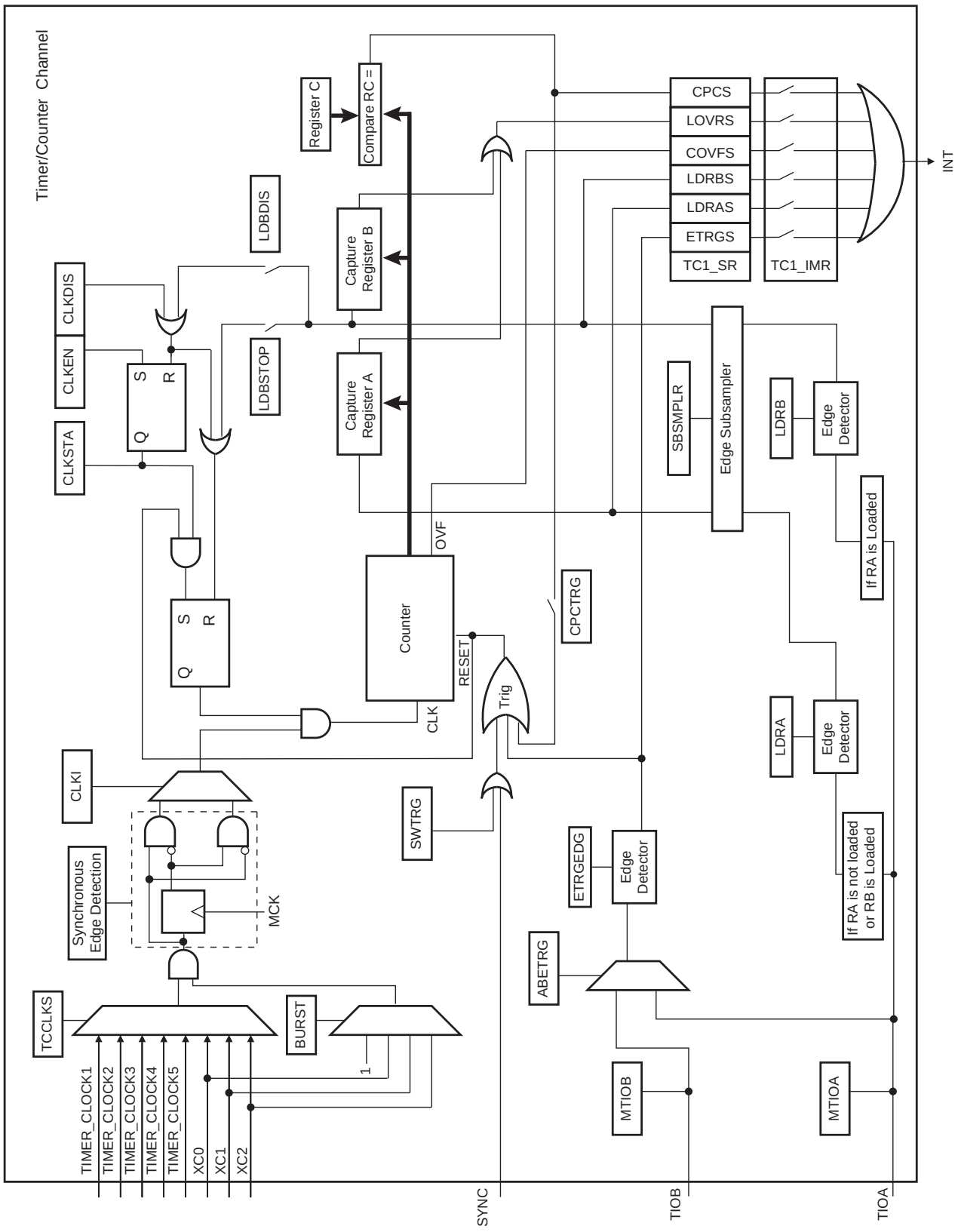


31.6.10 Trigger Conditions

In addition to the SYNC signal, the software trigger and the RC compare trigger, an external trigger can be defined.

The ABETRG bit in the TC_CMR selects TIOA or TIOB input signal as an external trigger . The External Trigger Edge Selection parameter (ETRGEDG field in TC_CMR) defines the edge (rising, falling, or both) detected to generate an external trigger. If ETRGEDG = 0 (none), the external trigger is disabled.

Figure 31-6. Capture Mode



31.6.11 Waveform Operating Mode

Waveform operating mode is entered by setting the WAVE parameter in TC_CMR (Channel Mode Register).

In Waveform Operating Mode the TC channel generates one or two PWM signals with the same frequency and independently programmable duty cycles, or generates different types of one-shot or repetitive pulses.

In this mode, TIOA is configured as an output and TIOB is defined as an output if it is not used as an external event (EEVT parameter in TC_CMR).

Figure 31-7 shows the configuration of the TC channel when programmed in Waveform Operating Mode.

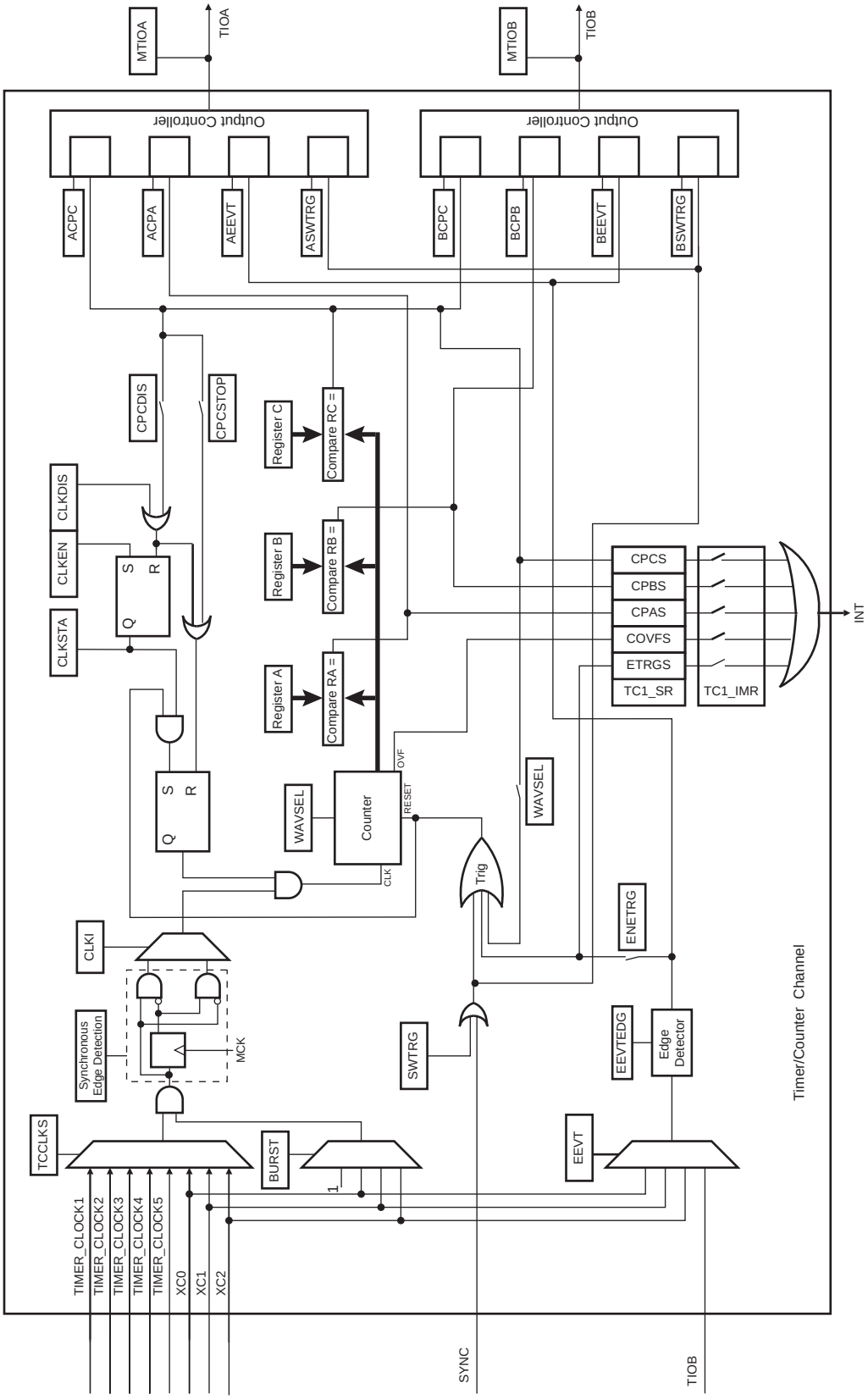
31.6.12 Waveform Selection

Depending on the WAVSEL parameter in TC_CMR (Channel Mode Register), the behavior of TC_CV varies.

With any selection, TC_RA, TC_RB and TC_RC can all be used as compare registers.

RA Compare is used to control the TIOA output, RB Compare is used to control the TIOB output (if correctly configured) and RC Compare is used to control TIOA and/or TIOB outputs.

Figure 31-7. Waveform Mode



31.6.12.1 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 31-8.

An external event trigger or a software trigger can reset the value of TC_CV. It is important to note that the trigger may occur at any time. See Figure 31-9.

RC Compare cannot be programmed to generate a trigger in this configuration. At the same time, RC Compare can stop the counter clock (CPCSTOP = 1 in TC_CMR) and/or disable the counter clock (CPCDIS = 1 in TC_CMR).

Figure 31-8. WAVSEL = 00 without trigger

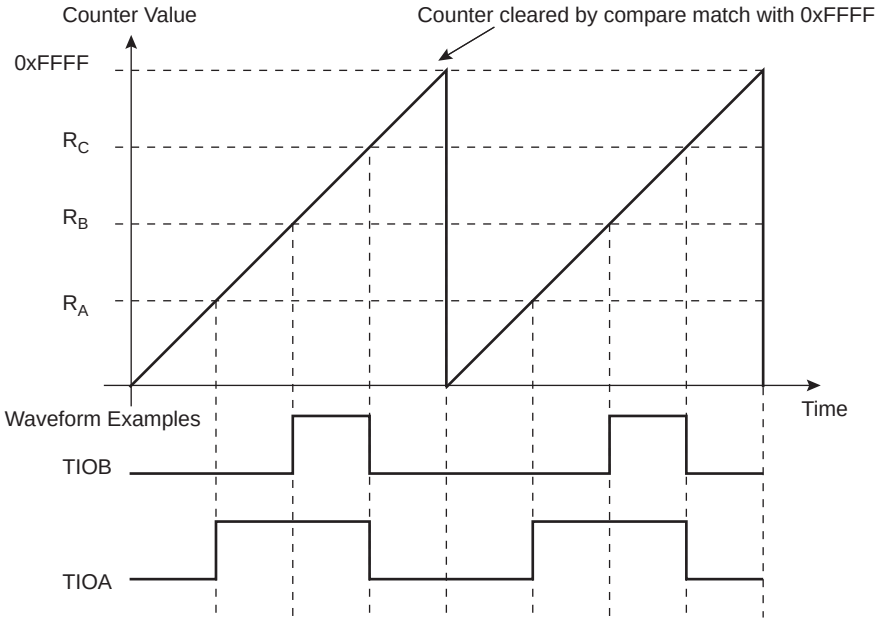
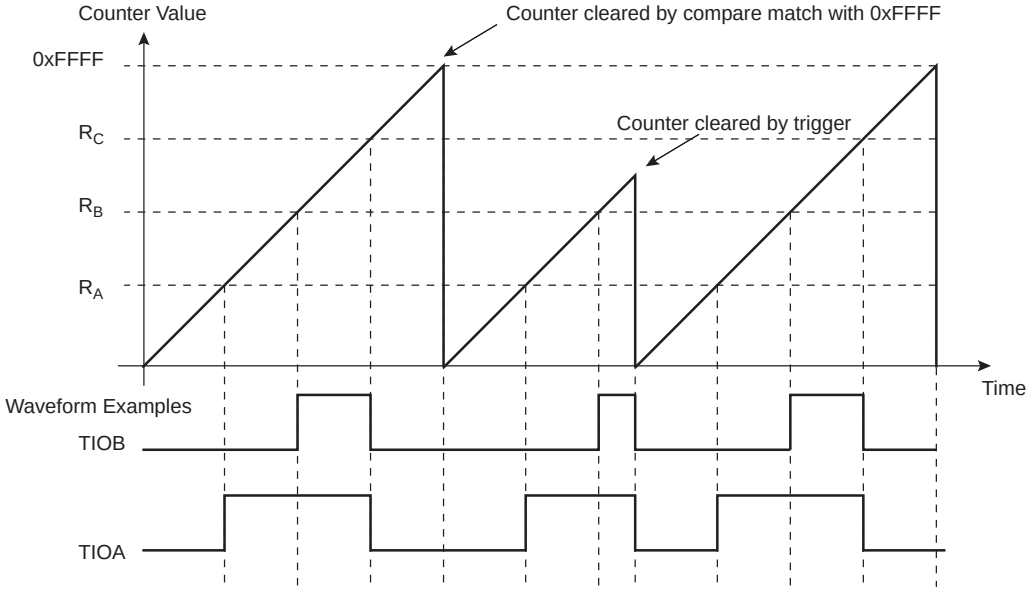


Figure 31-9. WAVSEL= 00 with Trigger



31.6.12.2 WAVSEL = 10

When WAVSEL = 10, the value of TC_CV is incremented from 0 to the value of RC, then automatically reset on a RC Compare. Once the value of TC_CV has been reset, it is then incremented and so on. See Figure 31-10.

It is important to note that TC_CV can be reset at any time by an external event or a software trigger if both are programmed correctly. See Figure 31-11.

In addition, RC Compare can stop the counter clock (CPCSTOP = 1 in TC_CMR) and/or disable the counter clock (CPCDIS = 1 in TC_CMR).

Figure 31-10. WAVSEL = 10 without Trigger

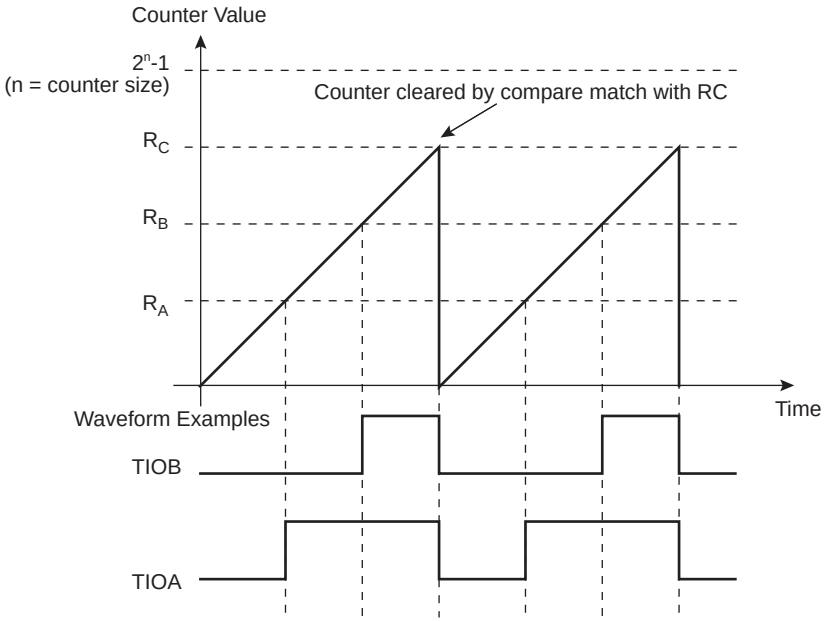
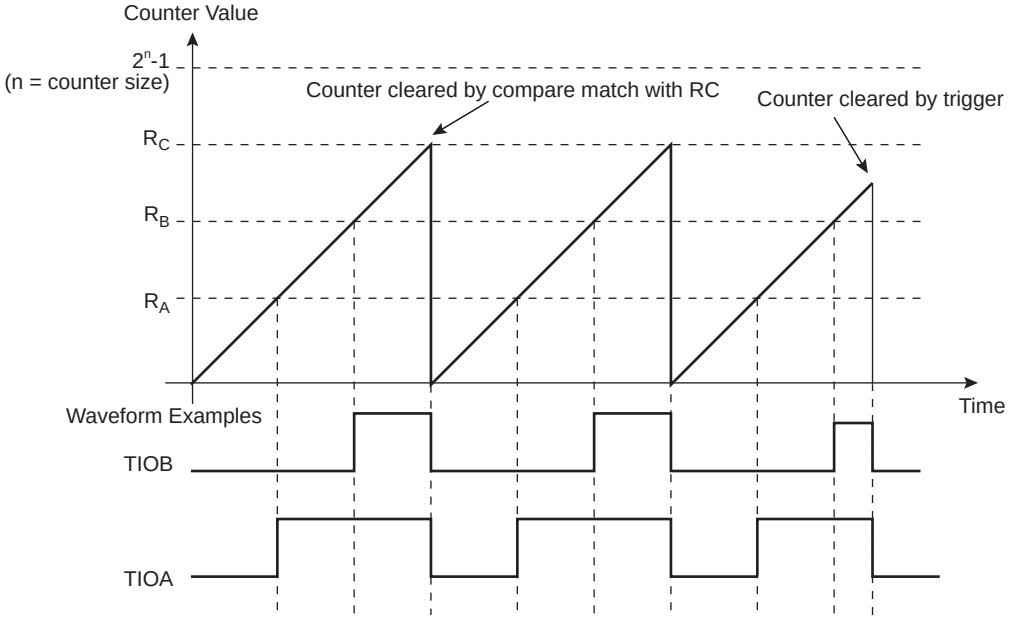


Figure 31-11. WAVSEL = 10 with Trigger



31.6.12.3 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 re-incremented to 2^{16}-1 and so on. See Figure 31-12.

A trigger such as an external event or a software trigger can modify TC_CV at any time. If a trigger occurs while TC_CV is incrementing, TC_CV then decrements. If a trigger is received while TC_CV is decrementing, TC_CV then increments. See Figure 31-13.

RC Compare cannot be programmed to generate a trigger in this configuration.

At the same time, RC Compare can stop the counter clock (CPCSTOP = 1) and/or disable the counter clock (CPCDIS = 1).

Figure 31-12. WAVSEL = 01 without Trigger

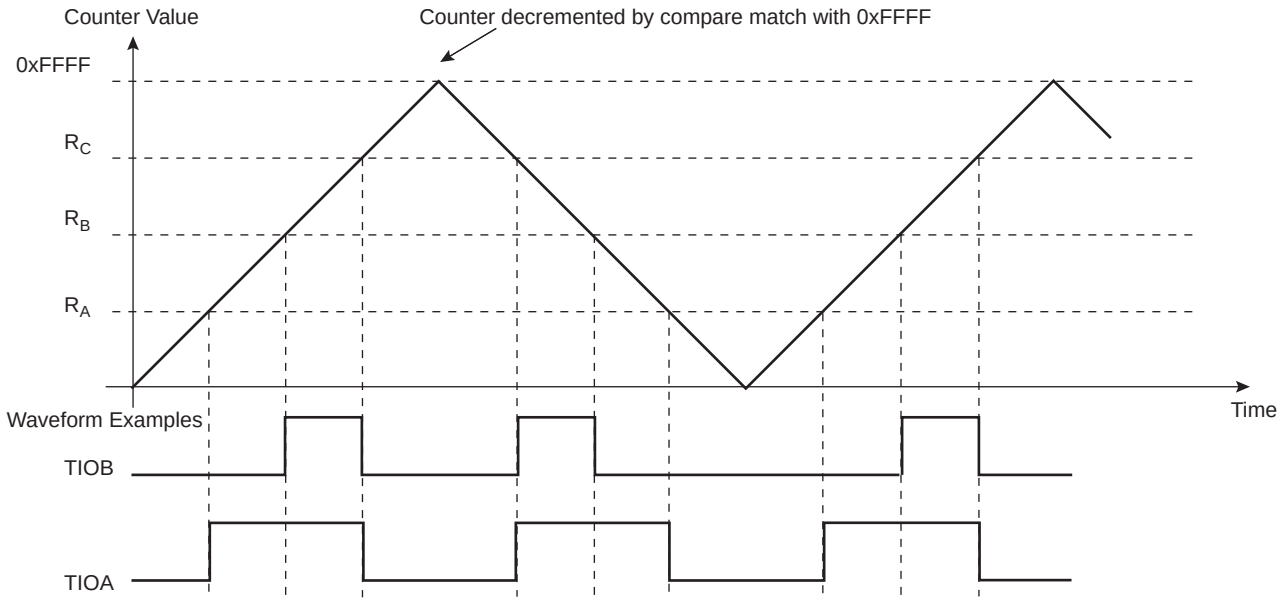
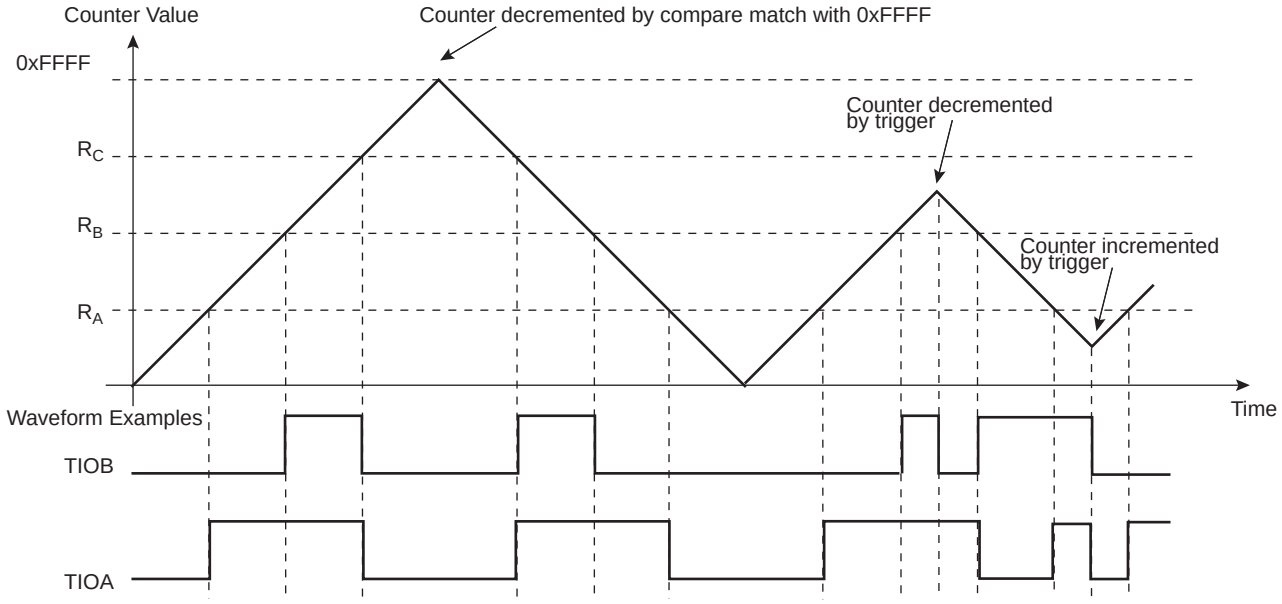


Figure 31-13. WAVSEL = 01 with Trigger



31.6.12.4 WAVSEL = 11

When WAVSEL = 11, the value of TC_CV is incremented from 0 to RC. Once RC is reached, the value of TC_CV is decremented to 0, then re-incremented to RC and so on. See Figure 31-14.

A trigger such as an external event or a software trigger can modify TC_CV at any time. If a trigger occurs while TC_CV is incrementing, TC_CV then decrements. If a trigger is received while TC_CV is decrementing, TC_CV then increments. See Figure 31-15.

RC Compare can stop the counter clock (CPCSTOP = 1) and/or disable the counter clock (CPCDIS = 1).

Figure 31-14. WAVSEL = 11 without Trigger

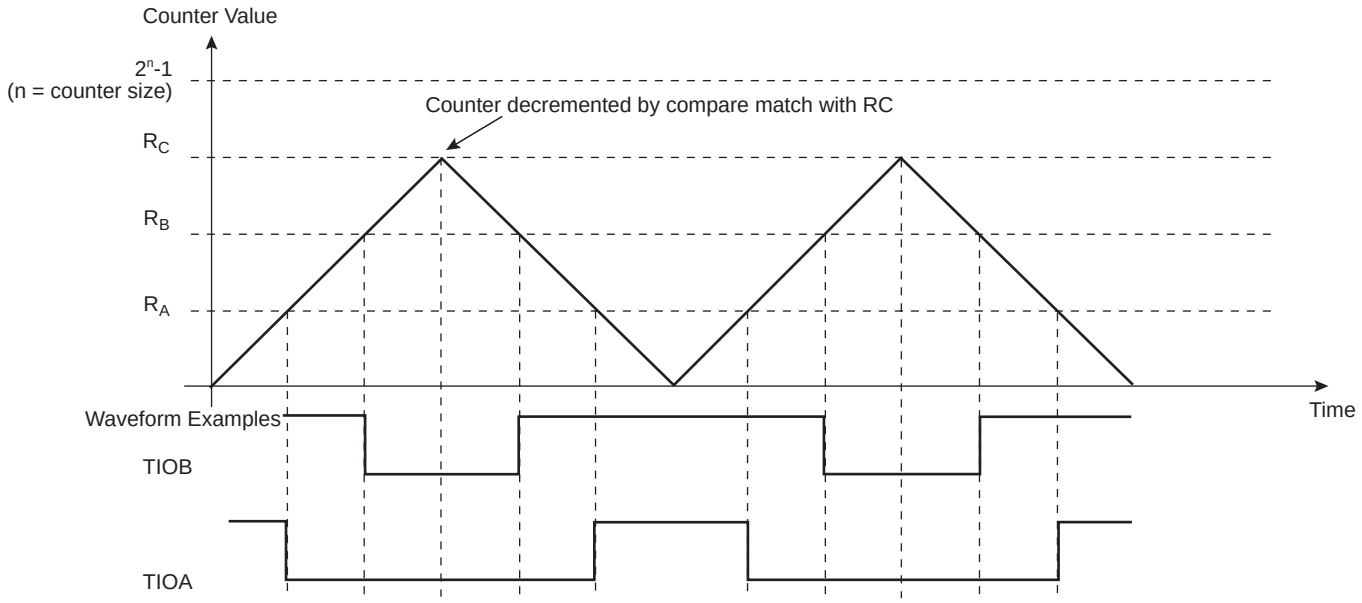
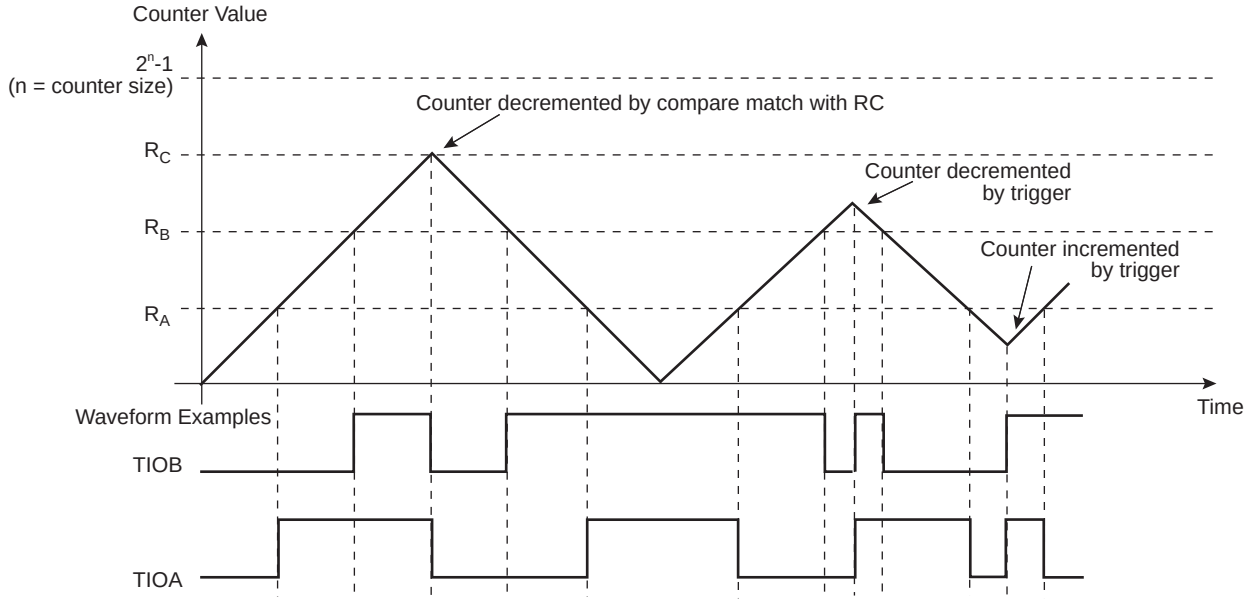


Figure 31-15. WAVSEL = 11 with Trigger



31.6.13 External Event/Trigger Conditions

An external event can be programmed to be detected on one of the clock sources (XC0, XC1, XC2) or TIOB. The external event selected can then be used as a trigger.

The EEVT parameter in TC_CMR selects the external trigger. The EEVTEDG parameter defines the trigger edge for each of the possible external triggers (rising, falling or both). If EEVTEDG is cleared (none), no external event is defined.

If TIOB is defined as an external event signal (EEVT = 0), TIOB is no longer used as an output and the compare register B is not used to generate waveforms and subsequently no IRQs. In this case the TC channel can only generate a waveform on TIOA.

When an external event is defined, it can be used as a trigger by setting bit ENETRG in the TC_CMR.

As in Capture Mode, the SYNC signal and the software trigger are also available as triggers. RC Compare can also be used as a trigger depending on the parameter WAVSEL.

31.6.14 Output Controller

The output controller defines the output level changes on TIOA and TIOB following an event. TIOB control is used only if TIOB is defined as output (not as an external event).

The following events control TIOA and TIOB: software trigger, external event and RC compare. RA compare controls TIOA and RB compare controls TIOB. Each of these events can be programmed to set, clear or toggle the output as defined in the corresponding parameter in TC_CMR.

31.6.15 2-bit Gray Up/Down Counter for Stepper Motor

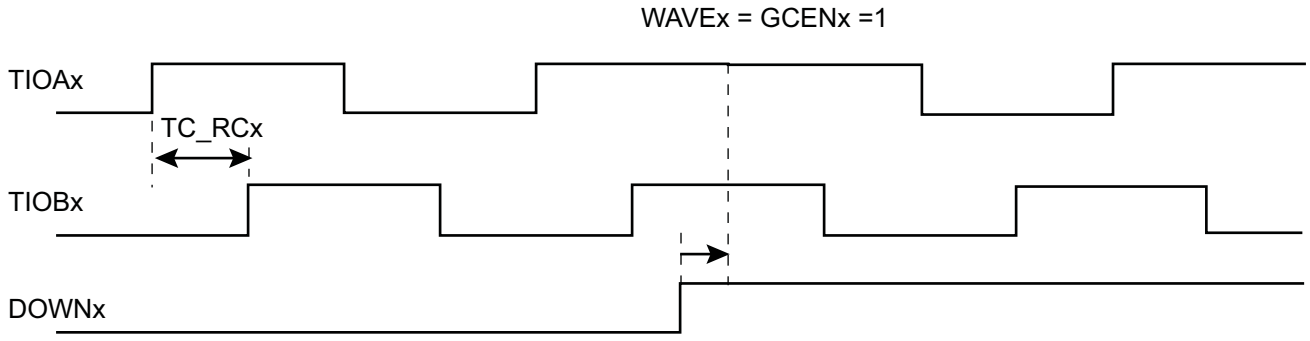
Each channel can be independently configured to generate a 2-bit gray count waveform on corresponding TIOA, TIOB outputs by means of the GCEN bit in TC_SMMRx.

Up or Down count can be defined by writing bit DOWN in TC_SMMRx.

It is mandatory to configure the channel in WAVE mode in the TC_CMR.

The period of the counters can be programmed in TC_RCx.

Figure 31-16. 2-bit Gray Up/Down Counter



31.6.16 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 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

31.7 Timer Counter (TC) User Interface

Table 31-4. 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
0xC8–0xD4	Reserved	–	–	–
0xD8	Reserved	–	–	–
0xE4	Write Protection Mode Register	TC_WPMR	Read/Write	0
0xE8–0xFC	Reserved	–	–	–
0x100–0x1A4	Reserved for PDC Registers	–	–	–

Notes: 1. Channel index ranges from 0 to 2.
2. Read-only if WAVE = 0

31.7.1 TC Channel Control Register

Name: TC_CCRx [x=0..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.

31.7.2 TC Channel Mode Register: Capture Mode

Name: TC_CMRx [x=0..2] (WAVE = 0)

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–

23	22	21	20	19	18	17	16
–	SBSMPLR			LDRB		LDRA	

15	14	13	12	11	10	9	8
WAVE	CPCTRG	–	–	–	ABETRG	ETRGEDG	

7	6	5	4	3	2	1	0
LDBDIS	LDBSTOP	BURST		CLKI	TCCLKS		

This register can only be written if the WPEN bit is cleared in the TC Write Protection Mode Register.

- **TCCLKS: Clock Selection**

Value	Name	Description
0	TIMER_CLOCK1	Clock selected: internal TIMER_CLOCK1 clock signal (from PMC)
1	TIMER_CLOCK2	Clock selected: internal TIMER_CLOCK2 clock signal (from PMC)
2	TIMER_CLOCK3	Clock selected: internal TIMER_CLOCK3 clock signal (from PMC)
3	TIMER_CLOCK4	Clock selected: internal TIMER_CLOCK4 clock signal (from PMC)
4	TIMER_CLOCK5	Clock selected: internal TIMER_CLOCK5 clock signal (from PMC)
5	XC0	Clock selected: XC0
6	XC1	Clock selected: XC1
7	XC2	Clock selected: XC2

- **CLKI: Clock Invert**

0: Counter is incremented on rising edge of the clock.

1: Counter is incremented on falling edge of the clock.

- **BURST: Burst Signal Selection**

Value	Name	Description
0	NONE	The clock is not gated by an external signal.
1	XC0	XC0 is ANDed with the selected clock.
2	XC1	XC1 is ANDed with the selected clock.
3	XC2	XC2 is ANDed with the selected clock.

- **LDBSTOP: Counter Clock Stopped with RB Loading**

0: Counter clock is not stopped when RB loading occurs.

1: Counter clock is stopped when RB loading occurs.

- **LDBDIS: Counter Clock Disable with RB Loading**

0: Counter clock is not disabled when RB loading occurs.

1: Counter clock is disabled when RB loading occurs.

- **ETRGEDG: External Trigger Edge Selection**

Value	Name	Description
0	NONE	The clock is not gated by an external signal.
1	RISING	Rising edge
2	FALLING	Falling edge
3	EDGE	Each edge

- **ABETRG: TIOA or TIOB External Trigger Selection**

0: TIOB is used as an external trigger.

1: TIOA is used as an external trigger.

- **CPCTRG: RC Compare Trigger Enable**

0: RC Compare has no effect on the counter and its clock.

1: RC Compare resets the counter and starts the counter clock.

- **WAVE: Waveform Mode**

0: Capture Mode is enabled.

1: Capture Mode is disabled (Waveform Mode is enabled).

- **LDRA: RA Loading Edge Selection**

Value	Name	Description
0	NONE	None
1	RISING	Rising edge of TIOA
2	FALLING	Falling edge of TIOA
3	EDGE	Each edge of TIOA

- **LDRB: RB Loading Edge Selection**

Value	Name	Description
0	NONE	None
1	RISING	Rising edge of TIOA
2	FALLING	Falling edge of TIOA
3	EDGE	Each edge of TIOA

- **SBSMPLR: Loading Edge Subsampling Ratio**

Value	Name	Description
0	ONE	Load a Capture Register each selected edge
1	HALF	Load a Capture Register every 2 selected edges
2	FOURTH	Load a Capture Register every 4 selected edges
3	EIGHTH	Load a Capture Register every 8 selected edges
4	SIXTEENTH	Load a Capture Register every 16 selected edges

31.7.3 TC Channel Mode Register: Waveform Mode

Name: TC_CMRx [x=0..2] (WAVE = 1)

Access: Read/Write

31	30	29	28	27	26	25	24
BSWTRG		BEEVT		BCPC		BCPB	
23	22	21	20	19	18	17	16
ASWTRG		AEEVT		ACPC		ACPA	
15	14	13	12	11	10	9	8
WAVE	WAVSEL		ENETRG	EEVT		EEVTEDG	
7	6	5	4	3	2	1	0
CPCDIS	CPCSTOP	BURST		CLKI	TCCLKS		

This register can only be written if the WPEN bit is cleared in the TC Write Protection Mode Register.

- **TCCLKS: Clock Selection**

Value	Name	Description
0	TIMER_CLOCK1	Clock selected: internal TIMER_CLOCK1 clock signal (from PMC)
1	TIMER_CLOCK2	Clock selected: internal TIMER_CLOCK2 clock signal (from PMC)
2	TIMER_CLOCK3	Clock selected: internal TIMER_CLOCK3 clock signal (from PMC)
3	TIMER_CLOCK4	Clock selected: internal TIMER_CLOCK4 clock signal (from PMC)
4	TIMER_CLOCK5	Clock selected: internal TIMER_CLOCK5 clock signal (from PMC)
5	XC0	Clock selected: XC0
6	XC1	Clock selected: XC1
7	XC2	Clock selected: XC2

- **CLKI: Clock Invert**

0: Counter is incremented on rising edge of the clock.

1: Counter is incremented on falling edge of the clock.

- **BURST: Burst Signal Selection**

Value	Name	Description
0	NONE	The clock is not gated by an external signal.
1	XC0	XC0 is ANDed with the selected clock.
2	XC1	XC1 is ANDed with the selected clock.
3	XC2	XC2 is ANDed with the selected clock.

- **CPCSTOP: Counter Clock Stopped with RC Compare**

0: Counter clock is not stopped when counter reaches RC.

1: Counter clock is stopped when counter reaches RC.

- **CPCDIS: Counter Clock Disable with RC Compare**

0: Counter clock is not disabled when counter reaches RC.

1: Counter clock is disabled when counter reaches RC.

- **EEVTEDG: External Event Edge Selection**

Value	Name	Description
0	NONE	None
1	RISING	Rising edge
2	FALLING	Falling edge
3	EDGE	Each edge

- **EEVT: External Event Selection**

Signal selected as external event.

Value	Name	Description	TIOB Direction
0	TIOB	TIOB(1)	Input
1	XC0	XC0	Output
2	XC1	XC1	Output
3	XC2	XC2	Output

Note: 1. *If TIOB is chosen as the external event signal, it is configured as an input and no longer generates waveforms and subsequently no IRQs*.

- **ENETRG: External Event Trigger Enable**

0: The external event has no effect on the counter and its clock.

1: The external event resets the counter and starts the counter clock.

Note: Whatever the value programmed in ENETRG, the selected external event only controls the TIOA output and TIOB if not used as input (trigger event input or other input used).

- **WAVSEL: Waveform Selection**

Value	Name	Description
0	UP	UP mode without automatic trigger on RC Compare
1	UPDOWN	UPDOWN mode without automatic trigger on RC Compare
2	UP_RC	UP mode with automatic trigger on RC Compare
3	UPDOWN_RC	UPDOWN mode with automatic trigger on RC Compare

- **WAVE: Waveform Mode**

0: Waveform Mode is disabled (Capture Mode is enabled).

1: Waveform Mode is enabled.

- **ACPA: RA Compare Effect on TIOA**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **ACPC: RC Compare Effect on TIOA**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **AEEVT: External Event Effect on TIOA**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **ASWTRG: Software Trigger Effect on TIOA**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **BCPB: RB Compare Effect on TIOB**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **BCPC: RC Compare Effect on TIOB**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **BEEVT: External Event Effect on TIOB**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **BSWTRG: Software Trigger Effect on TIOB**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

31.7.4 TC Stepper Motor Mode Register

Name: TC_SMMRx [x=0..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.

31.7.5 TC Counter Value Register

Name: TC_CVx [x=0..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.

31.7.6 TC Register A

Name: TC_RAx [x=0..2]

Access: Read-only if WAVE = 0, Read/Write if WAVE = 1

31	30	29	28	27	26	25	24
RA							

23	22	21	20	19	18	17	16
RA							

15	14	13	12	11	10	9	8
RA							

7	6	5	4	3	2	1	0
RA							

This register can only be written if the WPEN bit is cleared in the TC Write Protection Mode Register.

- **RA: Register A**

RA contains the Register A value in real time.

31.7.7 TC Register B

Name: TC_RBx [x=0..2]

Access: Read-only if WAVE = 0, Read/Write if WAVE = 1

31	30	29	28	27	26	25	24
RB							

23	22	21	20	19	18	17	16
RB							

15	14	13	12	11	10	9	8
RB							

7	6	5	4	3	2	1	0
RB							

This register can only be written if the WPEN bit is cleared in the TC Write Protection Mode Register.

- **RB: Register B**

RB contains the Register B value in real time.

31.7.8 TC Register C

Name: TC_RCx [x=0..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.

31.7.9 TC Status Register

Name: TC_SRx [x=0..2]

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	MTIOB	MTIOA	CLKSTA
15	14	13	12	11	10	9	8
–	–	–	–	–	–	RXBUFF	ENDRX
7	6	5	4	3	2	1	0
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow Status**

0: No counter overflow has occurred since the last read of the Status Register.

1: A counter overflow has occurred since the last read of the Status Register.

- **LOVRS: Load Overrun Status**

0: Load overrun has not occurred since the last read of the Status Register or WAVE = 1.

1: RA or RB have been loaded at least twice without any read of the corresponding register since the last read of the Status Register, if WAVE = 0.

- **CPAS: RA Compare Status**

0: RA Compare has not occurred since the last read of the Status Register or WAVE = 0.

1: RA Compare has occurred since the last read of the Status Register, if WAVE = 1.

- **CPBS: RB Compare Status**

0: RB Compare has not occurred since the last read of the Status Register or WAVE = 0.

1: RB Compare has occurred since the last read of the Status Register, if WAVE = 1.

- **CPCS: RC Compare Status**

0: RC Compare has not occurred since the last read of the Status Register.

1: RC Compare has occurred since the last read of the Status Register.

- **LDRAS: RA Loading Status**

0: RA Load has not occurred since the last read of the Status Register or WAVE = 1.

1: RA Load has occurred since the last read of the Status Register, if WAVE = 0.

- **LDRBS: RB Loading Status**

0: RB Load has not occurred since the last read of the Status Register or WAVE = 1.

1: RB Load has occurred since the last read of the Status Register, if WAVE = 0.

- **ETRGS: External Trigger Status**

0: External trigger has not occurred since the last read of the Status Register.

1: External trigger has occurred since the last read of the Status Register.

- **ENDRX: End of Receiver Transfer**

0: The Receive End of Transfer signal from the PDC channel is inactive.

1: The Receive End of Transfer signal from the PDC channel is active.

- **RXBUFF: Reception Buffer Full**

0: The Receive Buffer Full signal from the PDC channel is inactive.

1: The Receive Buffer Full signal from the PDC channel is active.

- **CLKSTA: Clock Enabling Status**

0: Clock is disabled.

1: Clock is enabled.

- **MTIOA: TIOA Mirror**

0: TIOA is low. If WAVE = 0, this means that TIOA pin is low. If WAVE = 1, this means that TIOA is driven low.

1: TIOA is high. If WAVE = 0, this means that TIOA pin is high. If WAVE = 1, this means that TIOA is driven high.

- **MTIOB: TIOB Mirror**

0: TIOB is low. If WAVE = 0, this means that TIOB pin is low. If WAVE = 1, this means that TIOB is driven low.

1: TIOB is high. If WAVE = 0, this means that TIOB pin is high. If WAVE = 1, this means that TIOB is driven high.

31.7.10 TC Interrupt Enable Register

Name: TC_IERx [x=0..2]

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	RXBUFF	ENDRX
7	6	5	4	3	2	1	0
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow**

0: No effect.

1: Enables the Counter Overflow Interrupt.

- **LOVRS: Load Overrun**

0: No effect.

1: Enables the Load Overrun Interrupt.

- **CPAS: RA Compare**

0: No effect.

1: Enables the RA Compare Interrupt.

- **CPBS: RB Compare**

0: No effect.

1: Enables the RB Compare Interrupt.

- **CPCS: RC Compare**

0: No effect.

1: Enables the RC Compare Interrupt.

- **LDRAS: RA Loading**

0: No effect.

1: Enables the RA Load Interrupt.

- **LDRBS: RB Loading**

0: No effect.

1: Enables the RB Load Interrupt.

- **ETRGS: External Trigger**

0: No effect.

1: Enables the External Trigger Interrupt.

- **ENDRX: End of Receiver Transfer**

0: No effect.

1: Enables the PDC Receive End of Transfer Interrupt.

- **RXBUFF: Reception Buffer Full**

0: No effect.

1: Enables the PDC Receive Buffer Full Interrupt.

31.7.11 TC Interrupt Disable Register

Name: TC_IDRx [x=0..2]

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	RXBUFF	ENDRX
7	6	5	4	3	2	1	0
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow**

0: No effect.

1: Disables the Counter Overflow Interrupt.

- **LOVRS: Load Overrun**

0: No effect.

1: Disables the Load Overrun Interrupt (if WAVE = 0).

- **CPAS: RA Compare**

0: No effect.

1: Disables the RA Compare Interrupt (if WAVE = 1).

- **CPBS: RB Compare**

0: No effect.

1: Disables the RB Compare Interrupt (if WAVE = 1).

- **CPCS: RC Compare**

0: No effect.

1: Disables the RC Compare Interrupt.

- **LDRAS: RA Loading**

0: No effect.

1: Disables the RA Load Interrupt (if WAVE = 0).

- **LDRBS: RB Loading**

0: No effect.

1: Disables the RB Load Interrupt (if WAVE = 0).

- **ETRGS: External Trigger**

0: No effect.

1: Disables the External Trigger Interrupt.

- **ENDRX: End of Receiver Transfer**

0: No effect.

1: Disables the PDC Receive End of Transfer Interrupt.

- **RXBUFF: Reception Buffer Full**

0: No effect.

1: Disables the PDC Receive Buffer Full Interrupt.

31.7.12 TC Interrupt Mask Register

Name: TC_IMRx [x=0..2]

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	RXBUFF	ENDRX
7	6	5	4	3	2	1	0
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow**

0: The Counter Overflow Interrupt is disabled.

1: The Counter Overflow Interrupt is enabled.

- **LOVRS: Load Overrun**

0: The Load Overrun Interrupt is disabled.

1: The Load Overrun Interrupt is enabled.

- **CPAS: RA Compare**

0: The RA Compare Interrupt is disabled.

1: The RA Compare Interrupt is enabled.

- **CPBS: RB Compare**

0: The RB Compare Interrupt is disabled.

1: The RB Compare Interrupt is enabled.

- **CPCS: RC Compare**

0: The RC Compare Interrupt is disabled.

1: The RC Compare Interrupt is enabled.

- **LDRAS: RA Loading**

0: The Load RA Interrupt is disabled.

1: The Load RA Interrupt is enabled.

- **LDRBS: RB Loading**

0: The Load RB Interrupt is disabled.

1: The Load RB Interrupt is enabled.

- **ETRGS: External Trigger**

0: The External Trigger Interrupt is disabled.

1: The External Trigger Interrupt is enabled.

- **ENDRX: End of Receiver Transfer**

0: The PDC Receive End of Transfer Interrupt is disabled.

1: The PDC Receive End of Transfer Interrupt is enabled.

- **RXBUFF: Reception Buffer Full**

0: The PDC Receive Buffer Full Interrupt is disabled.

1: The PDC Receive Buffer Full Interrupt is enabled.

31.7.13 TC Block Control Register

Name: TC_BCR

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.

31.7.14 TC Block Mode Register

Name: TC_BMR

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
–	–	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

31.7.15 TC Write Protection Mode Register

Name: TC_WPMR

Access: Read/Write

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

- **WPEN: 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).

See Section 31.6.16 “Register Write Protection” for the list of registers that can be write-protected.

- **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.

32. Analog-to-Digital Converter (ADC)

32.1 Description

The ADC is based on a 12-bit Analog-to-Digital Converter (ADC) managed by an ADC Controller. Refer to Figure 32-1, "Analog-to-Digital Converter Block Diagram". It also integrates a 8-to-1 analog multiplexer, making possible the analog-to-digital conversions of 8 analog lines. The conversions extend from 0V to VDDANA.

The ADC supports the 10-bit or 12-bit resolution mode. The 10-bit resolution mode prevents from using 16-bit Peripheral DMA transfer into memory when only 8-bit resolution is required by the application. Please 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, 4 samples are used, which gives an effective sample rate of 1/4 of the actual sample frequency. For 12-bitmode, 16 samples are used, giving an effective sample rate of 1/16 of the actual sample frequency. This arrangement allows conversion speed to be traded for better accuracy.

Software trigger, external trigger on rising edge of the ADTRG pin or internal triggers from Timer Counter output(s) are configurable.

The last Channel can be converted at a rate different from other channels to improve conversion and processing efficiency in case of a device which provides very low frequency variations such as a temperature sensor. A dedicated comparison circuitry on the last channel allows specific processing and interrupt.

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 being 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.

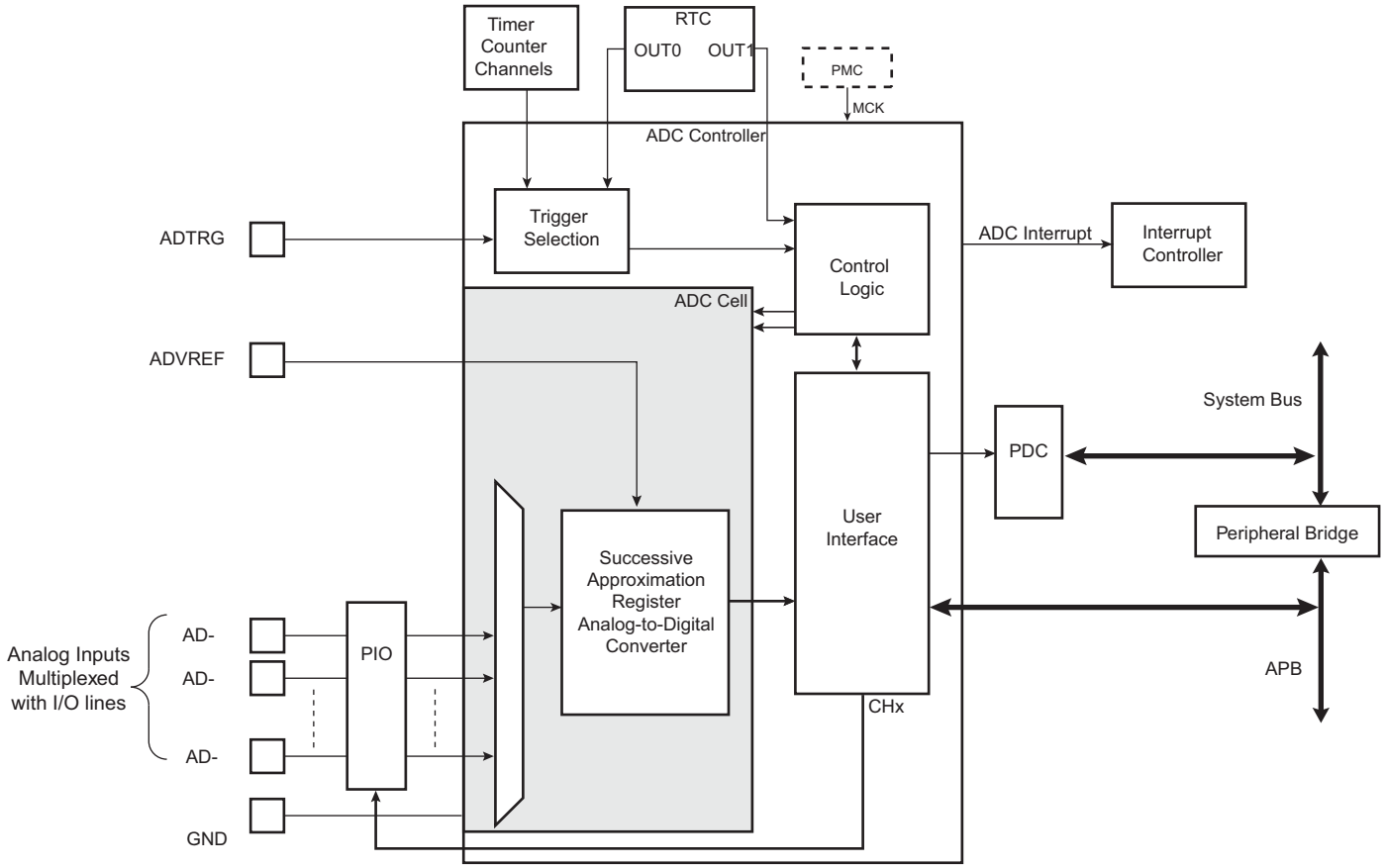
Finally, the user can configure ADC timings, such as Startup Time and Tracking Time.

32.2 Embedded Characteristics

- 12-bit Resolution with Enhanced Mode up to 12-bit
- 800 k Hz Conversion Rate
- Digital Averaging Function providing Enhanced Resolution Mode up to 12-bit
- Wide Range Power Supply Operation
- 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)
- Up to 2 Trigger Events Having Independent Rates
- 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

32.3 Block Diagram

Figure 32-1. Analog-to-Digital Converter Block Diagram



32.4 Signal Description

Table 32-1. ADC Pin Description

Pin Name	Description
AD0–AD7	Analog input channels
ADTRG	External trigger

32.5 Product Dependencies

32.5.1 Power Management

The ADC Controller is not continuously clocked. The programmer must first enable the ADC Controller MCK in the Power Management Controller (PMC) before using the ADC Controller. However, if the application does not require ADC operations, the ADC Controller clock can be stopped when not needed and restarted when necessary. Configuring the ADC Controller does not require the ADC Controller clock to be enabled.

32.5.2 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 32-2. Peripheral IDs

Instance	ID
ADC	29

32.5.3 Analog Inputs

The analog input pins can be multiplexed with PIO lines. In this case, the assignment of the ADC input is automatically done as soon as the corresponding channel is enabled by writing the register ADC_CHER. By default, after reset, the PIO line is configured as input with its pull-up enabled and the ADC input is connected to the GND.

32.5.4 I/O Lines

The pin ADTRG may be shared with other peripheral functions through the PIO Controller. In this case, the PIO Controller should be set accordingly to assign the pin ADTRG to the ADC function.

Table 32-3. I/O Lines

Instance	Signal	I/O Line	Peripheral
ADC	ADTRG	PA8	B
ADC	AD0	PA17	X1
ADC	AD1	PA18	X1
ADC	AD2	PA19	X1
ADC	AD3	PA20	X1
ADC	AD4	PB0	X1
ADC	AD5	PB1	X1
ADC	AD6/WKUP12	PB2	X1
ADC	AD7/WKUP13	PB3	X1

32.5.5 Timer Triggers

Timer Counters may or may not be used as hardware triggers depending on user requirements. Thus, some or all of the timer counters may be unconnected.

32.5.6 Conversion Performances

For performance and electrical characteristics of the ADC, see the product electrical characteristics.

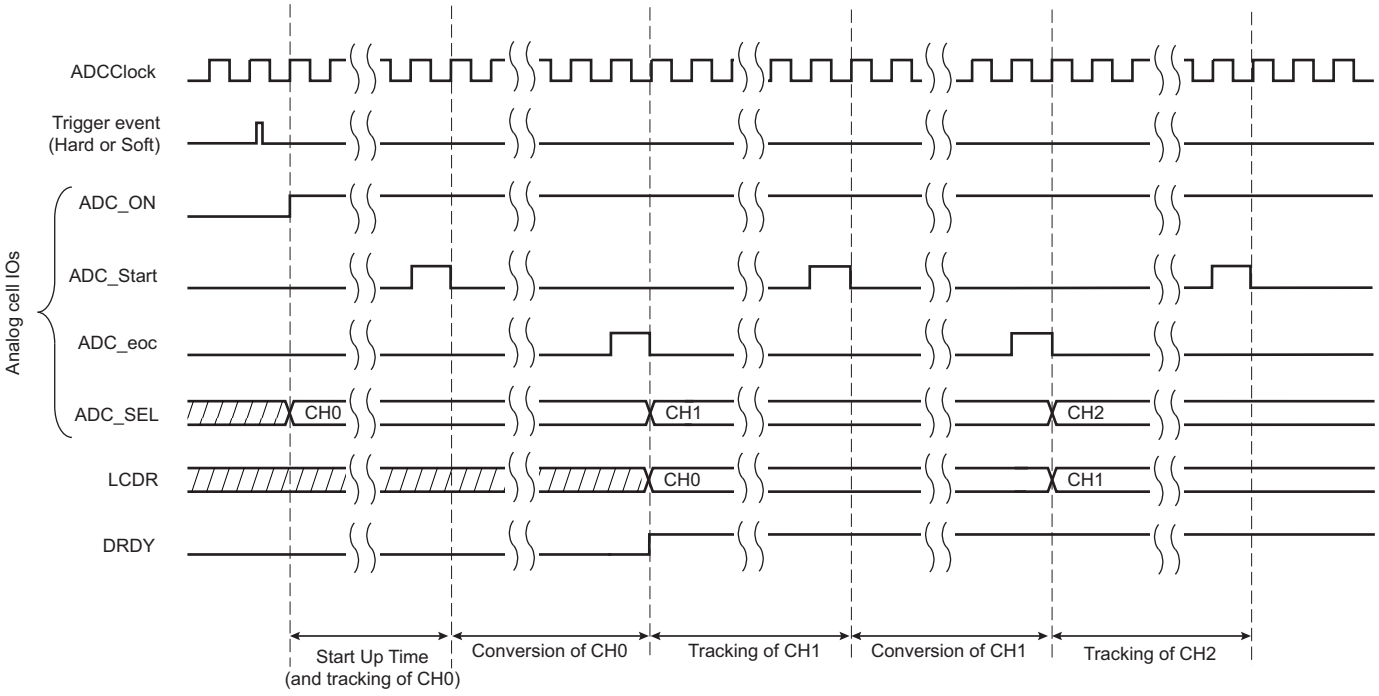
32.6 Functional Description

32.6.1 Analog-to-Digital Conversion

The ADC uses the ADC Clock to perform conversions. Converting a single analog value to a 12-bit digital data requires Tracking Clock cycles as defined in the field TRACKTIM of the “ADC Mode Register” . The ADC Clock frequency is selected in the PRESCAL field of the Mode Register (ADC_MR) unless DIV1 is written to 1 or DIV3 is written to 1. Bits DIV1 and DIV3 allow better prescaler granularity. DIV1 and DIV3 must not both be configured to 1. The tracking phase starts during the conversion of the previous channel. If the tracking time is longer than the conversion time, the tracking phase is extended to the end of the previous conversion.

The ADC clock range is between MCKif DIV1 is 1, and MCK/512, if PRESCAL is set to 255 (0xFF). PRESCAL must be programmed in order to provide an ADC clock frequency according to the parameters given in the product “Electrical Characteristics” section.

Figure 32-2. Sequence of ADC Conversions



32.6.2 Conversion Reference

The conversion is performed on a full range between 0V and the reference voltage connected to VDDANA. Analog inputs between these voltages convert to values based on a linear conversion.

32.6.3 Conversion Resolution

The ADC supports 10-bit or 12-bit resolutions. The 10-bit selection is performed by setting the LOWRES bit in the ADC Mode Register (ADC_MR). By default, after a reset, the resolution is the highest and the DATA field in the data registers is fully used. By setting the LOWRES bit, the ADC switches to the lowest resolution and the conversion results can be read in the lowest significant bits of the data registers. The two highest bits of the DATA field in the corresponding ADC_CDR and of the LDATA field in the ADC_LCDR read 0.

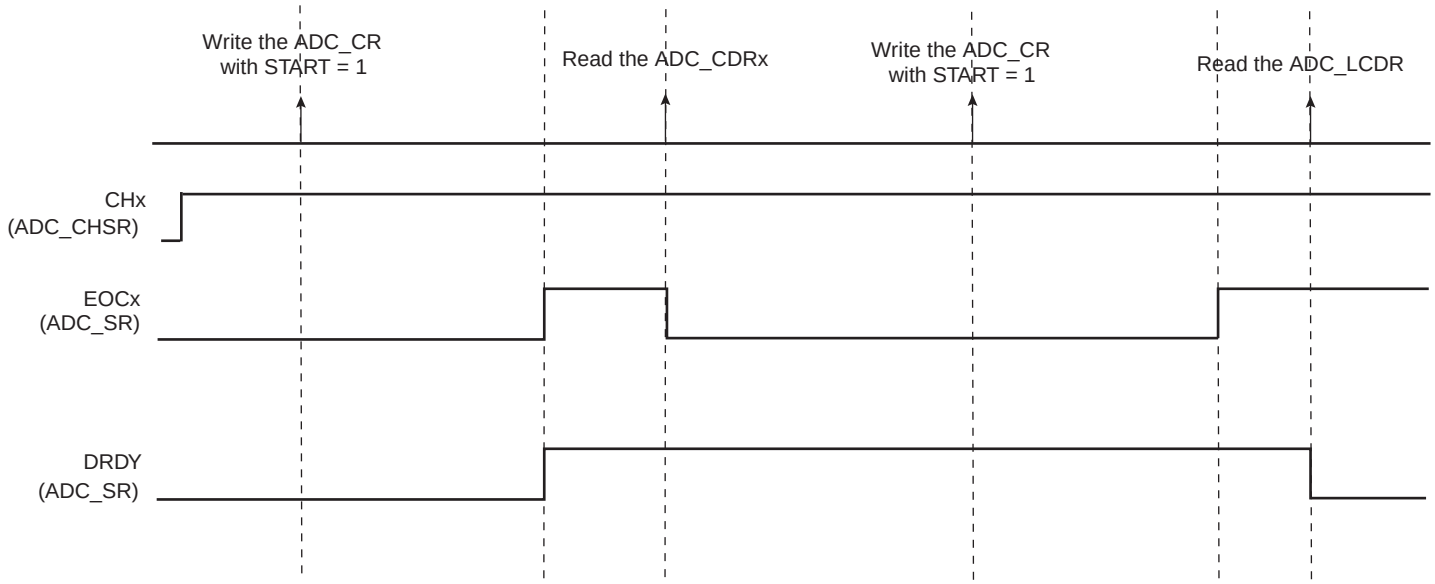
32.6.4 Conversion Results

When a conversion is completed, the resulting 12-bit digital value is stored in the Channel Data Register (ADC_CDRx) of the current channel and in the ADC Last Converted Data Register (ADC_LCDR). By setting the TAG option in the ADC_EMR, the ADC_LCDR presents the channel number associated to the last converted data in the CHNB field.

The channel EOC bit in the Status Register (ADC_SR) is set and the DRDY is set. In the case of a connected PDC channel, DRDY rising triggers a data request. In any case, either EOC and DRDY can trigger an interrupt.

Reading one of the Channel Data registers (ADC_CDRx) clears the corresponding EOC bit. Reading the ADC_LCDR clears the DRDY bit.

Figure 32-3. EOCx and DRDY Flag Behavior

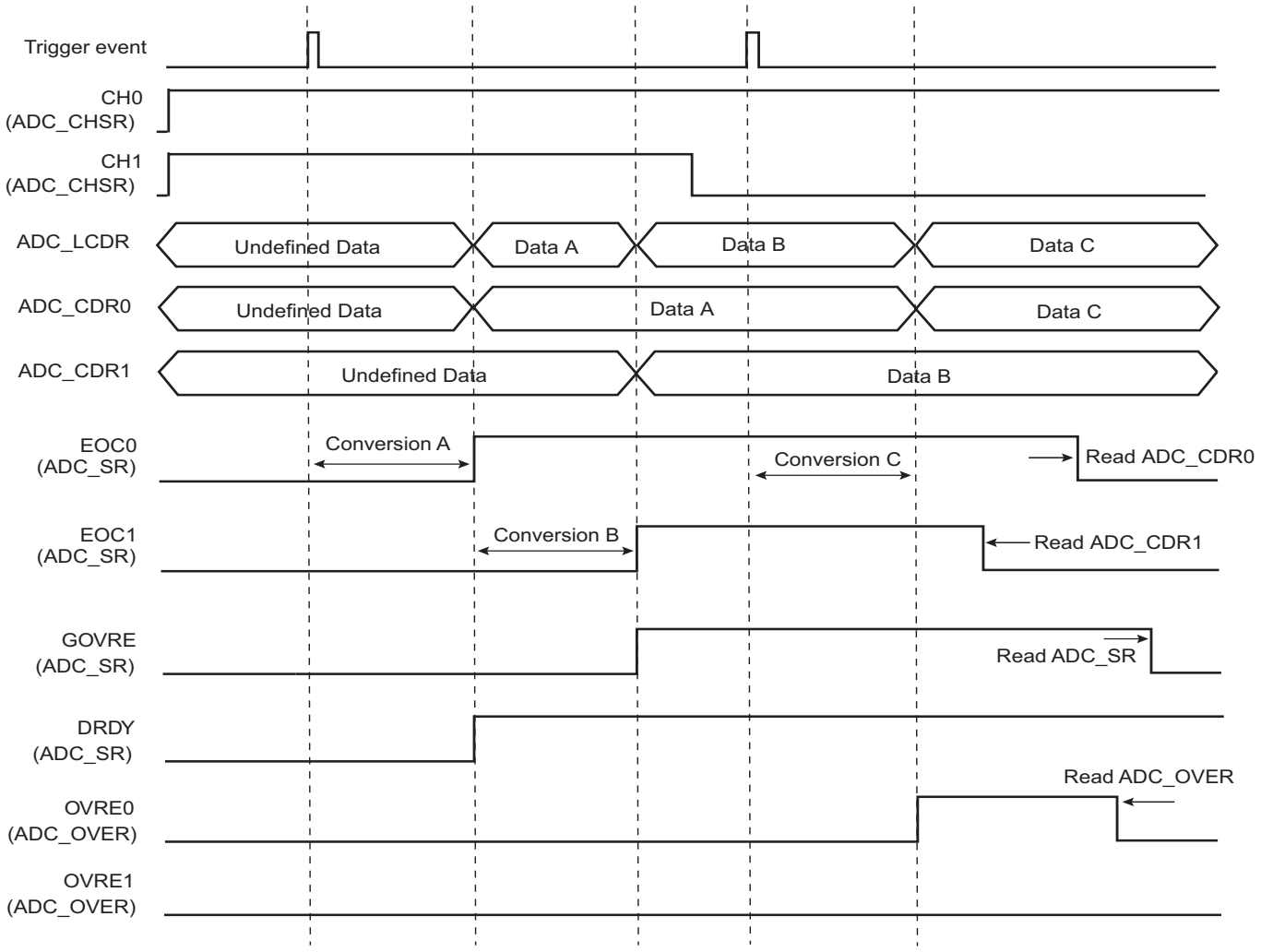


If the ADC_CDR is not read before further incoming data is converted, the corresponding Overrun Error (OVREx) flag is set in the Overrun Status Register (ADC_OVER).

Likewise, new data converted when DRDY is high sets the GOVRE bit (General Overrun Error) in ADC_SR.

The OVREx flag is automatically cleared when ADC_OVER is read, and GOVRE flag is automatically cleared when ADC_SR is read.

Figure 32-4. GOVRE and OVREx Flag Behavior



Warning: If the corresponding channel is disabled during a conversion or if it is disabled and then reenabled during a conversion, its associated data and its corresponding EOC and OVRE flags in ADC_SR are unpredictable.

32.6.5 Conversion Triggers

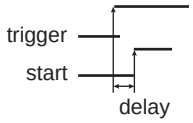
Conversions of the active analog channels are started with a software or hardware trigger. The software trigger is provided by writing the Control Register (ADC_CR) with the START bit at 1.

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 the ADC Mode Register (ADC_MR). The selected hardware trigger is enabled with the TRGEN bit in the ADC Mode Register.

The minimum time between two consecutive trigger events must be strictly greater than the duration time of the longest conversion sequence according to configuration of registers ADC_MR, ADC_CHSR, ADC_SEQRx.

If a hardware trigger is selected, the start of a conversion is triggered after a delay starting at each rising edge of the selected signal. Due to asynchronous handling, the delay may vary in a range of two MCK clock periods to on ADC clock period.

Figure 32-5. 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.

32.6.6 Sleep Mode and Conversion Sequencer

The ADC Sleep Mode maximizes power saving by automatically deactivating the ADC when it is not being used for conversions. Sleep Mode is selected by setting the SLEEP bit in the Mode Register (ADC_MR).

The Sleep mode is automatically managed by a conversion sequencer, which can automatically process the conversions of all channels at lowest power consumption.

This mode can be used when the minimum period of time between two successive trigger events is greater than the startup period of the analog-to-digital converter (see the product “ADC Characteristics” section).

When a start conversion request occurs, the ADC is automatically activated. As the analog cell requires a start-up time, the logic waits during this time and starts the conversion on the enabled channels. When all conversions are complete, the ADC is deactivated until the next trigger. Triggers occurring during the sequence are not taken into account.

A fast wake-up mode is available in the ADC_MR as a compromise between power saving strategy and responsiveness. Setting the FWUP bit enables the fast wake-up mode. In fast wake-up mode the ADC cell is not fully deactivated while no conversion is requested, thereby providing less power saving but faster wakeup.

The conversion sequencer allows automatic processing with minimum processor intervention and optimized power consumption. Conversion sequences can be performed periodically using a Timer/Counter output. The periodic acquisition of several samples can be processed automatically without any intervention of the processor thanks to the PDC.

The sequence can be customized by programming the Sequence Channel Register ADC_SEQR1 and setting the USEQ bit of the Mode Register (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 will be 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).

Note: The reference voltage pins always remain connected in normal mode as in sleep mode.

32.6.7 Comparison Window

The ADC Controller features automatic comparison functions. It compares converted values to a low threshold or a high threshold or both, according to the CMPMODE function chosen in the Extended Mode Register (ADC_EMR). The comparison can be done on all channels or only on the channel specified in CMPSEL field of ADC_EMR. To compare all channels the CMP_ALL parameter of ADC_EMR should be set.

Moreover a filtering option can be set by writing the number of consecutive comparison errors needed to raise the flag. This number can be written and read in the CMPFILTER field of ADC_EMR.

The flag can be read on the COMPE bit of the Interrupt Status Register (ADC_ISR) and can trigger an interrupt.

The High Threshold and the Low Threshold can be read/write in the Comparison Window Register (ADC_CWR).

If the comparison window is to be used with LOWRES bit set in the ADC_MR, the thresholds do not need to be adjusted as adjustment will be done internally. Whether or not the LOWRES bit is set, thresholds must always be configured in consideration of the maximum ADC resolution.

32.6.8 ADC Timings

Each ADC has its own minimal Startup Time that is programmed through the field STARTUP in the Mode Register (ADC_MR).

A minimal Tracking Time is necessary for the ADC to guarantee the best converted final value between two channel selections. This time has to be programmed through the TRACKTIM field in the 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 product “ADC Characteristics” section.

32.6.9 Last Channel Specific Measurement Trigger

The last channel (higher index available) embeds a specific mode allowing a measurement trigger period which differs from other active channels. This allows efficient management of the conversions especially if the channel is driven by a device whose variation has a totally different frequency from other converted channels (for example, but not limited to, temperature sensor).

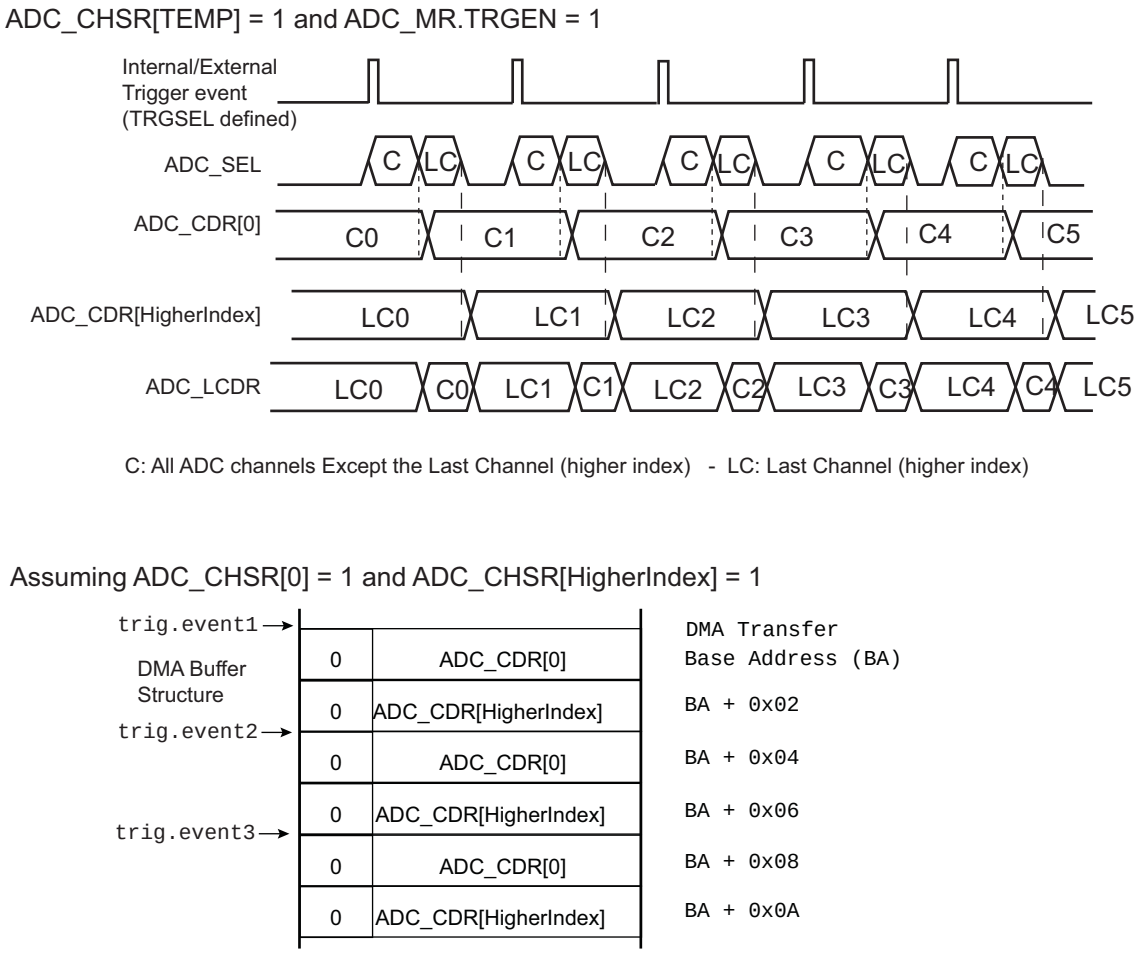
The temperature measurement can be made in different ways through the ADC controller. The different methods for the measure depend on the configuration bits TRGEN in the ADC_MR and CH7 in the ADC_CHSR.

The last channel measure can be triggered like the other channels by enabling its associated conversion channel index 7, writing 1 in CH7 of the ADC_CHER.

The manual start can only be performed if TRGEN bit in ADC_MR is disabled. When the START bit in the ADC_CR is set, the last channel conversion will be 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 the ADC_ISR.

If the TRGEN bit is set in the ADC_MR, the last channel is periodically converted together with the other enabled channels and the result is placed on the ADC_LCDR and ADC_CDR7 registers. Thus the last channel conversion result is part of the Peripheral DMA Controller buffer (see Figure 32-6).

Figure 32-6. Same Trigger for All Channels



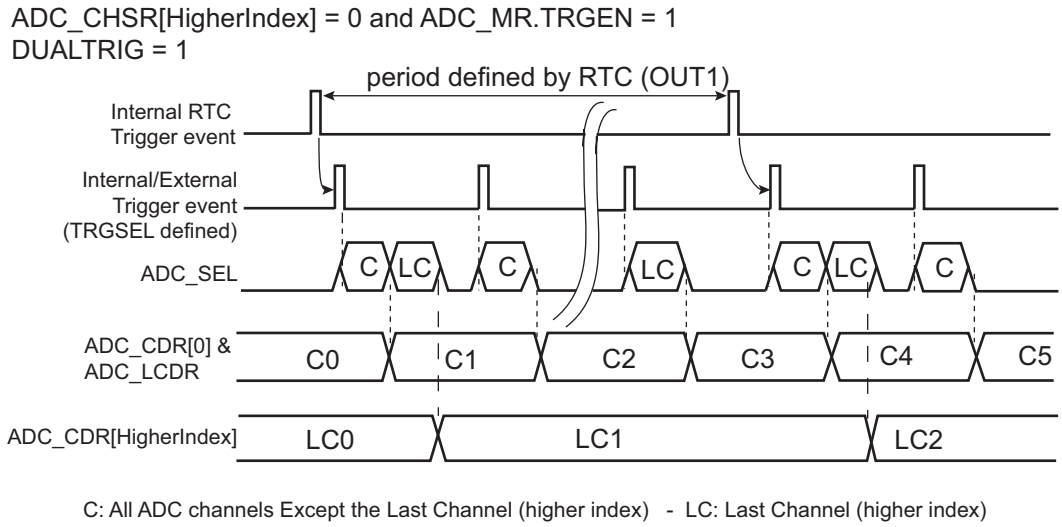
If the last channel is driven by a device having slower variation compared to other channels (temperature sensor for example) the channel can be enabled/disabled at any time but this may not be optimal for downstream processing.

The ADC controller allows a different way of triggering the measure when DUALTRIG is set in the ADC_LCTMR but CH7 is not set in the ADC_CHSR.

Under these conditions, the measure for last channel is triggered with a period defined by RTC_MR (OUT1 field) while other channels are still active. OUT1 configures an internal trigger generated by the RTC, totally independent of the internal/external triggers. The RTC event will be taken into account on the next internal/external trigger event.The internal/external trigger for other channels is selected through TRGSEL field of the ADC_MR.

Every second, a conversion is scheduled for channel 7 but the result of the conversion is only uploaded in the ADC_CDR7 register and not in the ADC_LCDR (see Figure 32-7). Therefore there is no change in the structure of the Peripheral DMA Controller buffer due to the conversion of the last channel, only the enabled channels are kept in the buffer. The end of conversion of the last channel is reported by means of EOC7 flag in the ADC_ISR.

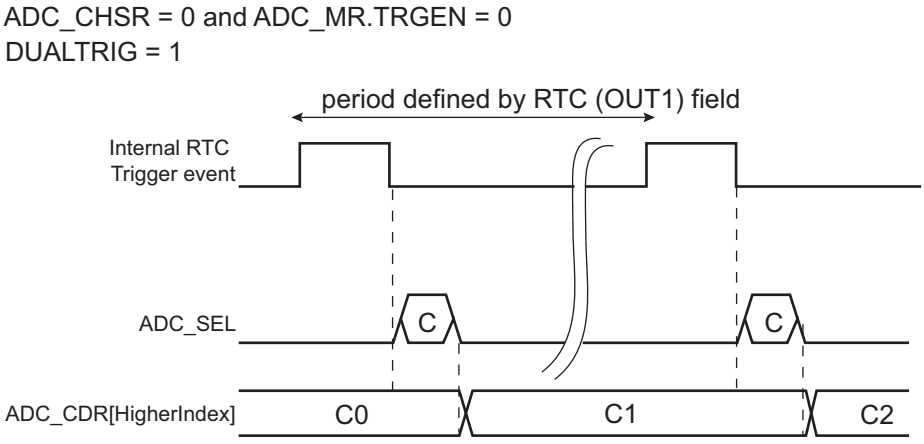
Figure 32-7. Independent Trigger Measurement for Last Channel



If DUALTRIG = 1, TRGEN is disabled and none of the channels is enabled in the ADC_CHSR (ADC_CHSR = 0), then only the channel 7 is converted at a rate defined by the RTC OUT1 field (see Figure 32-8).

This mode of operation, when combined with the sleep mode operation of the ADC controller, provides a low power mode for last channel measure (assuming there is no other ADC conversion to schedule at a high sampling rate or simply no other channel to convert).

Figure 32-8. Only Last Channel Measurement Triggered at Low Speed



32.6.10 Enhanced Resolution Mode and Digital Averaging Function

The Enhanced Resolution Mode is enabled if LOWRES is cleared in the ADC Mode Register (ADC_MR), and the OSR field is set to 1, 2 in ADC Extended Mode Register (ADC_EMR). The enhancement is based on a digital averaging function.

FREERUN must be cleared when digital averaging is used (OSR ≠ 0 in ADC_EMR).

There is no averaging on the last index channel if the measure is triggered by RTC event.

In this mode the ADC Controller will trade conversion performance for accuracy by averaging multiple samples, thus providing a digital low-pass filter function.

If 1-bit enhancement resolution is selected (OSR = 1 in the ADC_EMR), the ADC effective sample rate is the maximum ADC sample rate divided by 4, therefore the oversampling ratio is 4.

When the 2-bit enhancement resolution is selected (OSR = 2 in the ADC_EMR), the ADC effective 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 RTC event.

The average result is valid into the ADC_CDRx (x corresponding to the index of the channel) only if EOCn flag is set in the ADC_ISR and OVREn flag is cleared in the ADC_OVER. The average result for all channels is valid in the ADC_LCDR only if DRDY is set and GOVRE is cleared in the ADC_ISR.

Note that the ADC_CDRs are not buffered. Therefore, when an averaging sequence is ongoing, the value in these registers changes after each averaging sample. However, overrun flags in the ADC_OVER rise as soon as the first sample of an averaging sequence is received and thus the previous averaged value is not read (even if the new averaged value is not ready).

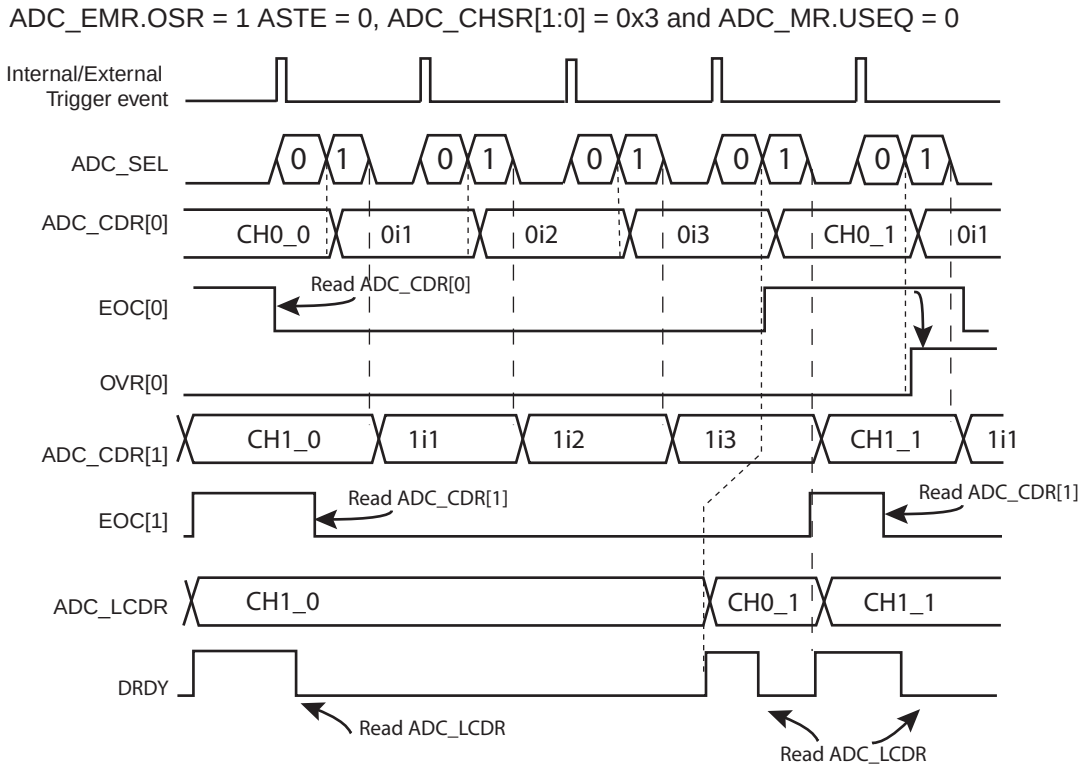
Consequently, when an overrun flag rises in the ADC_OVER, it means that the previous unread data is lost but it does not mean that this data has been overwritten by the new averaged value as the averaging sequence concerning this channel can still be ongoing.

32.6.10.1 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 the extended mode register (ADC_EMR) and the USEQ bit in the mode register (ADC_MR).

When USEQ is cleared, there are two possible ways to generate the averaging through the trigger event. If ASTE bit is cleared in the ADC_EMR, every trigger event generates one sample for each enabled channel as described in Figure 32-9, "Digital Averaging Function Waveforms Over Multiple Trigger Events". Therefore four trigger events are requested to get the result of averaging if OSR = 1.

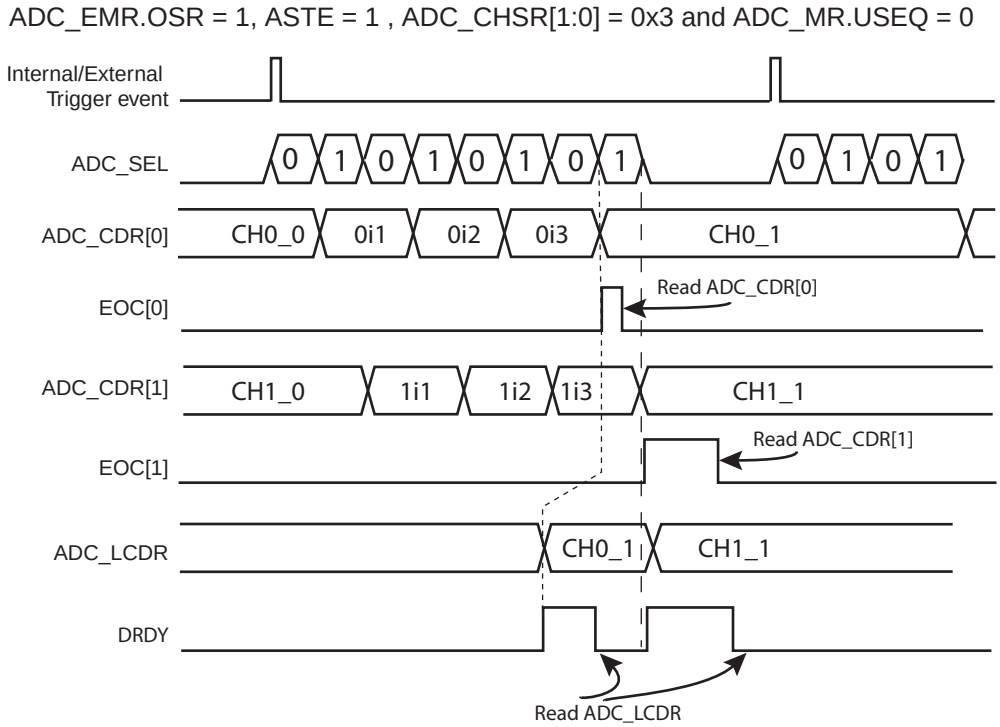
Figure 32-9. Digital Averaging Function Waveforms Over Multiple Trigger Events



Note: 0i1,0i2,0i3, 1i1, 1i2, 1i3 are intermediate results and CH0/1_0/1 are final result of average function.

If ASTE = 1 in the ADC_EMR and USEQ = 0 in the ADC_MR then the sequence to be converted, defined in the ADC_CHSR is automatically repeated n times (where n corresponds to the oversampling ratio defined in the OSR field in the ADC_EMR). As a consequence only 1 trigger is required to get the result of the averaging function as described in Figure 32-9, on page 796.

Figure 32-10. Digital Averaging Function Waveforms on a Single Trigger Event



Note: 0i1,0i2,0i3, 1i1, 1i2, 1i3 are intermediate results and CH0/1_0/1 are final result of average function.

When USEQ is set, the user can define the channel sequence to be converted by configuring the ADC_SEQRx and the ADC_CHER so that channels are not interleaved during averaging period. Under these conditions, a sample is defined for each end of conversion as described in Figure 32-11, on page 798.

Therefore, if the same channel is configured to be converted four times consecutively, and the OSR = 1 in the extended mode register (ADC_EMR), the averaging result will be placed in the corresponding channel data register ADC_CDRx and last converted data register (ADC_LCDR) for each trigger event.

In such case, the ADC effective sample rate remains the maximum ADC sample rate divided by 4 or 16, depending on OSR.

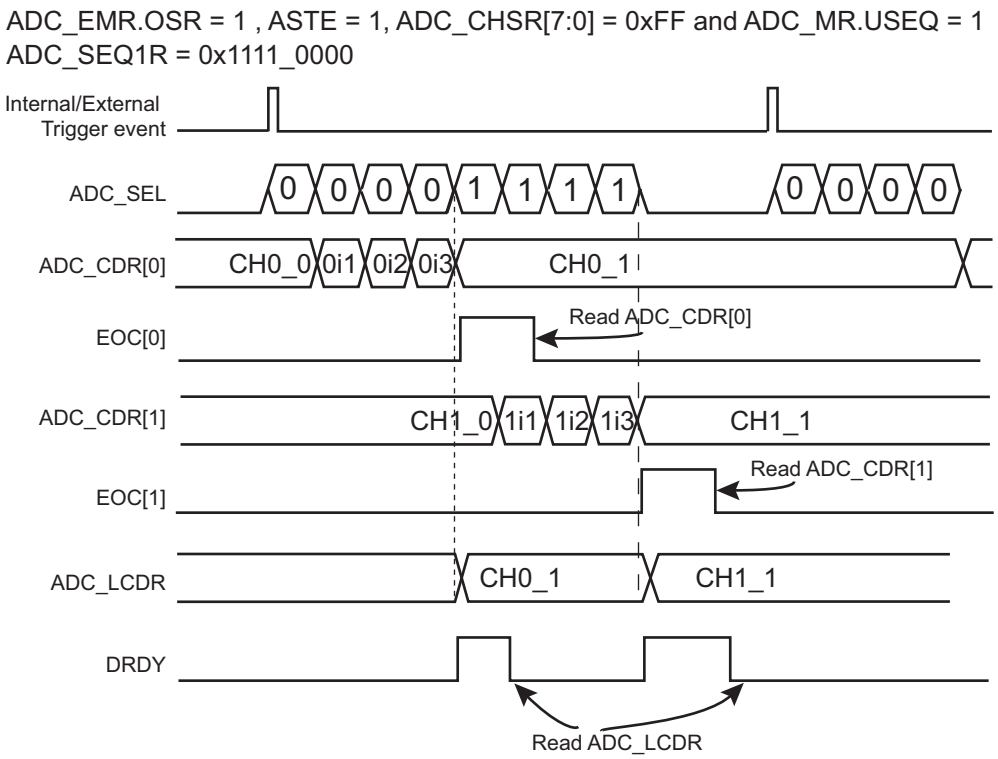
When USEQ = 1, ASTE = 1 and OSR is different from 0, it is important to notice that the user sequence must follow a specific pattern. The user sequence must be programmed in such a way that it generates a stream of conversion, where a given channel is successively converted with an integer multiple depending on OSR value. Up to four channels can be converted in this specific mode.

When OSR = 1, each channel to convert must be consecutively repeated four times in the sequence, so the four first single bit field enabled in the ADC_CHSR must have their associated channel index programmed to the same value in the ADC_SEQ1/2R registers. 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 consecutively repeated 16 times in the sequence, so all fields must be enabled in the ADC_CHSR, and their associated channel index programmed to the same value in the ADC_SEQ1/2R registers. 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 32-11. Digital Averaging Function Waveforms on a Single Trigger Event, Non-interleaved



Note: 0i1,0i2,0i3, 1i1, 1i2, 1i3 are intermediate results and CH0/1_0/1 are final result of average function.

32.6.10.2 Oversampling Digital Output Range

When an oversampling is performed, the maximum value that can be read on the 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 then is right shifted by 1 (divided by 2).

The maximum output value carried on the ADC_CDRx or ADC_LCDR depends on the OSR value configured in the ADC_EMR.

Table 32-4. 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

32.6.11 Automatic Calibration

The ADC features an automatic calibration (AUTOCALIB) mode for gain errors (calibration).

The automatic calibration sequence can be started at any time writing to '1' the AUTOCAL bit of the ADC Control Register. The automatic calibration sequence requires a software reset command (SWRST in the ADC_CR) prior to write AUTOCAL bit. The end of calibration sequence is given by the EOCAL bit in the interrupt status register (ADC_ISR), and an interrupt is generated if EOCAL interrupt has been enabled (ADC_IER).

The calibration sequence will perform an automatic calibration on all enabled channels. The channels required for conversion do not need to be all enabled during the calibration process if they are programmed with the same gain. Only

channels with different gain settings need to be enabled. The gain settings of all enabled channels must be set before starting the AUTOCALIB sequence. If the gain settings (ADC_CGR and ADC_COR) for a given channel are changed, the AUTOCALIB sequence must then be started again.

The calibration data (on one or more enabled channels) is stored in the internal ADC memory.

Then, when a new conversion is started (on one or more enabled channels), the converted value (in ADC_LCDR or ADC_CDRx) is a calibrated value.

Autocalibration is for settings, not for channels. Therefore, if a specific combination of gain has been already calibrated, and a new channel with the same settings is enabled after the initial calibration, there is no need to restart a calibration. If different enabled channels have different gain settings, the corresponding channels must be enabled before starting the calibration.

If a software reset is performed (SWRST bit in ADC_CR) or after power up (or wake-up from Backup mode), the calibration data in the ADC memory is lost.

Changing the ADC running mode (in ADC_CR) does not affect the calibration data.

Changing the ADC reference voltage (ADVREF pin) requires a new calibration sequence.

For calibration time, gain error after calibration, refer to the 12-bit ADC electrical characteristics section of the product.

32.6.12 Buffer Structure

The PDC read channel is triggered each time a new data is stored in ADC_LCDR. The same structure of data is repeatedly stored in ADC_LCDR each time a trigger event occurs. Depending on user mode of operation (ADC_MR, ADC_CHSR, ADC_SEQR1) the structure differs. Each data read to PDC buffer, carried on a half-word (16-bit), consists of last converted data right aligned and when TAG is set in ADC_EMR, the four most significant bits are carrying the channel number thus allowing an easier post-processing in the PDC buffer or better checking the PDC buffer integrity.

32.6.13 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 the protected registers 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 flag is automatically reset by 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 Last Channel Trigger Mode Register
- ADC Last Channel Compare Window Register
- ADC Extended Mode Register
- ADC Compare Window Register

32.7 Analog-to-Digital Converter (ADC) User Interface

Table 32-5. 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 Register 1	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	Last Channel Trigger Mode Register	ADC_LCTMR	Read/Write	0x00000000
0x38	Last Channel Compare Window Register	ADC_LCCWR	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	–	–	–
0x98–0xAC	Reserved	–	–	–
0xC4–0xE0	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”.

32.7.1 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
–	–	–	–	AUTOCAL	–	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.

- **AUTOCAL: Automatic Calibration of ADC**

0: No effect.

1: Launches an automatic calibration of the ADC cell on the next sequence.

32.7.2 ADC Mode Register

Name: ADC_MR

Address: 0x40038004

Access: Read/Write

31	30	29	28	27	26	25	24
USEQ	DIV3	TRANSFER		TRACKTIM			
23	22	21	20	19	18	17	16
–	DIV1	–	–	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 the 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
1	ADC_TRIG1	TIO Output of the Timer Counter Channel 0
2	ADC_TRIG2	TIO Output of the Timer Counter Channel 1
3	ADC_TRIG3	TIO Output of the Timer Counter Channel 2
4	ADC_TRIG4	RTCOUT0
5	ADC_TRIG5	RTTINC
6	ADC_TRIG6	Reserved
7	–	Reserved

- **LOWRES: Resolution**

Value	Name	Description
0	BITS_12	12-bit resolution. For higher resolution by averaging, please refer to “ADC Extended Mode Register” on page 818
1	BITS_10	10-bit resolution

- **SLEEP: Sleep Mode**

Value	Name	Description
0	NORMAL	Normal Mode: The ADC Core and reference voltage circuitry are kept ON between conversions
1	SLEEP	Sleep Mode: The ADC Core and reference voltage circuitry are OFF between conversions

- **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 cleared when digital averaging is used (OSR ≠ 0 in ADC_EMR).

- **PRESCAL: Prescaler Rate Selection**

  ADCClock = MCK / ((PRESCAL+1) × 2). If DIV1 = 1 or DIV3 = 1 then PRESCAL has no effect and the clock division is defined by either DIV1 bit or DIV3 bit.

- **STARTUP: Start Up Time**

Value	Name	Description
0	SUT0	0 periods of ADCClock
1	SUT8	8 periods of ADCClock
2	SUT16	16 periods of ADCClock
3	SUT24	24 periods of ADCClock
4	SUT64	64 periods of ADCClock
5	SUT80	80 periods of ADCClock
6	SUT96	96 periods of ADCClock
7	SUT112	112 periods of ADCClock
8	SUT512	512 periods of ADCClock
9	SUT576	576 periods of ADCClock
10	SUT640	640 periods of ADCClock
11	SUT704	704 periods of ADCClock
12	SUT768	768 periods of ADCClock
13	SUT832	832 periods of ADCClock
14	SUT896	896 periods of ADCClock
15	SUT960	960 periods of ADCClock

- **DIV1: ADCClock Prescaler Division forced to 1**

Value	Name	Description
0	NO_DIV1	The PRESCAL field is used to generate the ADCClock unless DIV3 is written to 1.
1	FORCE_DIV1	The ADCClock equals MCK (PRESCAL and DIV3 fields have no effects).

- **TRACKTIM: Tracking Time**

  Tracking Time = (TRACKTIM + 1) × ADCClock periods

- **DIV3: ADCClock Prescaler Division forced to 3**

Value	Name	Description
0	NO_DIV3	The PRESCAL field is used to generate the ADCClock unless DIV1 is written to 1.
1	FORCE_DIV3	The ADCClock equals MCK/3 unless DIV1 is written to 1.

- **TRANSFER: Transfer Period**

  This field must be programmed with value 2.

- **USEQ: Use 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 register and can be used to convert the same channel several times.

32.7.3 ADC Channel Sequence 1 Register

Name: ADC_SEQR1

Address: 0x40038008

Access: Read/Write

31	30	29	28	27	26	25	24
–				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 the “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, according to user needs.

When configuring consecutive fields with the same value, the associated channel is sampled as many time as the number of consecutive values, this part of the conversion sequence being triggered by a unique event.

32.7.4 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 the “ADC Write Protection Mode Register” .

- **CHx: Channel x Enable**

0: No effect.

1: Enables the corresponding channel.

Note: If USEQ = 1 in the ADC_MR, CHx corresponds to the xth channel of the sequence described in ADC_SEQR1.

32.7.5 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 the “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 then reenabled during a conversion, its associated data and its corresponding EOC and OVRE flags in ADC_SR are unpredictable.

32.7.6 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.

32.7.7 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 bit is set in the ADC_EMR. If the TAG bit is not set, CHNB = 0.

32.7.8 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
EOCAL	–	–	–	LCCHG	–	–	–
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**
- **LCCHG: Last Channel Change Interrupt Enable**
- **EOCAL: End of Calibration Sequence**
- **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**

32.7.9 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
EOCAL	–	–	–	LCCHG	–	–	–
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**
- **LCCHG: Last Channel Change Interrupt Disable**
- **EOCAL: End of Calibration Sequence**
- **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**

32.7.10 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
EOCAL	–	–	–	LCCHG	–	–	–
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**
- **LCCHG: Last Channel Change Interrupt Mask**
- **EOCAL: End of Calibration Sequence**
- **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**

32.7.11 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
EOCAL	–	–	–	LCCHG	–	–	–
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**

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.

- **LCCHG: Last Channel Change**

0: There is no comparison match (defined in the Last Channel Compare Window Register (ADC_LCCWR)) since the last read of the 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 the Last Channel Trigger Mode Register (ADC_LCTMR) and Last Channel Compare Window Register (ADC_LCCWR).

- **EOCAL: End of Calibration Sequence**

0: Calibration sequence is ongoing, or no calibration sequence has been requested.

1: Calibration sequence is complete.

- **DRDY: Data Ready**

0: No data has been converted since the last read of ADC_LCDR.

1: At least one data has been converted and is available in ADC_LCDR.

- **GOVRE: General Overrun Error**

0: No General Overrun Error occurred since the last read of ADC_ISR.

1: At least one General Overrun Error has occurred since the last read of ADC_ISR.

- **COMPE: Comparison Error**

0: No Comparison Error since the last read of ADC_ISR.

1: At least one Comparison Error (defined in the ADC_EMR and ADC_CWR) has occurred since the last read of ADC_ISR.

- **ENDRX: End of Receiver Transfer**

0: The end of transfer signal from the receive PDC channel is inactive.

1: The end of transfer signal from the receive PDC channel is active.

- **RXBUFF: Reception Buffer Full**

0: The signal Buffer Full from the Receive PDC channel is inactive.

1: The signal Buffer Full from the Receive PDC channel is active.

32.7.12 ADC Last Channel Trigger Mode Register

Name: ADC_LCTMR

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
–	–	CMPMOD		–	–	–	DUALTRIG

This register can only be written if the WPEN bit is cleared in the “ADC Write Protection Mode Register” .

- **DUALTRIG: Dual Trigger ON**

0: All channels are triggered by event defined by ADC_MR.TRGSEL.

1: Last channel (higher index) trigger period is defined by RTC_MR.OUT1.

- **CMPMOD: Last Channel 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.

32.7.13 ADC Last Channel Compare Window Register

Name: ADC_LCCWR

Address: 0x40038038

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the “ADC Write Protection Mode Register” .

- **LOWTHRES: Low Threshold**

Low threshold associated to compare settings of the ADC_LCTMR.

- **HIGHTHRES: High Threshold**

High threshold associated to compare settings of the ADC_LCTMR.

32.7.14 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. Please refer to Section 32.6.10 ”Enhanced Resolution Mode and Digital Averaging Function” for details.

32.7.15 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 the "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 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 the ADC Mode Register.

Note: FREERUN (see ADC_MR) must be cleared 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**

0: Sets CHNB to zero in ADC_LDCR.

1: Appends the channel number to the conversion result in ADC_LDCR.

32.7.16 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 the “ADC Write Protection Mode Register” .

- **LOWTHRES: Low Threshold**

Low threshold associated to compare settings of the ADC_EMR.

If LOWRES is set in ADC_MR, only the 12 LSB of LOWTHRES must be programmed. The two LSB will be automatically discarded to match the value carried on ADC_CDR (10-bit).

- **HIGHTHRES: High Threshold**

High threshold associated to compare settings of the ADC_EMR.

If LOWRES is set in ADC_MR, only the 12 LSB of HIGHTHRES must be programmed. The two LSB will be automatically discarded to match the value carried on ADC_CDR (10-bit).

32.7.17 ADC Channel Data Register

Name: ADC_CDRx [x=0..7]

Address: 0x40038050

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
–	–	–	–	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.

32.7.18 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 Protection Enable**

0: Disables the write protection if WPKEY value corresponds to 0x414443 (“ADC” in ASCII).

1: Enables the write protection if WPKEY value corresponds to 0x414443 (“ADC” in ASCII).

See Section 32.6.13 “Register Write Protection” for list of write-protected registers.

- **WPKEY: Write Protection 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

32.7.19 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.

1: A write protection violation has occurred since the last read of the ADC_WPSR. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protection Violation Source**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

33. SAM G51 Electrical Characteristics

33.1 Absolute Maximum Ratings

Table 33-1. Absolute Maximum Ratings*

Storage temperature.....-60°C to + 150°C
Voltage on input pins with respect to ground.....-0.3V to + 2.4V
Maximum operating voltage (VDDIO).....2.4V
Total DC output current on all I/O lines 49-lead WLCSP.....100 mA 100-lead LQFP.....100 mA

*NOTICE: Stresses beyond those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. **Exposure to absolute maximum rating conditions for extended periods may affect device reliability.**

33.2 DC Characteristics

The following characteristics are applicable to the operating temperature range: T_A = -40°C to +85°C, unless otherwise specified.

Table 33-2. DC Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{DDCORE}	DC Supply Core	Supplied by voltage regulator ONLY	–	–	1.2	V
V_{VDDIO}	DC Supply I/Os		1.62	1.8	2	V
V_{IL}	Input Low-level Voltage	PA0–PA24, PB0–PB12 NRST	-0.3	–	[0.8V:0.3 x V_{VDDIO}]	V
V_{IH}	Input High-level Voltage	PA0–PA24, PB0–PB12 NRST	[2.0V:0.7 x V_{VDDIO}]	–	V_{VDDIO} +0.3V	V
V_{OH}	Output High-level Voltage	PA7–PA8, PB8–PB9, PB12(I_{OH} = 6.0 mA) Others[2](I_{OH} = 4.0 mA) NRST	V_{VDDIO} -0.4V	–	–	V
V_{OL}	Output Low-level Voltage	PA7–PA8, PB8–PB9, PB12 (I_{OH} = 6.0 mA) Others[2](I_{OH} = 4.0 mA) NRST	–	–	0.4	V
V_{Hys}	Hysteresis Voltage	PA0–PA24, PB0–PB12 (Hysteresis mode enabled)	150	–	–	mV
I_O	I_{OH} (or I_{SOURCE})	VDDIO [1.62V : 2.0V] ; V_{OH} = V_{VDDIO} - 0.4 - PA3–PA4, PA14 - PA7–PA13, PA15–PA24, PB0–PB4, PB8–PB9, PB12 - others[1]	–	–	-6 -4 -2	mA
		VDDIO [1.62V : 2.0V] ; V_{OH} = V_{VDDIO} - 0.4 - NRST	–	–	-4	
I_O	I_{OL} (or I_{SINK})	VDDIO [1.62V : 2.0V] ; V_{OH} = V_{VDDIO} - 0.4 - PA3–PA4, PA14 - PA7–PA13, PA15–PA24, PB0–PB4, PB8–PB9, PB12 - others[1]	–	–	6 4 2	mA
		VDDIO [1.62V : 2.0V] ;V_{OL} = 0.4V - NRST	–	–	4	
I_{IL}	Input Low	Pull-up OFF	-1	–	1	uA
		Pull-up ON	10	–	30	
I_{IH}	Input High	Pull-down OFF	-1	–	1	uA
		Pull-down ON	10	–	30	

Table 33-2. DC Characteristics (Continued)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
R_{PULLUP}	Pull-up Resistor	PA0–PA24, PB0–PB12 NRST	70	100	130	kΩ
$R_{PULLDOWN}$	Pull-down Resistor	PA0–PA24, PB0–PB12 NRST	70	100	130	kΩ
R_{ODT}	On-die Series Termination Resistor	PA0–PA24, PB0–PB12	–	36	–	Ω
I_{CC}	Flash Active Current on VDDCORE	Random 128-bit read @max frequency @ 25°C	–	8	12	mA
		Random 64-bit read @max frequency @ 25°C	–	4	6	
		Program @ 25°C	–	3	5	
		Erase @ 25°C	–	3	5	
I_{CC20}	Flash Active Current on VDDIO	Random 128-bit read @max frequency @ 25°C	–	6	10	mA
		Random 64-bit read @max frequency @ 25°C	–	6	10	
		Program @ 25°C	–	10	15	
		Erase @ 25°C	–	10	15	

Note: 1. PA0–PA2, PB5–PB7, PB10–PB11]
2. PA0–PA6, PA9–PA24, PB0–PB7, PB10–PB11

Table 33-3. VDDCORE Voltage Regulator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{VDDIO}	DC Input Voltage Range		1.62	1.8	2.0	V
	Allowable Voltage Ripple	RMS value 10 kHz to 1 MHz RMS value > 1 MHz	–	–	50 10	mV
V_{VDDOUT}	DC Output Voltage	Normal mode	–	–	1.2	V
$V_{ACCURACY}$	Output Voltage Accuracy	Initial output voltage accuracy	-6.5	–	5.5	%
I_{LOAD}	Maximum DC Output Current		–	–	25	mA
V_{LINE}	Line Regulation	V_{VDDIO} from1.62V to 2V; I_{Load} MAX	–	10	18	mV
$V_{LINE-TR}$	Transient Line Regulation	V_{VDDIO} from 1.62V to 2V; tr = tf = 50 µs; I_{Load} MAX			95	
V_{LOAD}	Load Regulation	I_{Load} = 20% to 80% MAX	–	1	10	mV
$V_{LOAD-TR}$	Transient Load Regulation	I_{Load} = 20% to 80% MAX; tr = tf = 5 µs		40	70	
I_Q	Quiescent Current	Normal mode; @ I_{Load} = 0 mA	–	1.03	1.32	µA
CD_{IN}	Input Decoupling Capacitor	Cf. External Capacitor Requirements (1)	–	4.7	–	µF
CD_{OUT}	Output Decoupling Capacitor	Cf. External Capacitor Requirements (2)	1.0	2.2	3.3	µF
t_{ON}	Turn-on Time for Standby to Normal Mode	CD_{OUT}= 2.2 µF, V_{VDDOUT} reaches max VDDOUT	–	–	350	µs

Notes: 1. A 4.7 µF or higher ceramic capacitor must be connected between VDDIO and the closest GND pin of the device. This large decoupling capacitor is mandatory to reduce startup current, thus improving transient response and noise rejection.

2. To ensure stability, an external 2.2 µF output capacitor CD_{OUT} must be connected between the VDDOUT and the closest GND pin of the device. The ESR (Equivalent Series Resistance) of the capacitor must be in the range 20 to 200m Ω.Ceramic capacitors are suitable as output capacitors.
A 100 nF bypass capacitor between VDDOUT and the closest GND pin of the device helps to decrease output noise and improves the load transient response.

Table 33-4. Core Power Supply Brownout Detector Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{TH-}	Supply Falling Threshold	After trimming	0.58	–	0.91	V
V_{HYST}	Hysteresis		5	20	55	mV
V_{TH+}	Supply Rising Threshold		0.60	0.8	0.945	V
I_{DDON}	Current Consumption on VDDCORE	Brownout Detector enabled	–	4.5	6.9	µA
I_{DDOFF}		Brownout Detector disabled			1	uA
I_{DDIOON}	Current Consumption on VDDIO	Brownout Detector enabled	–	1.3	2	µA
$I_{DDIOOFF}$		Brownout Detector disabled			1	µA
t_{d-}	V_{TH-} Detection Propagation Time		–	1	15	µs
t_{START}	Startup Time	From disabled state to enabled state	90	200	450	µs

Figure 33-1. Core Brownout Output Waveform

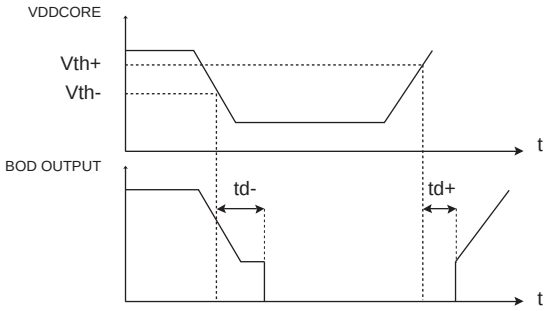


Table 33-5. VDDIO Supply Monitor

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{TH}	Supply Monitor Threshold	4 selectable steps	1.6	–	2	V
$T_{ACCURACY}$	Threshold Level Accuracy	[-40°C/+85°C]	-3	–	+3	%

Table 33-5. VDDIO Supply Monitor

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{HYST}	Hysteresis		–	20	30	mV
I_{DD}	Current Consumption	All temperature ranges, normal mode standby mode	–	–	 40 2	 µA µA
t_{START}	Startup Time	From disabled state to enabled state	–	–	300	µs

The threshold selection is done through the SUPC_SMMR register in the Supply Controller.

Table 33-6. Threshold Selection

Digital Code	Threshold min (V)	Threshold typ (V)	Threshold max (V)
0000	1.55	1.6	1.65
0001	1.67	1.72	1.78
0010	1.78	1.84	1.86
0011	1.90	1.96	2.02

Figure 33-2. VDDIO Supply Monitor

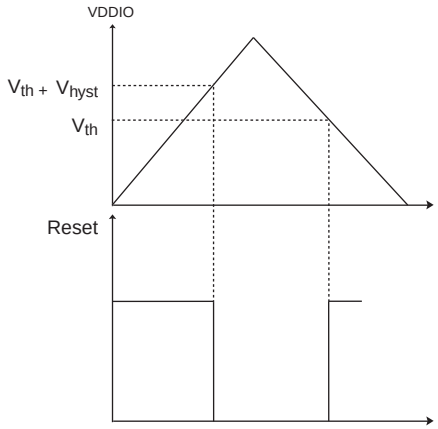
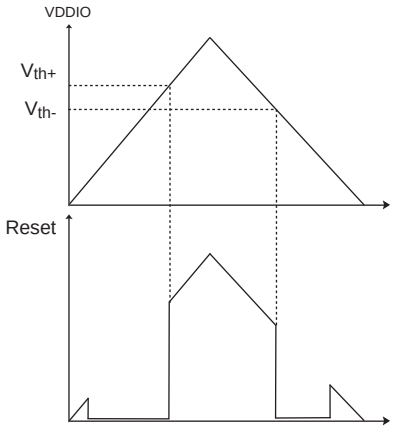


Table 33-7. Power-on Reset Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{th+}	Threshold Voltage Rising	At startup	1.45	1.53	1.595	V
V_{th-}	Threshold Voltage Falling		1.35	1.45	1.55	V
t_{RES}	Reset Timeout Period	Temperature @ 25°C	100	240	500	µs
I_{POR}	Current Consumption		–	300	700	nA

Figure 33-3. Power-on Reset Characteristics



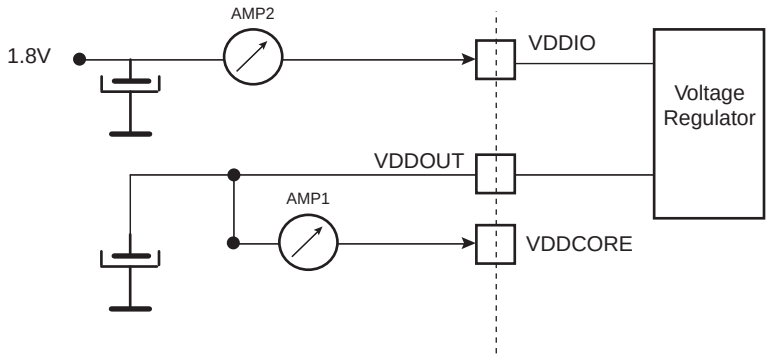
33.3 Power Consumption

- Power consumption of the device in low power mode (Wait Mode) and active mode.
- Power consumption by peripheral: calculated as the difference in current measurement after having enabled then disabled the corresponding clock.

33.3.1 Wait Mode Current Consumption

The wait mode configuration and measurements are defined below.

Figure 33-4. Wait Mode Measurement Setup



- Core clock and master clock stopped
- There is no activity on the I/Os of the device.
- Current measurement as shown in the above figure
- All peripheral clocks deactivated
- BOD disabled
- VDDIO = 1.8V
- VDDCORE = internal voltage regulator used

Table 33-8 “Typical Current Consumption in Wait Mode” gives current consumption in typical conditions.

Table 33-8. Typical Current Consumption in Wait Mode

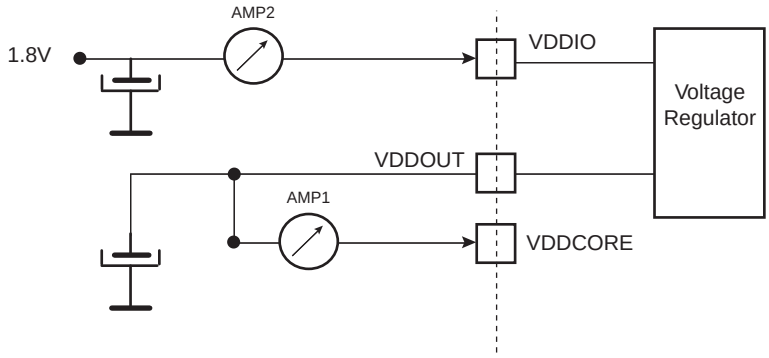
Wait Mode Consumption	Typical Value @25°C		Typical Value @85°C		
Conditions	VDDOUT Consumption (AMP1)	Total Consumption (AMP2)	VDDOUT Consumption (AMP1)	Total Consumption (AMP2)	Unit
See Figure 33-4 on page 830 with the Flash in standby mode	4.6	10.9	88	98	µA
See Figure 33-4 on page 830 with the Flash in deep-power-down mode	4.5	6.8	87.6	93	µA

33.3.2 Sleep Mode Current Consumption

The sleep mode configuration and measurements are defined below.

Reminder : The purpose of sleep mode is to optimize power consumption of the device with respect to response time.

Figure 33-5. Measurement Setup for Sleep Mode



33.3.2.1 Sleep Mode Configuration

- Core clock off
- VDDIO=1.8V
- No activity on I/O lines
- Master clock (MCK) running at various frequencies with PLLA or the fast RC oscillator
- Fast start-up through WKUP0–15 pins
- Current measurement as shown in Figure 33-5
- All peripheral clocks deactivated
- VDDCORE = internal voltage regulator used
- Temperature = 25°C

Table 33-9 below gives current consumption in typical conditions.

Figure 33-6. Current Consumption in Sleep Mode (AMP1) versus Master Clock Ranges (Conditions from Table 33-9)

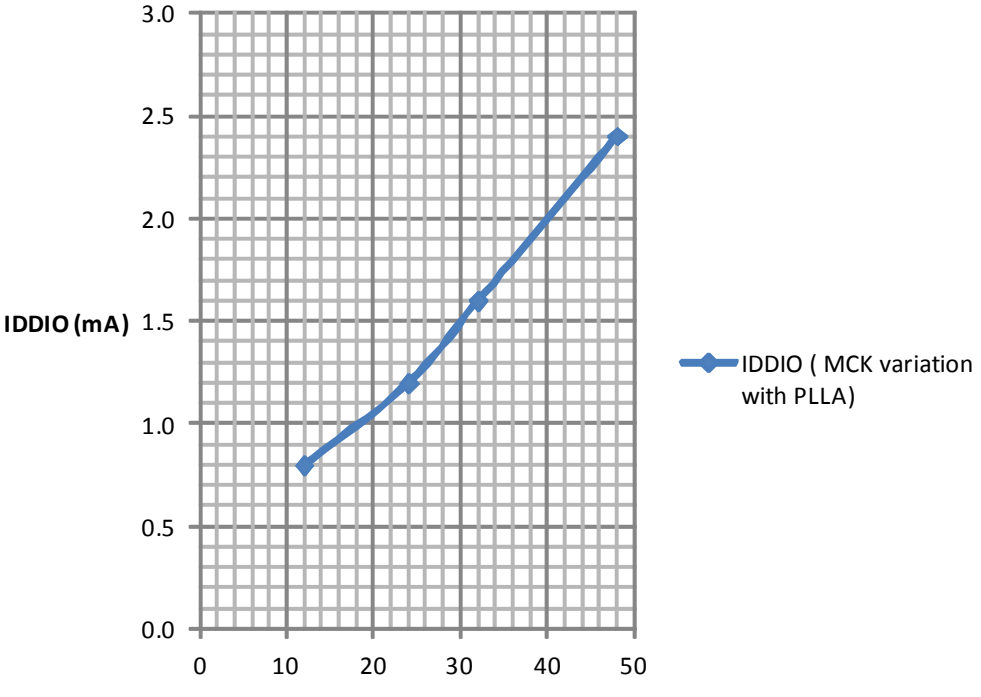


Table 33-9. Typical Sleep Mode Current Consumption versus Master Clock (MCK) Variation with PLLA

Sleep Mode Consumption	Typical Value @25°C		
Core Clock/MCK (MHz)	VDDCORE Consumption (AMP1)	Total Consumption (AMP2)	Unit
48	2.2	2.4	mA
32	1.4	1.6	
24	1.1	1.2	
12	0.8	0.7	

Table 33-10. Typical Sleep Mode Current Consumption versus Master Clock (MCK) Variation with Fast RC

Sleep Mode Consumption	Typical Value @25°C		
Core Clock/MCK (MHz)	VDDCORE Consumption (AMP1)	Total Consumption (AMP2)	Unit
8	520	530	µA
4	350	360	
2	270	280	
1	230	240	
0.5	210	210	
0.25	200	200	

33.3.3 Active Mode Power Consumption

The active mode configuration and measurements are defined as follows:

- VDDIO = 1.8V
- VDDCORE = internal voltage regulator used
- T_A = 25°C
- Fibonacci algorithm runs from Flash memory with128-bit or 64-bit access mode or from SRAM
- All peripheral clocks are deactivated
- Master Clock (MCK) runs at various frequencies with PLLA or the fast RC oscillator
- Current measurement on AMP1 (VDDCORE) and total current on AMP2

Figure 33-7. Active Mode Measurement Setup

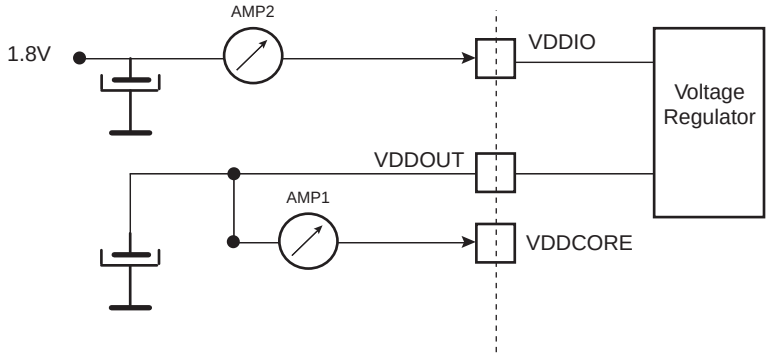


Table 33-11 “Typical Active Power Consumption Running @25°C with PLLA” and Table 33-12 “Typical Active Power Consumption Running @25°C with Fast RC” provide active mode current consumption in typical conditions.

Table 33-11. Typical Active Power Consumption Running @25°C with PLLA

Core Clock (MHz)	Wait State	FIbonacci Algorithm					Unit
		128-bit Flash Access		64-bit Flash Access		SRAM	
		AMP1	AMP2	AMP1	AMP2	AMP2	
48	3	9.1	11.3	6.6	8.7	4.97	mA
32	2	6.3	9.1	5.5	7.1	3.35	
24	1	5.4	7.8	4.8	6.4	2.67	
12	0	4.1	5.3	3.3	4.5	1.75	

Table 33-12. Typical Active Power Consumption Running @25°C with Fast RC

Core Clock (MHz)	Wait State	FIbonacci Algorithm					Unit
		128-bit Flash Access		64-bit Flash Access		SRAM	
		AMP1	AMP2	AMP1	AMP2	AMP2	
8	0	2.7	3.2	2.1	3.0	1.1	mA
4	0	1.2	1.9	1.1	2.0	0.7	
2	0	0.8	1.5	0.7	1.5	0.4	
1	0	0.4	0.8	0.4	0.8	0.3	
0.5	0	0.3	0.5	0.3	0.5	0.3	
0.25	0	0.3	0.4	0.3	0.4	0.2	

33.3.4 Peripheral Power Consumption in Active Mode

The peripheral consumption configuration and measurements are defined as follows:

- VDDIO = 1.8V
- VDDCORE = internal voltage regulator used
- T_A = 25°C
- Frequency = 48 MHz

Table 33-13. Power Consumption on V_{DDCORE}

Peripheral	Consumption (Typ)	Unit
PIO Controller A (PIOA)	1.7	µA/MHz
PIO Controller B (PIOB)	0.4	
MEM2MEM	3.2	
UART0	3.4	
UART1	3.6	
USART0	4.8	
TWIHS	5.1	
TWI1	3.5	
TWI2	3.7	
SPI	3.1	
Timer Counter (TC0_CH0)	3.1	
Timer Counter (TC0_CH1)	0.3	
Timer Counter (TC0_CH2)	0.3	
ADC	3.1	

33.4 Oscillator Characteristics

33.4.1 32 kHz RC Oscillator Characteristics

Table 33-14. 32 kHz RC Oscillator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
	RC Oscillator Frequency		20	32	44	kHz
	Frequency Supply Dependency		-3	–	3	%/V
	Frequency Temperature Dependency	Over temperature range (-40°C/ +85°C) versus 25°C	-7	–	7	%
Duty	Duty Cycle		45	50	55	%
t_{ON}	Startup Time		–		100	µs
I_{DDON}	Current Consumption	After startup time Temp. range = -40°C to +85°C Typical consumption at 2V supply and temp = 25°C	–	540	860	nA

33.4.2 8/16/24 MHz RC Oscillators Characteristics

Table 33-15. 8/16/24 MHz RC Oscillator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{RANGE}	RC Oscillator Frequency Range	(1)	8	16	24	MHz
ACC_8	8 MHz Total Accuracy	8 MHz output selected (1)(2)	-40	–	+40	%
ACC_{16}	16 MHz Total Accuracy	16 MHz output selected (1)(2)	-40	–	+40	%
		16 MHz output selected (1)(3) (with VDDCORE=VDDOUT)	-5	–	+5	
ACC_{24}	24 MHz Total Accuracy	24 MHz output selected (1)(2)	-40	–	+40	%
		24 MHz output selected (1)(3) (with VDDCORE=VDDOUT)	-5	–	+5	
Duty	Duty Cycle		45	50	55	%
t_{ON}	Startup Time	Time after onrc is set to 1	–	3.2	6	µs
t_{ONTRIM}	Stabilizing Time		–	1	–	µs
I_{DDON}	Active Current Consumption(2)	8 MHz 16 MHz 24 MHz	–	92 132 166	130 180 250	µA
	Acitve Current Consumption(3)	8 MHz 16 MHz 24 MHz	–	86 121 153	120 160 190	µA

Notes: 1. Frequency range can be configured in the Supply Controller registers.
2. Not trimmed from factory
3. After trimming from factory

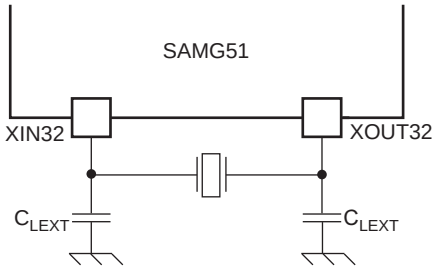
The 16/24 MHz fast RC oscillator is calibrated in production. This calibration can be read through the Get CALIB Bit command (see EEFC section) and the frequency can be trimmed by software through the PMC.

33.4.3 32.768 kHz Crystal Oscillator Characteristics

Table 33-16. 32.768 kHz Crystal Oscillator Characteristics

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
f_{REQ}	Operating Frequency	Normal mode with crystal		–	–	32.768	kHz
	Supply Ripple Voltage (on VDDIO)	Rms value, 10 kHz to 10 MHz		–	–	30	mV
	Duty Cycle			40	50	60	%
t_{ON}	Startup Time	R_S < 50 KΩ	$C_{crystal}$ = 12.5 pF	–	–	900	ms
			$C_{crystal}$ = 6 pF			300	
		R_S < 100 KΩ (1)	$C_{crystal}$ = 12.5 pF			1200	
			$C_{crystal}$ = 6 pF			500	
I_{DDON}	Current Consumption	R_S < 50 KΩ	$C_{crystal}$ = 12.5 pF	–	–	1150	nA
			$C_{crystal}$ = 6 pF			980	
		R_S < 100 KΩ (1)	$C_{crystal}$ = 12.5 pF			1600	
			$C_{crystal}$ = 6 pF			1350	
P_{ON}	Drive Level			–	–	0.1	µW
R_f	Internal Resistor	Between XIN32 and XOUT32		–	10	–	MΩ
C_{LEXT}	Maximum External Capacitor on XIN32 and XOUT32			–	–	20	pF
C_{para}	Internal Parasitic Capacitance			0.6	0.7	0.8	pF

Note: 1. R_S is the series resistor.



$C_{LEXT} = 2x(C_{CRYSTAL} – C_{para} – C_{PCB})$.

Where C_{PCB} is the capacitance of the printed circuit board (PCB) track layout from the crystal to the SAM G51 pin.

33.4.4 32.768 kHz Crystal Characteristics

Table 33-17. Crystal Characteristics

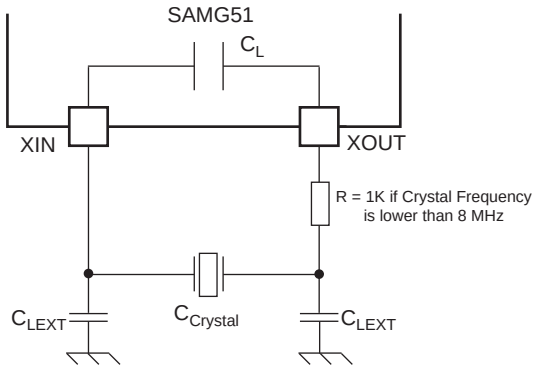
Symbol	Parameter	Conditions	Min	Typ	Max	Unit
ESR	Equivalent Series Resistor ($R_{S)}$	Crystal @ 32.768 KHz	–	50	100	KΩ
C_M	Motional Capacitance	Crystal @ 32.768 KHz	0.6	–	3	fF
C_{SHUNT}	Shunt Capacitance	Crystal @ 32.768 KHz	0.6	–	2	pF
C_{LOAD}	Load Capacitance	Cystal @ 32.768 KHz Max external capacitor 20 pF	6	–	12.5	pF

33.4.5 3 to 20 MHz Crystal Oscillator Characteristics

Table 33-18. 3 to 20 MHz Crystal Oscillator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{REQ}	Operating Frequency	Normal mode with crystal	3	16	20	MHz
VDDIO	Supply voltage		1.62	1.8	2.0	V
VDDCORE	Supply Voltage	Supplied by voltage regulator ONLY	–	–	1.2	V
	Supply Ripple Voltage	Rms value, 10 KHz to 10 MHz	–	–	30	mV
	Duty Cycle		40	50	60	%
t_{ON}	Startup 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_{DD_ON}	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
I_{DD_ON}	Current Consumption (on VDDCORE)	3 MHz(2) 8 MHz(3) 16 MHz(4) 20 MHz(5)	–	6 12 20 24	7 14 23 30	µA
P_{ON}	Drive Level	3 MHz 8 MHz 16 MHz, 20 MHz	–	–	15 30 50	µW
R_f	Internal resistor	Between XIN and XOUT	–	0.5		MΩ
C_{LEXT}	Maximum External Capacitor on XIN and XOUT		–	–	17	pF
$C_{INTLOAD}$	Internal Load Capacitance	Integrated load capacitance ((XIN) (GND) and (XOUT)(GND) in series)	7.5	9	10.5	
C_L	Internal Equivalent Load Capacitance	Integrated load capacitance (XIN and XOUT in series)	12.5	–	17.5	pF
$C_{PARASTANDBY}$	Internal Parasitic During Standby	On XIN-ONOSC = 0 On XOUT-ONOSC = 0	–	5.5 2.9	6.3 3.3	pF
$R_{PARASTANDBY}$	Internal Impedance During Standby	On XIN-ONOSC = 0	–	300	–	Ω

Notes:
1. R_S is the series resistor
2. R_S = 100–200 Ω; Cs = 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; Cs = 2.0–2.5 pF; C_M = 4–3 fF(typ, worst case).
4. R_S = 25–50 Ω; Cs = 2.5–3.0 pF; C_M = 7–5 fF (typ, worst case).
5. R_S = 20–50 Ω; Cs = 3.2–4.0 pF; C_M = 10–8 fF(typ, worst case).



$C_{LEXT} = 2x(C_{CRYSTAL} – C_L – C_{PCB})$.

Where C_{PCB} is the capacitance of the printed circuit board (PCB) track layout from the crystal to the SAM G51 pin.

33.4.6 3 to 20 MHz Crystal Characteristics

Table 33-19. Crystal Characteristics

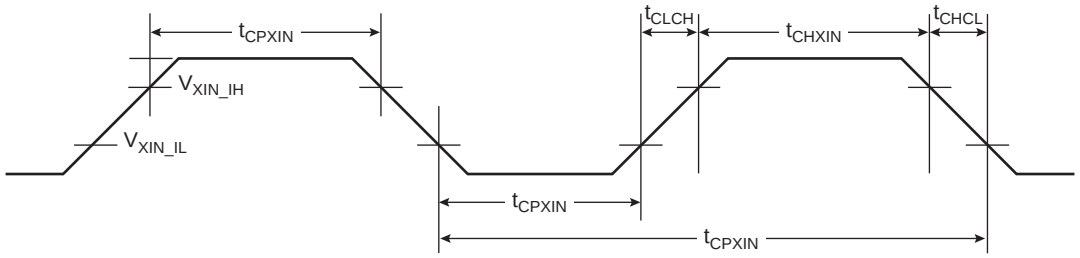
Symbol	Parameter	Conditions	Min	Typ	Max	Unit
ESR	Equivalent Series Resistor (R_S)	Fundamental @ 3 MHz Fundamental @ 8 MHz Fundamental @ 16 MHz Fundamental @ 20 MHz	–	–	200 100 80 50	Ω
C_M	Motional Capacitance		–	–	8	fF
C_{SHUNT}	Shunt Capacitance		–	–	7	pF
C_{LOAD}	Load Capacitance	Max external capacitors: 17 pF	12.5	–	17.5	pF

33.4.7 3 to 20 MHz XIN Clock Input Characteristics in Bypass Mode

Table 33-20. XIN Clock Electrical Characteristics (In Bypass Mode)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$1/(t_{CPXIN})$	XIN Clock Frequency	(1)	–	–	50	MHz
t_{CPXIN}	XIN Clock Period	(1)	20	–	–	ns
t_{CHXIN}	XIN Clock High Half-period	(1)	8	–	–	ns
t_{CLXIN}	XIN Clock Low Half-period	(1)	8	–	–	ns
t_{CLCH}	Rise Time	(1)	2.2	–	–	ns
t_{CHCL}	Fall Time	(1)	2.2	–	–	ns
V_{XIN_IL}	V_{XIN} Input Low-level Voltage	(1)	-0.3	–	[0.8V:0.3 x V_{VDDIO}]	V
V_{XIN_IH}	V_{XIN} Input High-level Voltage	(1)	[2.0V:0.7 x V_{VDDIO}]	–	V_{VDDIO} +0.3V	V

Note: 1. These characteristics apply only when the 3–20 MHz XTAL Oscillator is in bypass mode.



33.4.8 Crystal Oscillator Design Considerations

33.4.8.1 Choosing a Crystal

When choosing a crystal for the 32768 Hz slow clock oscillator or for the 3–20 MHz oscillator, several parameters must be taken into account. Important parameters between crystal and SAM G51 specifications are as follows:

- Load Capacitance
  - $C_{crystal}$ is the equivalent capacitor value the oscillator must “show” to the crystal in order to oscillate at the target frequency. The crystal must be chosen according to the internal load capacitance (C_L) of the on-chip oscillator. Having a mismatch for the load capacitance will result in a frequency drift.
- Drive Level
  - Crystal drive level >= Oscillator Drive Level. Having a crystal drive level number lower than the oscillator specification may damage the crystal.
- Equivalent Series Resistor (ESR)
  - Crystal ESR <= Oscillator ESR Max. Having a crystal with ESR value higher than the oscillator may cause the oscillator to not start.
- Shunt Capacitance
  - Max. crystal Shunt capacitance <= Oscillator Shunt Capacitance (C_{SHUNT}). Having a crystal with ESR value higher than the oscillator may cause the oscillator to not start.

33.4.8.2 Printed Circuit Board (PCB)

SAM G51 oscillators are low-power oscillators requiring particular attention when designing PCB systems.

33.5 PLL Characteristics

Table 33-21. PLL Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{IN}	Input Frequency		–	32	–	KHz
f_{OUT}	Output Frequency		24	–	48	MHz
I_{PLLON}	Current Consumption	PLL is in active mode f_{OUT} = 24 MHz f_{OUT} = 48 MHz	–	 75 150	–	µA
I_{PLLOFF}	Current Consumption	PLL is in standby mode @25°C All temperature ranges	–	 0.05 0.05	 0.30 5	µA
t_{START}	Startup Time	Lock PLL	–	–	1.5	ms

33.6 12-bit ADC Characteristics

Table 33-22. Analog Power Supply Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{VDDIO} $V_{VDDCORE}$	ADC Analog Supply ADC Digital Supply	 Supplied by voltage regulator ONLY	1.6	1.8	2 1.2	V
I_{VDDIO}	Current Consumption	Sleep mode (Clock OFF) Normal mode (IBCTL= 01)	–	– 360	2 520	µA
$I_{VDDCORE}$	Current Consumption	Sleep mode (Clock OFF) Normal mode (IBCTL= 01)	–	– 10	1 20	µA

Table 33-23. Channel Conversion Time and ADC Clock

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{ADC}	ADC Clock Frequency	No missing code	1	–	10(1)	MHz
t_{CP_ADC}	ADC Clock Period		100	–	1000	ns
t_{CONV}	ADC Conversion Time	ADC clock frequency = 10 MHz	1	–	–	µs
f_S	Sampling Rate	(10-bit mode) (11-bit mode) (12-bit mode)	–	–	800(1) 200(1) 50(1)	kSPS
$t_{START-UP}$	ADC Startup Time	After normal mode (after onadi =1)	–	–	5	µs
$t_{TRACKTIM}$	Track and Hold Time	(1)	250	–	–	ns

Note: 1. The full speed is obtained for an input source impedance <50 Ω or $t_{TRACKING}$ = 250 ns.

33.6.1 ADC Resolution

33.6.1.1 Conditions

- @ 25°C
- f_{ADC} = 10 MHz
- OSR : Number of averaged samples
- VDDIO = 1.8V
- VDDCORE = Internal voltage regulator used

Table 33-24. ADC Resolution Following Digital Averaging(1)

Parameter Averaging Resolution OSR(2) (ADC_EMR)	Oversampling Ratio	Mode (bits)	INL (LSB)	DNL (LSB)	SNR (dB)	THD (dB)	ENOB (Bits)	FS (KSps)
OSR = 0	1	10	±1	- 3.5/+1	58	-67	9.2	800
OSR = 1	4	11	± -1.5	-1/+1.5	63	-76	10.2	200
OSR = 2	16	12	-4/+3.2	-1/+3.5	64	-76	10.3	50

Note: 1. Typical value
2. This field is active if LOWRES is cleared (ADC_MR)

33.6.2 Static Performance Characteristics

Table 33-25. Static Performance Characteristics 10-bit Mode INL, DNL

Parameter	Conditions	Min	Typ	Max	Unit
Native ADC Resolution		–	10	–	bit
Resolution with Digital Averaging	See ADC Controller	–	12	–	bit
Integral Non-linearity (INL)	VDDIO = 1.8V All temperature range, f_{ADC} =1 MHz	-3	±1	3	LSB
Differential Non-linearity (DNL)	VDDIO = 1.8V, all temperature ranges	-1	±0.5	1	LSB

Table 33-26. Static Performance Characteristics Gain and Error Offset 10-bit Mode

Parameter	Conditions	Min	Typ	Max	Unit
Offset Error	All temperature, f_{ADC} full range	-3	–	+3	LSB
Gain Error	All temperature, f_{ADC} full range	-5	–	+5	

33.6.3 Dynamic Performance Characteristics

Table 33-27. Dynamic Performance Characteristics in 10-bit Mode (1)

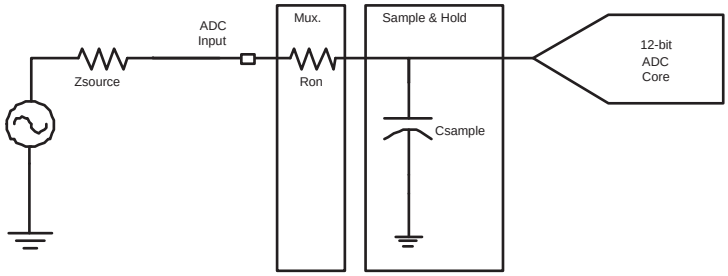
Parameter	Conditions	Min	Typ	Max	Unit
Signal to Noise Ratio - SNR		55	–	60	dB
Total Harmonic Distortion - THD		–	-70	-62	dB
Signal to Noise and Distortion - SINAD		53	59	–	dB
Effective Number of Bits ENOB		8.6	9.5	–	bits

Note: 1. ADC Clock (f_{ADC}) = 10 MHz, Fs=700 kS/s, f_{IN} = 2.2 kHz, IBCTL = 01, FFT using 1024 points or more, frequency band = [1 kHz, 500 kHz] – Nyquist conditions fulfilled.

33.6.4 Track and Hold Time versus Source Output Impedance

Figure 33-8 shows a simplified acquisition path.

Figure 33-8. Simplified Acquisition Path



During the tracking phase, the ADC must track the input signal during the tracking time shown below:

10-bit mode: $t_{TRACK} = 0.12 \times Z_{SOURCE} + 250$

With t_{TRACK} expressed in ns and Z_{SOURCE} expressed in Ohms.

Table 33-28. Analog Inputs

Parameter	Min	Typ	Max	Unit
Input Voltage Range	0	–	V_{DDIO}	
Input Capacitance	–	5	6	pF
Input Source Impedance	–	50	2000	Ω

Note: 1. Input Voltage range can be up to VDDIO without destruction or over-consumption. If VDDIO < V_{ADVREF} max input voltage is VDDIO.

33.7 AC Characteristics

33.7.1 Master Clock Characteristics

Table 33-29. Master Clock Waveform Parameters

Symbol	Parameter	Conditions	Min	Max	Unit
$1/(t_{CPMCK})$	Master Clock Frequency		–	48	MHz

33.7.2 I/O Characteristics

Criteria used to define the maximum frequency of the I/Os are:

- 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 33-30. I/O Characteristics

Symbol	Parameter	Conditions		Min	Max	Unit
FreqMax1	Pin Group 1 (1) Maximum Output Frequency	10 pF	V_{DDIO} = 1.62V	–	70	MHz
		30 pF	V_{DDIO} = 1.62V	–	45	
$PulseminH_1$	Pin Group 1 (1) High Level Pulse Width	10 pF	V_{DDIO} = 1.62V	7.2	–	ns
		30 pF	V_{DDIO} = 1.62V	11	–	
$PulseminL_1$	Pin Group 1 (1) Low Level Pulse Width	10 pF	V_{DDIO} = 1.62V	7.2	–	ns
		30 pF	V_{DDIO} = 1.62V	11	–	
FreqMax2	Pin Group 2 (2)Maximum Output Frequency	10 pF	V_{DDIO} = 1.62V	–	46	MHz
		25 pF	V_{DDIO} = 1.62V	–	23	
$PulseminH_2$	Pin Group 2 (2) High Level Pulse Width	10 pF	V_{DDIO} = 1.62V	11	–	ns
		25 pF	V_{DDIO} = 1.62V	21.8	–	
$PulseminL_2$	Pin Group 2 (2) Low Level Pulse Width	10 pF	V_{DDIO} = 1.62V	11	–	ns
		25 pF	V_{DDIO} = 1.62V	21.8	–	
FreqMax3	Pin Group3(3) Maximum Output Frequency	10 pF	V_{DDIO} = 1.62V	–	70	MHz
		25 pF	V_{DDIO} = 1.62V	–	35	
$PulseminH_3$	Pin Group 3 (3) High Level Pulse Width	10 pF	V_{DDIO} = 1.62V	7.2	–	ns
		25 pF	V_{DDIO} = 1.62V	14.2	–	
$PulseminL_3$	Pin Group 3 (3) Low Level Pulse Width	10 pF	V_{DDIO} = 1.62V	7.2	–	ns
		25 pF	V_{DDIO} = 1.62V	14.2	–	
FreqMax4	Pin Group 4(4)Maximum Output Frequency	10 pF	V_{DDIO} = 1.62V	–	58	MHz
		25 pF	V_{DDIO} = 1.62V	–	29	
$PulseminH_4$	Pin Group 4(4) High Level Pulse Width	10 pF	V_{DDIO} = 1.62V	8.6	–	ns
		25 pF	V_{DDIO} = 1.62V	17.2	–	
$PulseminL_4$	Pin Group 4(4) Low Level Pulse Width	10 pF	V_{DDIO} = 1.62V	8.6	–	ns
		25 pF	V_{DDIO} = 1.62V	17.2	–	

Notes: 1. Pin Group 1 = PA3, PA4, PA14

2. ,Pin Group 2 = PA5–PA6, PA9–PA13, PA15–PA24, PB0–PB7,PB10–PB12
3. Pin Group 3 =PA7,PA8,PB8,PB9
4. Pin Group 4 = PA0–PA2

33.7.3 SPI Characteristics

Figure 33-9. SPI Master Mode with (CPOL= NCPHA = 0) or (CPOL= NCPHA= 1)

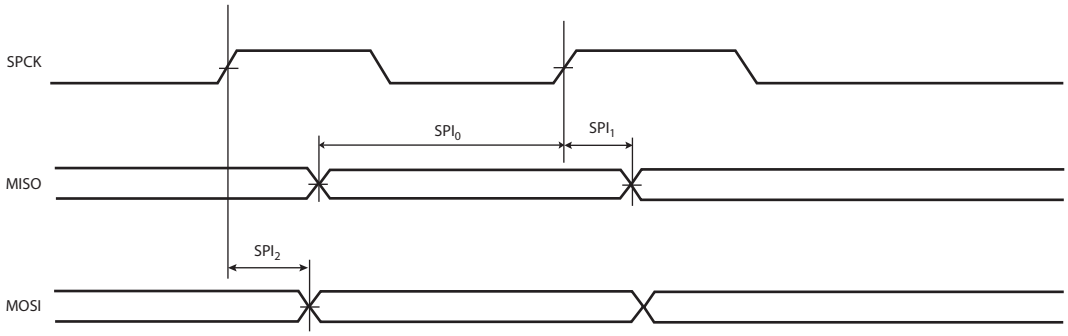


Figure 33-10.SPI Master Mode with (CPOL = 0 and NCPHA=1) or (CPOL=1 and NCPHA= 0)

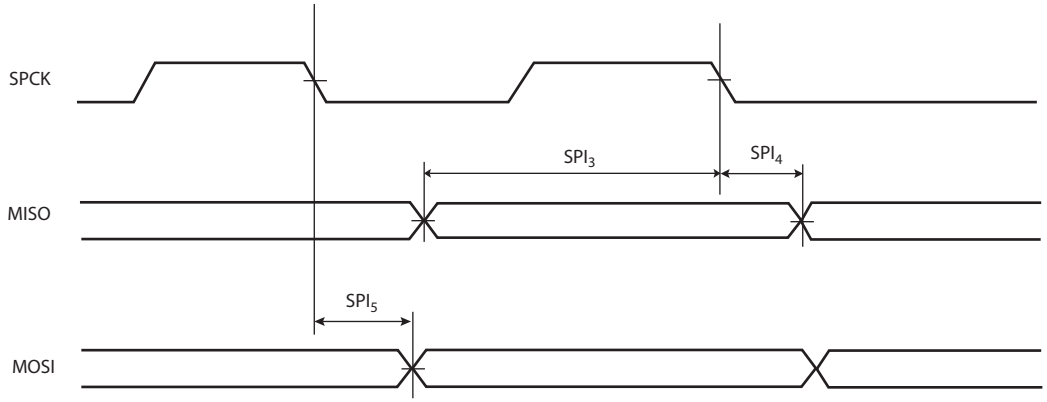


Figure 33-11.SPI Slave Mode with (CPOL = 0 and NCPHA = 1) or (CPOL = 1 and NCPHA = 0)

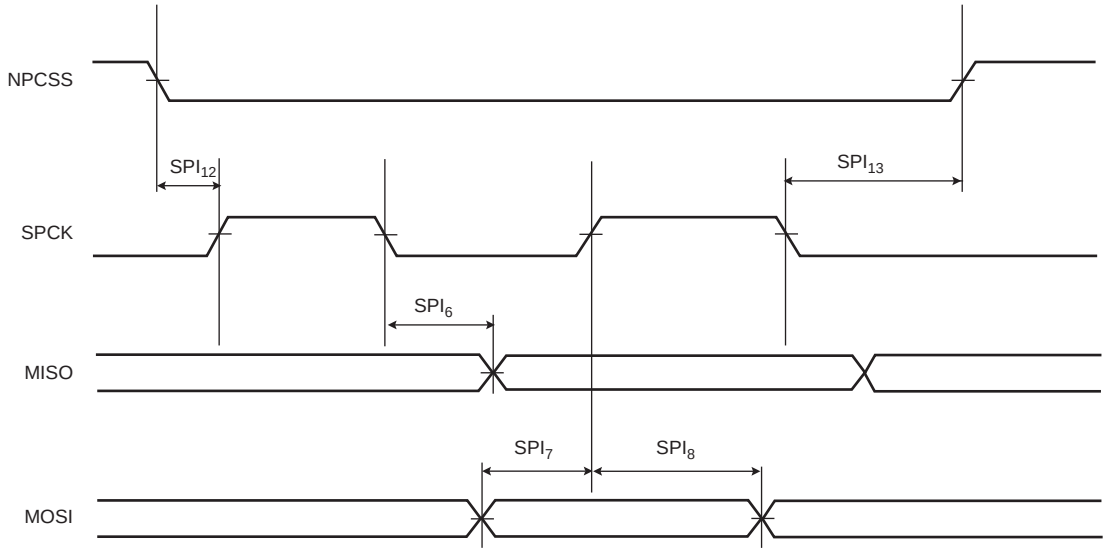
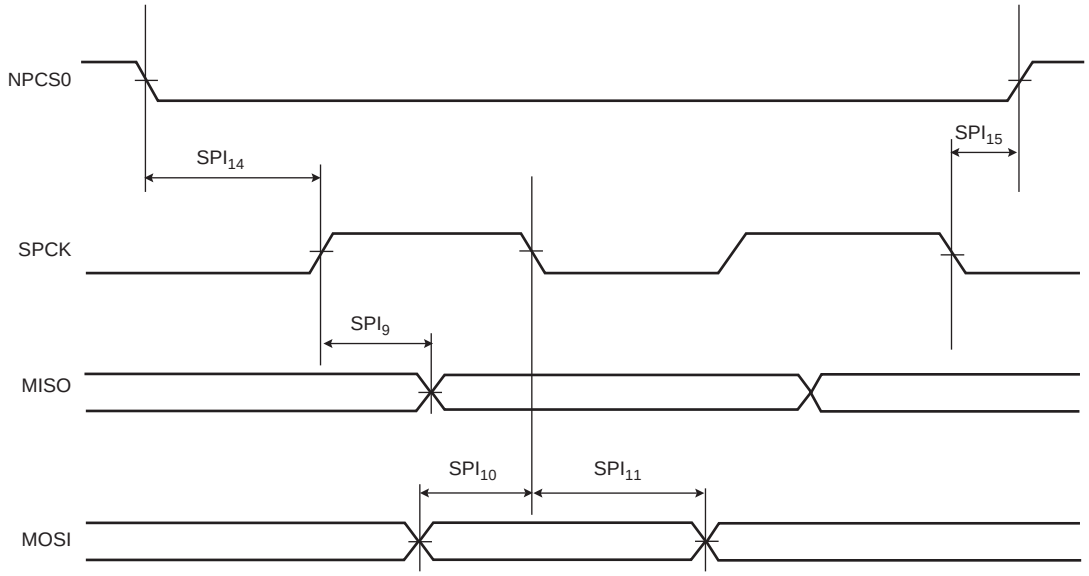


Figure 33-12.SPI Slave Mode with(CPOL = NCPHA = 0) or (CPOL = NCPHA =1)



33.7.3.1 Maximum SPI Frequency

The following formulas give maximum SPI frequency in master read and write modes and in slave read and write modes.

Master Write Mode

The SPI is only sending data to a slave device such as an LCD, for example. The limit is given by SPI_2 (or SPI_5) timing. Since it gives a maximum frequency above the maximum pad speed (see Section 33.7.2 “I/O 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 = 29.0 MHz @ VDDIO = 2V.

Slave Read Mode

In slave mode, SPCK is the input clock for the SPI. The max SPCK frequency is given by setup and hold timings SPI_7/SPI_8(or SPI_{10}/SPI_{11}). Since this gives a frequency well above the pad limit, the limit in slave read mode is given by SPCK pad.

Slave Write Mode

$$f_{SPCK}Max = \frac{1}{2x(SPI_{6max}(orSPI_{9max}) + t_{SETUP})}$$

For I/O domain and SPI6, f_{SPCK}Max = 20 MHz. t_{SETUP} is the setup time from the master before sampling data.

33.7.3.2 SPI Timings

Table 33-31. SPI Timings

Symbol	Parameter	Conditions	Min	Max	Unit
SPI_0	MISO setup time before SPCK rises (master)	1.8V domain	22.2	–	ns
SPI_1	MISO hold time after SPCK rises (master)	1.8V domain(1)	0	–	ns
SPI_2	SPCK rising to MOSI Delay (master)	1.8V domain(1)	-2.1	5.3	ns
SPI_3	MISO setup time before SPCK falls (master)	1.8V domain(1)	22.4	–	ns
SPI_4	MISO hold time after SPCK falls (master)	1.8V domain(1)	0	–	ns
SPI_5	SPCK falling to MOSI Delay (master)	1.8V domain(1)	-1.8	4.5	ns
SPI_6	SPCK falling to MISO Delay (slave)	1.8V domain(1)	6	24.6	ns
SPI_7	MOSI setup time before SPCK rises (slave)	1.8V domain(1)	0.3	–	ns
SPI_8	MOSI hold time after SPCK rises (slave)	1.8V domain(1)	4.3	–	ns
SPI_9	SPCK rising to MISO Delay (slave)	1.8V domain(1)	5.9	24.4	ns
SPI_{10}	MOSI setup time before SPCK falls (slave)	1.8V domain(1)	1.8	–	ns
SPI_{11}	MOSI hold time after SPCK falls (slave)	1.8V domain(1)	3.2	–	ns
SPI_{12}	NPCS setup to SPCK rising (slave)	1.8V domain(1)	6.5	–	ns
SPI_{13}	NPCS hold after SPCK falling (slave)	1.8V domain(1)	0	–	ns
SPI_{14}	NPCS setup to SPCK falling (slave)	1.8V domain(1)	7.9	–	ns
SPI_{15}	NPCS hold after SPCK falling (slave)	1.8V domain(1)	0	–	ns

Notes: 1. 1.8V domain: V_{VDDIO} from 1.65V to 2V, maximum external capacitor = 20 pF.

2. Note that in SPI master mode the SAM G51 does not sample the data (MISO) on the opposite edge where data clocks out (MOSI). Data sampling is done on the same edge. This is shown in Figure 33-9 and Figure 33-10.

33.7.4 USART in SPI Mode Timings

Timings are given in the following domain:

1.8V domain: VDDIO from 1.65V to 2V, maximum external capacitor = 20 pF

Figure 33-13.USART SPI Master Mode

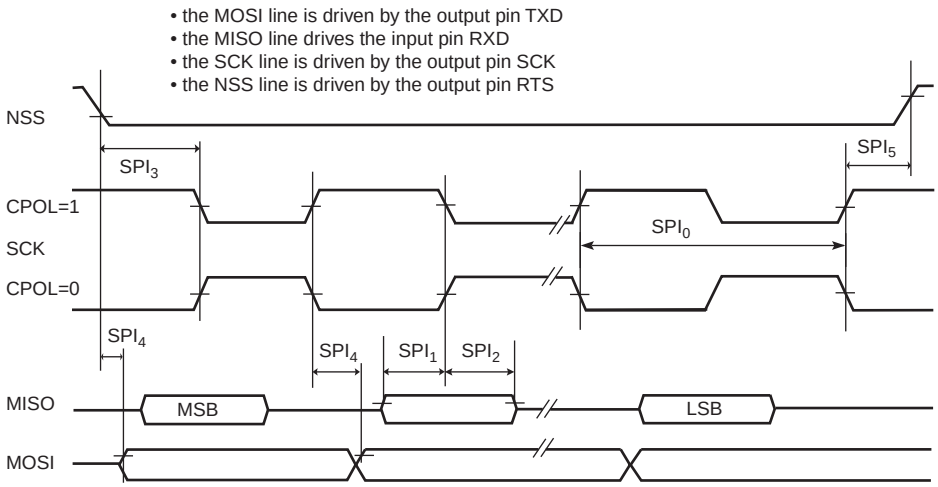


Figure 33-14. USART SPI Slave Mode (Mode 1 or 2) r

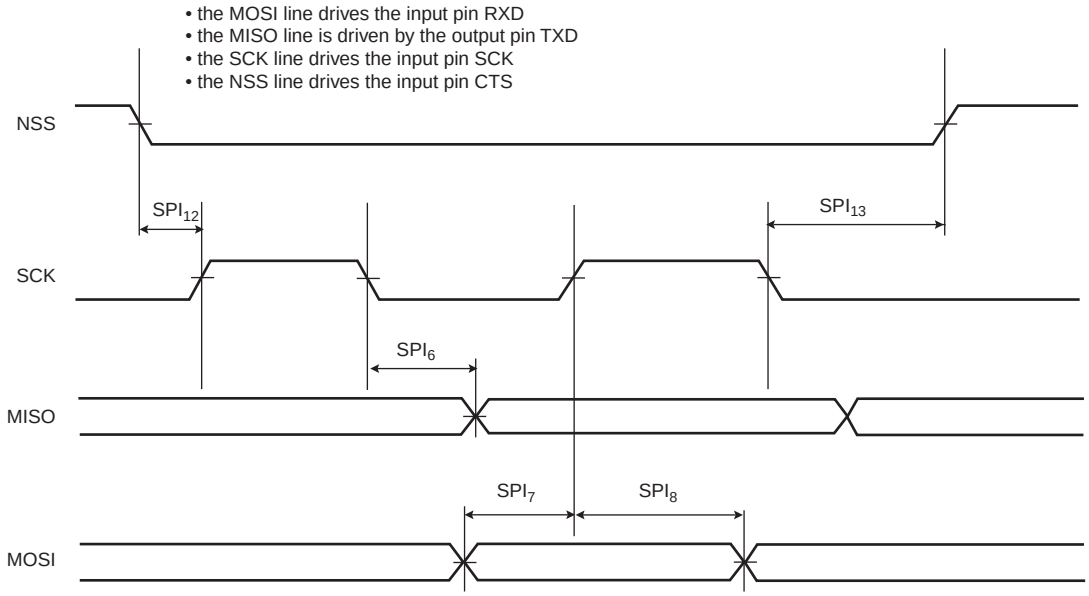
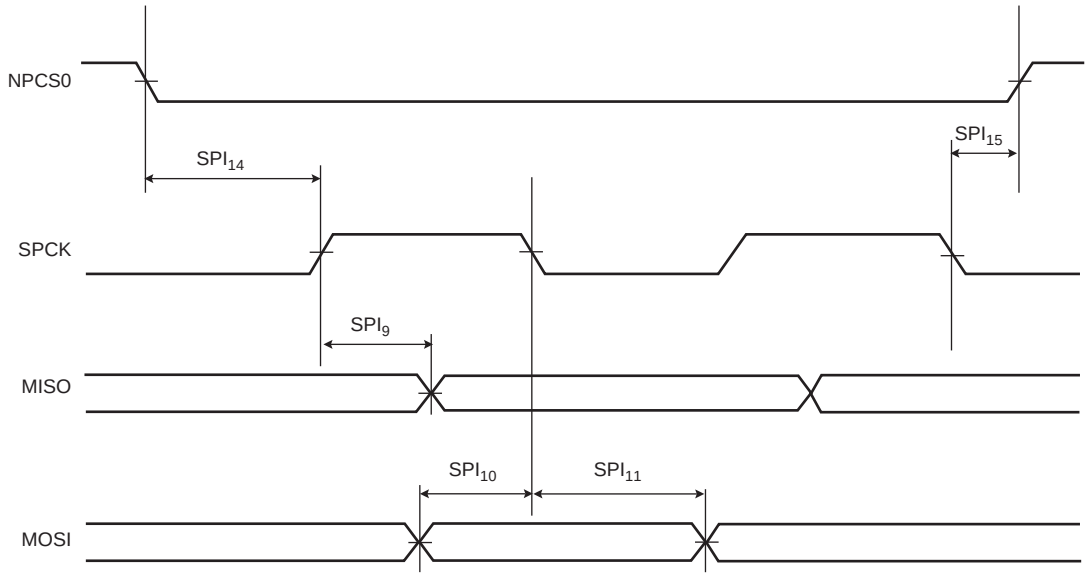


Figure 33-15. USART SPI Slave Mode (Mode 0 or 3) r



33.7.4.1 USART SPI TImings

Table 33-32. USART SPI Timings

Symbol	Parameter	Conditions	Min	Max	Unit
Master Mode					
SPI_0	SCK Period	1.8V domain	MCK/6	–	ns
SPI_1	Input Data Setup Time	1.8V domain	6.1	–	ns
SPI_2	Input Data Hold Time	1.8V domain	0	–	ns
SPI_3	Chip Select Active to Serial Clock	1.8V domain	-5	–	ns
SPI_4	Output Data Setup Time	1.8V domain	6.3	20.3	ns
SPI_5	Serial Clock to Chip Select Inactive	1.8V domain	0	–	ns
Slave Mode					
SPI_6	SCK falling to MISO	1.8V domain	5.2	22.1	ns
SPI_7	MOSI Setup time before SCK rises	1.8V domain	3.1	–	ns
SPI_8	MOSI Hold time after SCK rises	1.8V domain	0	–	ns
SPI_9	SCK rising to MISO	1.8V domain	5.2	22	ns
SPI_{10}	MOSI Setup time before SCK falls	1.8V domain	4.5	–	ns
SPI_{11}	MOSI Hold time after SCK falls	1.8V domain	0.1	–	ns
SPI_{12}	NPCS0 setup to SCK rising	1.8V domain	0.7	–	ns
SPI_{13}	NPCS0 hold after SCK falling	1.8V domain	0.8	–	ns
SPI_{14}	NPCS0 setup to SCK falling	1.8V domain	0.8	–	ns
SPI_{15}	NPCS0 hold after SCK rising	1.8V domain	0.4	–	ns

Notes: 1. 1.8V domain: VDDIO from 1.65V to 2V, maximum external capacitor = 25 pF

33.7.5 Two-wire Serial Interface Characteristics

Table 33-33 “Two-wire Serial Bus Requirements” shows the requirements for devices connected to the Two-wire Serial Bus and the compliance of the device with them.. For timing symbols refer to Figure 33-16.

Table 33-33. Two-wire Serial Bus Requirements

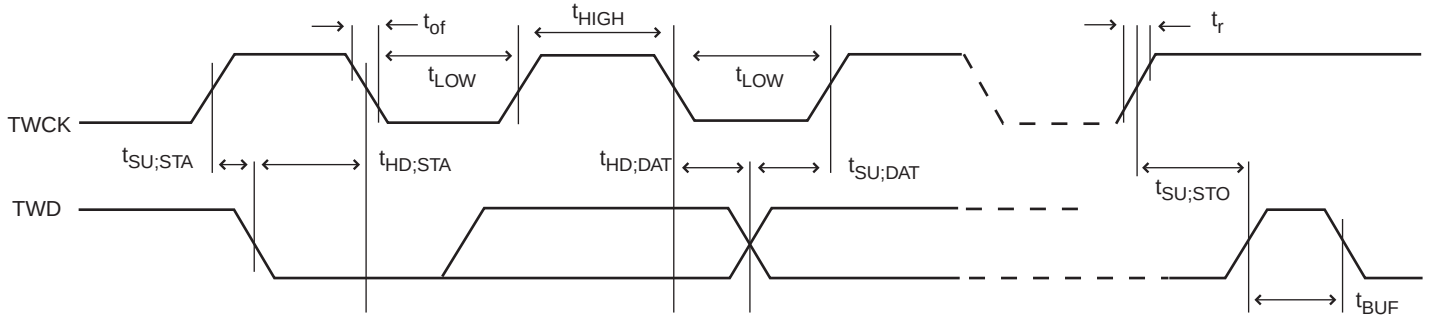
Symbol	Parameter	Condition	Min	Max	Unit
V_{IL}	Input Low-voltage		-0.3	0.3 V_{VDDIO}	V
V_{IH}	Input High-voltage		0.7xV_{VDDIO}	V_{DDIO} + 0.3	V
V_{HYS}	Hysteresis of Schmitt Trigger Inputs		0.150	–	V
V_{OL}	Output Low-voltage	3 mA sink current	–	0.4	V
t_R	Rise Time for both TWD and TWCK		20 + 0.1C_b (1)(2)	300	ns
t_{OF}	Output Fall Time from V_{IHmin} to V_{ILmax}	10 pF < C_b < 400 pF Figure 33-16	20 + 0.1C_b (1)(2)	250	ns
C_i (1)	Capacitance for each I/O Pin		–	10	pF
f_{TWCK}	TWCK Clock Frequency		0	400	kHz

Table 33-33. Two-wire Serial Bus Requirements (Continued)

Symbol	Parameter	Condition	Min	Max	Unit
Rp	Value of Pull-up Resistor	$f_{TWCK} \le 100$ kHz	$\frac{V_{VDDIO} - 0.4V}{3mA}$	$\frac{1000ns}{C_b}$	Ω
		$f_{TWCK} > 100$ kHz	$\frac{V_{VDDIO} - 0.4V}{3mA}$	$\frac{300ns}{C_b}$	Ω
t_{LOW}	Low Period of the TWCK Clock	$f_{TWCK} \le 100$ kHz	(3)	–	µs
		$f_{TWCK} > 100$ kHz	(3)	–	µs
t_{HIGH}	High Period of the TWCK Clock	$f_{TWCK} \le 100$ kHz	(4)	–	µs
		$f_{TWCK} > 100$ kHz	(4)	–	µs
$t_{HD;STA}$	Hold Time (repeated) START Condition	$f_{TWCK} \le 100$ kHz	t_{HIGH}	–	µs
		$f_{TWCK} > 100$ kHz	t_{HIGH}	–	µs
$t_{SU;STA}$	Set-up Time for a Repeated Start Condition	$f_{TWCK} \le 100$ kHz	t_{HIGH}	–	µs
		$f_{TWCK} > 100$ kHz	t_{HIGH}	–	µs
$t_{HD;DAT}$	Data Hold Time	$f_{TWCK} \le 100$ kHz	0	$3 \times t_{CP\_MCK}$ (5)	µs
		$f_{TWCK} > 100$ kHz	0	$3 \times t_{CP\_MCK}$ (5)	µs
$t_{SU;DAT}$	Data Setup Time	$f_{TWCK} \le 100$ kHz	$t_{LOW} - 3 \times t_{CP\_MCK}$ (5)	–	ns
		$f_{TWCK} > 100$ kHz	$t_{LOW} - 3 \times t_{CP\_MCK}$ (5)	–	ns
$t_{SU;STO}$	Setup Time for STOP Condition	$f_{TWCK} \le 100$ kHz	t_{HIGH}	–	µs
		$f_{TWCK} > 100$ kHz	t_{HIGH}	–	µs
$t_{HD;STA}$	Hold Time (repeated) START Condition	$f_{TWCK} \le 100$ kHz	t_{HIGH}	–	µs
		$f_{TWCK} > 100$ kHz	t_{HIGH}	–	µs

Notes:
1. Required only for $f_{TWCK} > 100$ kHz.
2. C_b = Capacitance of one bus line in pF. Per I^2C standard, C_b max = 400 pF
3. The TWCK low period is defined as follows: $t_{LOW} = ((CLDIV \times 2^{CKDIV}) + 4) \times t_{MCK}$
4. The TWCK high period is defined as follows: $t_{HIGH} = ((CHDIV \times 2^{CKDIV}) + 4) \times t_{MCK}$
5. t_{CP_MCK} = MCK Bus Period.

Figure 33-16.Two-wire Serial Bus Timing



33.7.6 High-Speed Two-wire Serial Interface Characteristics

High-speed TWI requires an MCK at a minimum frequency of 21 MHz. The Ccode which handles the HS TWI application must be located in Flash memory.

Table 33-34 “Two-wire Serial Bus Requirements” describes the requirements for devices connected to the Two-wire Serial Bus. For timing symbols refer to Figure 33-16.

Table 33-34. Two-wire Serial Bus Requirements

Symbol	Parameter	Condition	Min	Max	Unit
f_{TWCK}	TWCK Clock Frequency	Capacitive load C_b = 100 pF (max) C_b = 400 pF(2)	–	 3.4 1.7	MHz
V_{IL}	Low-voltage Input		-0.3	0.3 V_{VDDIO}	V
V_{IH}	High-voltage Input		0.7xV_{VDDIO}	V_{DDIO} + 0.3	V
V_{HYS}	Hysteresis of Schmitt Trigger Inputs		0.1xV_{VDDIO}	–	V
V_{OL}	Low-voltage Output		–	0.2V_{DDIO}	V
t_{Rd}	Rise Time for both TWD/TWCK(After Acknowledge)	Capacitive load C_b = 100 pF (max) C_b = 400 pF(2)	 10 20	 80 160	ns
t_{OF}	Output Fall Time from V_{IHmin} to V_{ILmax}	Capacitive load C_b = 100 pF (max) C_b = 400 pF(2)	 10 20	 40 80	ns
C_i(1)	Capacitance for Each I/O Pin		–	10	pF
R_P	Value of Pull-up Resistor		$\frac{V_{VDDIO} - 0.4V}{3\ mA}$	$\frac{1000ns}{C_b}$	Ω
		z	$\frac{V_{VDDIO} - 0.4V}{3\ mA}$	$\frac{300ns}{C_b}$	Ω
$t_{HD;DAT}$	Data Hold Time	Capacitive load from 10 pF to 100 pF	0	70	µs
		Capacitive load C_b = 400 pF(2)	0	150	µs
$t_{SU;DAT}$	Data Setup Time	Capacitive load from 10 pF to 100 pF	10	–	ns
		Capacitive load C_b = 400 pF(2)	10	–	ns

Notes: 1. Required only for f_{TWCK} > 100 kHz.
2. For bus line loads C_b between 100 pF and 400 pF the timing parameters must be linearly interpolated.

33.7.7 Embedded Flash Characteristics

The maximum operating frequency is given in Table 33-35, “Embedded Flash Wait State at VDDIO 1.62V (Max Value),” below but is limited by the embedded Flash access time when the processor is fetching code out of it. Table 33-35 “Embedded Flash Wait State at VDDIO 1.62V (Max Value)” and Table 33-36 “Embedded Flash Wait State at VDDIO 1.8V (Typical Value)” below give the device maximum and typical operating frequency depending on the field FWS of the MC_FMR register. This field defines the number of wait states required to access the embedded Flash memory.

Table 33-35. Embedded Flash Wait State at VDDIO 1.62V (Max Value)

FWS	Read Operations	Maximum Operating Frequency (MHz)
0	1 cycle	12
1	2 cycles	24
2	3 cycles	36
3	4 cycles	48

Table 33-36. Embedded Flash Wait State at VDDIO 1.8V (Typical Value)

FWS	Read Operations	Maximum Operating Frequency (MHz)
0	1 cycle	15
1	2 cycles	30
2	3 cycles	45
3	4 cycles	60

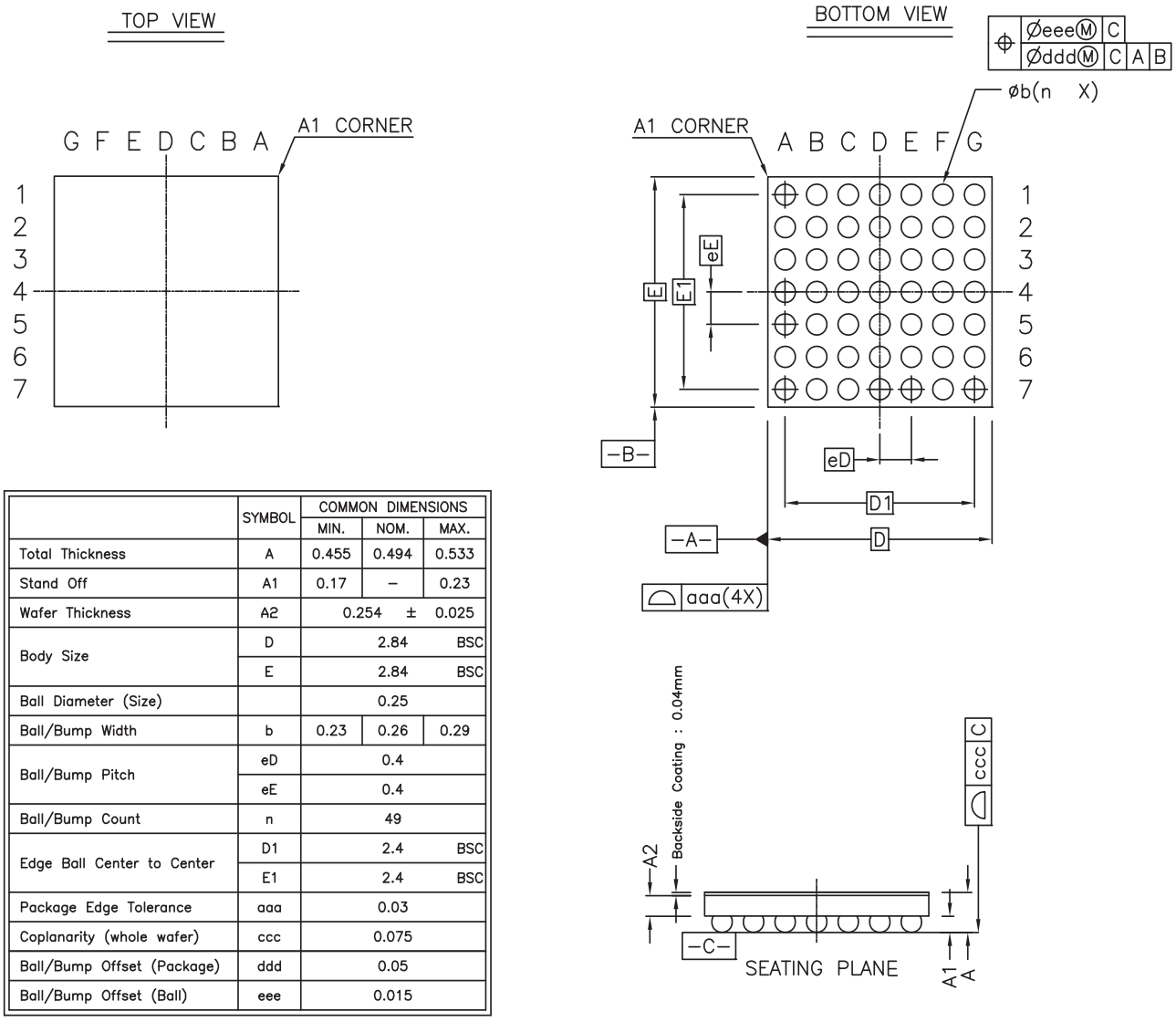
Table 33-37. AC Flash Characteristics

Parameter	Conditions	Min	Typ	Max	Unit
Program/Erase Cycle Time	Write page (512 bytes)	–	1.5	3	ms
	Write word – 64-bit word – 128-bit word	 – –	 20 40	 40 80	µs
	Erase page mode	–	10	50	ms
	Erase block mode (by 4 KBytes)	–	50	200	ms
	Erase sector mode (Sector of 64 KBytes)	–	400	950	ms
	Lock/Unlock Time per region	–	1.5	3	ms
Full Chip Erase	256 Kbytes	–	3	6	s
Data Retention	Not powered or powered	–	20		years
Endurance	Write/Erase cycles per page, block or sector @ 85°C	10K	–	–	cycles

34. SAM G51 Mechanical Characteristics

34.1 49-lead WLCSP Package

Figure 34-1. 49-lead WLCSP Package Mechanical Drawing



	SYMBOL	COMMON DIMENSIONS		
		MIN.	NOM.	MAX.
Total Thickness	A	0.455	0.494	0.533
Stand Off	A1	0.17	–	0.23
Wafer Thickness	A2		0.254 ± 0.025	
Body Size	D		2.84 BSC	
	E		2.84 BSC	
Ball Diameter (Size)			0.25	
Ball/Bump Width	b	0.23	0.26	0.29
Ball/Bump Pitch	eD		0.4	
	eE		0.4	
Ball/Bump Count	n		49	
Edge Ball Center to Center	D1		2.4 BSC	
	E1		2.4 BSC	
Package Edge Tolerance	aaa		0.03	
Coplanarity (whole wafer)	ccc		0.075	
Ball/Bump Offset (Package)	ddd		0.05	
Ball/Bump Offset (Ball)	eee		0.015	

Table 34-1. Device and WLCSP Package Maximum Weight

SAM G51G18	10	mg

Table 34-2. Package Reference

JEDEC Drawing Reference	na
JESD97 Classification	e1

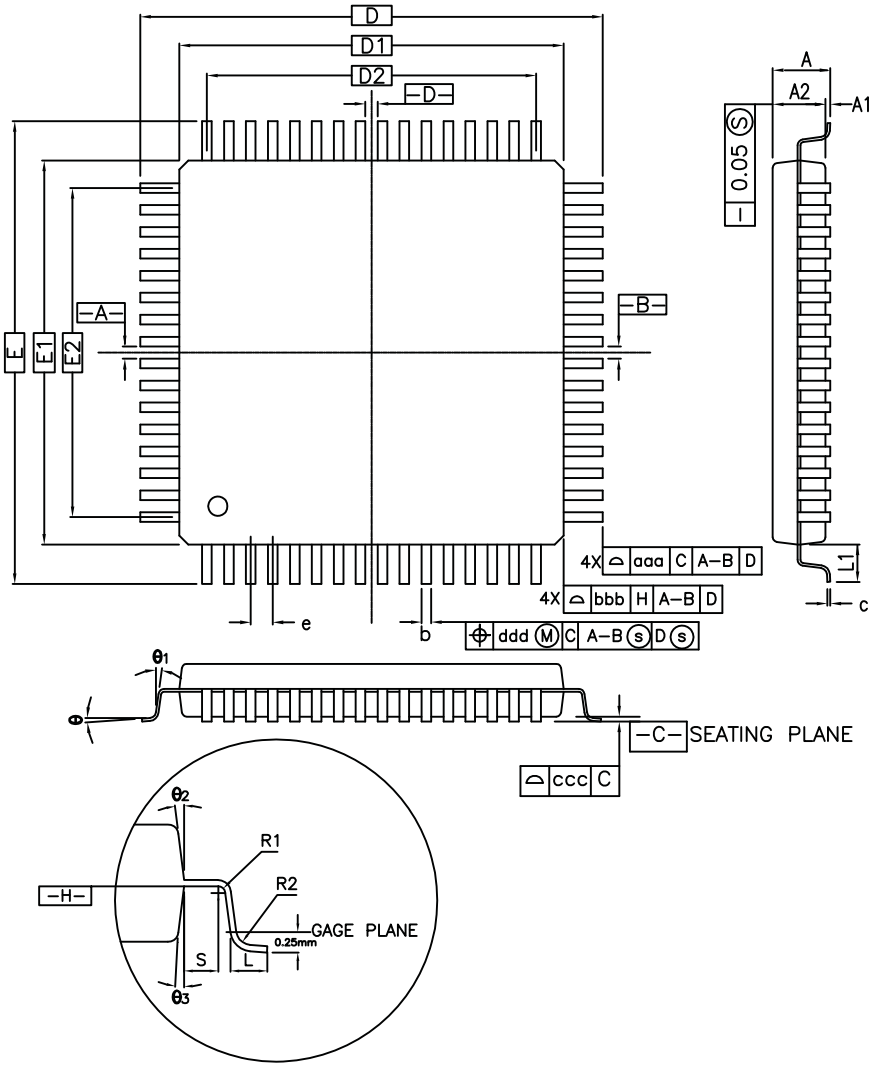
Table 34-3. WLCSP Package Characteristics

Moisture Sensitivity Level	1

This package respects the recommendations of the NEMI User Group.

34.2 100-lead LQFP Package

Figure 34-2. 100-lead LQFP Package Mechanical Drawing



COTROL 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	16.00 BSC.			0.630 BSC.		
D_1	14.00 BSC.			0.551 BSC.		
E	16.00 BSC.			0.630 BSC.		
E_1	14.00 BSC.			0.551 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	100L					
	MILLIMETER			INCH		
	MIN.	NOM.	MAX.	MIN.	NOM.	MAX.
b	0.17	0.20	0.27	0.007	0.008	0.011
e	0.50 BSC.			0.020 BSC.		
D2	12.00			0.472		
E2	12.00			0.472		
aaa	0.20			0.008		
bbb	0.20			0.008		
ccc	0.08			0.003		
ddd	0.08			0.003		

Table 34-4. Device and LQFP Package Maximum Weight

SAM G51N18	675	mg

Table 34-5. Package Reference

JEDEC Drawing Reference	MS-026
JESD97 Classification	e3

Table 34-6. LQFP Package Characteristics

Moisture Sensitivity Level	3

This package respects the recommendations of the NEMI User Group.

34.3 Soldering Profile

Table 34-7 gives the recommended soldering profile from J-STD-020C.

Table 34-7. 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.

34.4 Packaging Resources

Land Pattern Definition.

Refer to the following IPC Standards:

- IPC-7351A and IPC-782 (*Generic Requirements for Surface Mount Design and Land Pattern Standards*) http://landpatterns.ipc.org/default.asp
- Atmel Green and RoHS Policy and Package Material Declaration Data Sheet http://www.atmel.com/green/

35. Ordering Information

Table 35-1. Ordering Codes for SAM G51 Devices

Ordering Code	MRL	Flash (Kbytes)	Package (Kbytes)	Package Type	Temperature Operating Range
SAMG51G18A-UUT	A	256	WLCSP49	Green	Industrial -40°C to 85°C
SAMG51N18A-AU	A	256	LQFP100	Green	Industrial -40°C to 85°C

36. Revision History

In the tables that follow, the most recent version of the document appears first.

Table 36-1. SAM G51 Datasheet Rev. 11209C 20-Dec-13 Revision History

Doc. Date	Change
20-Dec-13	First public issue.

Table 36-2. SAM G51 Datasheet Rev. 11209B 23-Jul-13 Revision History

Doc. Date	Changes
23-Jul-13	Atmel internal document.

Table 36-3. SAM G51 Datasheet Rev. 11209A Revision History

Doc. Date	Changes
08-Mar-13	Atmel internal document.

Table of Contents

Description 1

Features 2

1. Configuration Summary 3
2. SAM G51 Block Diagram 4
3. Signal Description 5
4. Package and Pinout 7
   - 4.1 49-ball WLCSP Pinout 7
   - 4.2 100-lead LQFP Pinout 8
5. Power Considerations 9
   - 5.1 Power Supplies 9
   - 5.2 Voltage Regulator 9
   - 5.3 Typical Powering Schematics 9
   - 5.4 Functional Modes 10
   - 5.5 Fast Start-up 12
6. Processor and Architecture 13
   - 6.1 ARM Cortex-M4 Processor 13
   - 6.2 APB/AHB Bridge 13
   - 6.3 Peripheral DMA Controller 13
   - 6.4 Debug and Test Features 14
   - 6.5 Product Mapping 15
7. Memories 16
   - 7.1 Embedded Memories 16
8. System Controller 19
   - 8.1 System Controller and Peripherals Mapping 19
   - 8.2 Power-on Reset, Supply Monitor 19
   - 8.3 Reset Controller 19
   - 8.4 Supply Controller 19
   - 8.5 Clock Generator 20
   - 8.6 Power Management Controller 21
   - 8.7 Watchdog Timer 21
   - 8.8 SysTick Timer 22
   - 8.9 Real-Time Timer 22
   - 8.10 Real Time Clock 22
   - 8.11 General-Purpose Backup Registers 22
   - 8.12 Nested Vectored Interrupt Controller 22
   - 8.13 Chip Identification 22
   - 8.14 PIO Controllers 23
   - 8.15 Peripheral Identifiers 24
   - 8.16 Peripherals Signals Multiplexing on I/O Lines 26
9. Real-Time Event Management 29

9.1 Embedded Characteristics 29
9.2 Real-Time Event Mapping 29

10. ARM Cortex-M4 Processor 30
10.1 Description 30
10.2 Embedded Characteristics 31
10.3 Block Diagram 31
10.4 Cortex-M4 Models 32
10.5 Power Management 60
10.6 Cortex-M4 Instruction Set 62
10.7 Cortex-M4 Core Peripherals 209
10.8 Nested Vectored Interrupt Controller (NVIC) 210
10.9 System Control Block (SCB) 220
10.10 System Timer (SysTick) 246
10.11 Memory Protection Unit (MPU) 252
10.12 Floating Point Unit (FPU) 275
10.13 Glossary 284

11. Debug and Test Features 289
11.1 Description 289
11.2 Embedded Characteristics 289
11.3 Application Examples 290
11.4 Debug and Test Pin Description 291
11.5 Functional Description 292

12. Reset Controller (RSTC) 297
12.1 Description 297
12.2 Embedded Characteristics 297
12.3 Block Diagram 297
12.4 Functional Description 298
12.5 Reset Controller (RSTC) User Interface 305

13. Real-time Timer (RTT) 309
13.1 Description 309
13.2 Embedded Characteristics 309
13.3 Block Diagram 309
13.4 Functional Description 310
13.5 Real-time Timer (RTT) User Interface 312

14. Real-time Clock (RTC) 317
14.1 Description 317
14.2 Embedded Characteristics 317
14.3 Block Diagram 318
14.4 Product Dependencies 318
14.5 Functional Description 318
14.6 Real-time Clock (RTC) User Interface 324

15. Watchdog Timer (WDT) 338
15.1 Description 338
15.2 Embedded Characteristics 338
15.3 Block Diagram 339
15.4 Functional Description 340

15.5 Watchdog Timer (WDT) User Interface 342

16. Supply Controller (SUPC) 347
16.1 Description 347
16.2 Embedded Characteristics 347
16.3 Block Diagram 348
16.4 Supply Controller Functional Description 349
16.5 Supply Controller (SUPC) User Interface 354

17. General-Purpose Backup Registers (GPBR) 360
17.1 Description 360
17.2 Embedded Characteristics 360
17.3 General Purpose Backup Registers (GPBR) User Interface 361

18. Memory to Memory (MEM2MEM) 363
18.1 Description 363
18.2 Embedded Characteristics 363
18.3 Block Diagram 364
18.4 Product Dependencies 364
18.5 Functional Description 365
18.6 Memory to Memory (MEM2MEM) User Interface 366

19. **Enhanced Embedded Flash Controller (EEFC) 373**
19.1 **Description 373**
19.2 **Embedded Characteristics 373**
19.3 Product Dependencies 373
19.4 Functional Description 374
19.5 Enhanced Embedded Flash Controller (EEFC) User Interface 390

20. Bus Matrix (MATRIX) 396
20.1 Description 396
20.2 Embedded Characteristics 396
20.3 Master/Slave Management 396
20.4 Memory Mapping 397
20.5 Special Bus Granting Techniques 397
20.6 Arbitration 398
20.7 System I/O Configuration 399
20.8 Register Write Protection 400
20.9 Bus Matrix (MATRIX) User Interface 401

21. Peripheral DMA Controller (PDC) 408
21.1 Description 408
21.2 Embedded Characteristics 408
21.3 Block Diagram 409
21.4 Functional Description 410
21.5 Peripheral DMA Controller (PDC) User Interface 412

22. Clock Generator 423
22.1 Description 423
22.2 Embedded Characteristics 423
22.3 Block Diagram 424
22.4 Slow Clock 425

22.5 Main Clock 426
22.6 Divider and PLL Block. 430

23. Power Management Controller (PMC) 431
23.1 Description 431
23.2 Embedded Characteristics 431
23.3 Block Diagram. 432
23.4 Master Clock Controller 432
23.5 Processor Clock Controller 433
23.6 SysTick Clock 433
23.7 Peripheral Clock Controller 433
23.8 Free-Running Processor Clock. 433
23.9 Programmable Clock Output Controller 433
23.10 Fast Startup 434
23.11 Startup from Embedded Flash 435
23.12 Main Clock Failure Detector 435
23.13 Programming Sequence 436
23.14 Clock Switching Details. 439
23.15 Register Write Protection 441
23.16 Power Management Controller (PMC) User Interface 442

24. Chip Identifier (CHIPID) 467
24.1 Description 467
24.2 Embedded Characteristics 467
24.3 Chip Identifier (CHIPID) User Interface 468

25. Parallel Input/Output Controller (PIO) 473
25.1 Description 473
25.2 Embedded Characteristics 473
25.3 Block Diagram. 474
25.4 Product Dependencies 475
25.5 Functional Description 476
25.6 I/O Lines Programming Example 484
25.7 Parallel Input/Output Controller (PIO) User Interface 485

26. Serial Peripheral Interface (SPI) 521
26.1 Description 521
26.2 Embedded Characteristics 522
26.3 Block Diagram. 523
26.4 Application Block Diagram 523
26.5 Signal Description 524
26.6 Product Dependencies 524
26.7 Functional Description 525
26.8 Serial Peripheral Interface (SPI) User Interface 539

27. Two-Wire Interface (TWIHS) 555
27.1 Description 555
27.2 Embedded Characteristics 555
27.3 List of Abbreviations 556
27.4 Block Diagram. 556
27.5 Application Block Diagram 557
27.6 Product Dependencies 558

27.7 Functional Description 559
27.8 Two-wire Interface High Speed (TWIHS) User Interface 596

28. Two-wire Interface (TWI) 619
28.1 Description 619
28.2 Embedded Characteristics 619
28.3 List of Abbreviations 620
28.4 Block Diagram 620
28.5 Application Block Diagram 621
28.6 Product Dependencies 621
28.7 Functional Description 622
28.8 Two-wire Interface (TWI) User Interface 646

29. Universal Asynchronous Receiver Transmitter (UART) 663
29.1 Description 663
29.2 Embedded Characteristics 663
29.3 Block Diagram 663
29.4 Product Dependencies 664
29.5 UART Operations 665
29.6 Universal Asynchronous Receiver Transmitter (UART) User Interface 671

30. Universal Synchronous Asynchronous Receiver Transceiver (USART) 682
30.1 Description 682
30.2 Embedded Characteristics 682
30.3 Block Diagram 683
30.4 Application Block Diagram 684
30.5 I/O Lines Description 685
30.6 Product Dependencies 686
30.7 Functional Description 687
30.8 Universal Synchronous Asynchronous Receiver Transmitter (USART) User Interface 712

31. Timer Counter (TC) 742
31.1 Description 742
31.2 Embedded Characteristics 742
31.3 Block Diagram 743
31.4 Pin Name List 744
31.5 Product Dependencies 744
31.6 Functional Description 745
31.7 Timer Counter (TC) User Interface 759

32. Analog-to-Digital Converter (ADC) 784
32.1 Description 784
32.2 Embedded Characteristics 785
32.3 Block Diagram 786
32.4 Signal Description 786
32.5 Product Dependencies 787
32.6 Functional Description 788
32.7 Analog-to-Digital Converter (ADC) User Interface 800

33. SAM G51 Electrical Characteristics 824

33.1 Absolute Maximum Ratings 824
33.2 DC Characteristics 825
33.3 Power Consumption 830
33.4 Oscillator Characteristics 835
33.5 PLL Characteristics 842
33.6 12-Bit ADC Characteristics 843
33.7 AC Characteristics 846

34. SAM G51 Mechanical Characteristics 855
34.1 49-lead WLCSP Package 855
34.2 100-lead LQFP Package 856
34.3 Soldering Profile 857
34.4 Packaging Resources 857

35. Ordering Information 858

36. Revision History 859

Table of Contents i

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation
1600 Technology Drive
San Jose, CA 95110
USA
Tel: (+1) (408) 441-0311
Fax: (+1) (408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan G.K.
16F Shin-Osaki Kangyo Bldg
1-6-4 Osaki, Shinagawa-ku
Tokyo 141-0032
JAPAN
Tel: (+81) (3) 6417-0300
Fax: (+81) (3) 6417-0370