SAM G55G / SAM G55J

Atmel | SMART ARM-based Flash MCU

DATASHEET

Description

The Atmel® | SMART SAM G55 is a series of Flash microcontrollers based on the high-performance 32-bit ARM® Cortex®-M4 RISC processor with FPU (Floating Point Unit). It operates at a maximum speed of 120 MHz and features 512 Kbytes of Flash and up to 176 Kbytes of SRAM. The peripheral set includes eight flexible communication units comprising USARTs, SPIs and I^2C-bus interfaces (TWIs), two three-channel general-purpose 16-bit timers, two I^2S controllers, one-channel pulse density modulation, one 8-channel 12-bit ADC, one real-time timer (RTT) and one real-time clock (RTC), both located in the ultra low-power backup area.

The Atmel | SMART SAM G55 devices have three software-selectable low-power modes: Sleep, Wait and Backup. In Sleep mode, the processor is stopped while all other functions can be kept running. In Wait mode, all clocks and functions are stopped but some peripherals can be configured to wake up the system based on events, including partial asynchronous wakeup (SleepWalking™). In Backup mode, RTT, RTC and wakeup logic are running.

For power consumption optimization, the flexible clock system offers the capability of having different clock frequencies for some peripherals. Moreover, the processor and bus clock frequency can be modified without affecting the peripheral processing.

The real-time event management allows peripherals to receive, react to and send events in Active and Sleep modes without processor intervention.

The SAM G55 devices are general-purpose low-power microcontrollers that offer high performance, processing power and small package options combined with a rich and flexible peripheral set. With this unique combination of features, the SAM G55 series is suitable for a wide range of applications including consumer, industrial control and PC peripherals.

The device operates from 1.62V to 3.6V and is available in three packages: 49-pin WLCSP, 64-pin QFN and 64-pin LQFP.

Features

- Core
  - ARM Cortex-M4 with up to 16 Kbytes SRAM on I/D bus providing 0 wait state execution at up to 120 MHz (1)
  - Memory Protection Unit (MPU)
  - DSP Instructions
  - Floating Point Unit (FPU)
  - Thumb®-2 instruction set

Note: 1. 120 MHz with $V_{DDCOREXT120}$ or with V_{DDCORE} trimmed by regulator.

- Memories
  - Up to 512 Kbytes embedded Flash
  - Up to 176 Kbytes embedded SRAM
  - 8 Kbytes ROM with embedded boot loader, single-cycle access at full speed
- System
  - Embedded voltage regulator for single-supply operation
  - Power-on reset (POR) and Watchdog for safe operation
  - Quartz or ceramic resonator oscillators: 3 to 20 MHz with clock failure detection and 32.768 kHz for RTT or system clock
  - High-precision 8/16/24 MHz 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 48 MHz to 120 MHz for device clock
  - PLL range from 24 MHz to 48 MHz for USB device and USB OHCI
  - Up to 30 peripheral DMA (PDC) channels
  - 256-bit General-Purpose Backup Registers (GPBR)
  - 16 external interrupt lines
- Peripherals
  - 8 flexible communication units supporting:
    - USART
    - SPI
    - Two-wire Interface (TWI) featuring TWI masters and high-speed TWI slaves
  - Crystal-less USB 2.0 Device and USB Host OHCI with On-chip Transceiver
  - 2 Inter-IC Sound Controllers (I^2S)
  - 1 Pulse Density Modulation Interface (PDMIC) (supports up to two microphones)
  - 2 three-channel 16-bit Timer/Counters (TC) with capture, waveform, compare and PWM modes
  - 1 48-bit Real-Time Timer (RTT) with 16-bit prescaler and 32-bit counter
  - 1 RTC with calendar and alarm features
  - 1 32-bit Cyclic Redundancy Check Calculation Unit (CRCCU)
- I/O
  - Up to 48 I/O lines with external interrupt capability (edge or level), debouncing, glitch filtering and on-die series resistor termination. Individually programmable open-drain, pull-up and pull-down resistor and synchronous output
  - Two PIO Controllers provide control of up to 48 I/O lines

- Analog
  - One 8-channel ADC, resolution up to 12 bits, sampling rate up to 500 ksps
- Package
  - 49-lead WLCSP
  - 64-lead LQFP
  - 64-lead QFN
- Operating Temperature Range
  - Industrial (-40°C to +85°C)

1. Configuration Summary

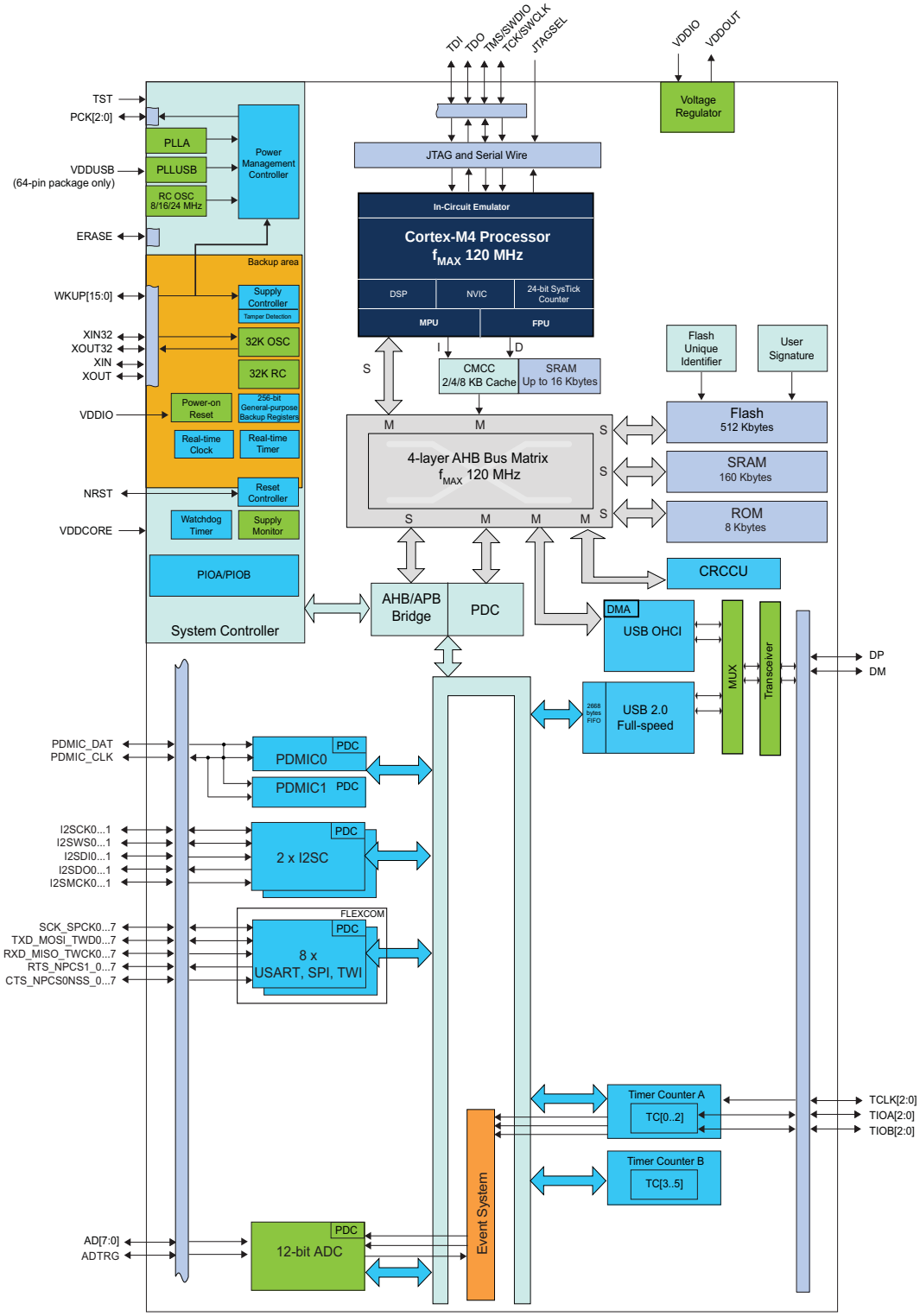
Table 1-1 summarizes the SAM G55 device configurations.

Table 1-1. Configuration Summary

Feature	SAM G55G19	SAM G55J19
Flash	512 Kbytes	512 Kbytes
Cache (CMCC)	up to 8 Kbytes	up to 8 Kbytes
SRAM	160 Kbytes + up to 16 Kbytes (Cache + I/D RAM)	160 Kbytes + up to 16 Kbytes (Cache + I/D RAM)
Package	WLCSP49	QFN64, LQFP64
Number of PIOs	38	48
Event System	Yes	Yes
External Interrupt	16	16
12-bit ADC	8 channels Performance: 500 kSps	8 channels Performance: 500 kSps
16-bit Timer	6 channels (3 external channels)	6 channels (3 external channels)
I2SC/PDM	2 / 1-channel 2-way	2 / 1-channel 2-way
PDC Channels	28	30
USART	7	8
SPI		
TWI		
USB	Full Speed / OHCI	Full Speed / OHCI
CRCCU	1	1
RTT	1 (backup area)	1 (backup area)
RTC	1 (backup area)	1 (backup area)

2. Block Diagram

Figure 2-1. SAM G55 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	Peripheral I/O Lines, Voltage Regulator, ADC Power Supply	Power	–	–	1.62V to 3.6V
VDDOUT	Voltage Regulator Output	Power	–	–	1.08V to 1.32V
VDDCORE	Core Chip Power Supply	Power	–	–	Connected externally to VDDOUT or $V_{DDCOREXT100}$ or $V_{DDCOREXT120}$
VDDUSB	USB Power Supply	Power	–	–	Only available on 64-pin package
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

Table 3-1. Signal Description List (Continued)

Signal Name	Function	Type	Active Level	Voltage Reference	Comments
PIO Controller - PIOA - PIOB					
PA0–PA31	Parallel I/O Controller A	I/O	–	VDDIO	Pulled-up input at reset. No pull-down for PA3/PA4/PA14
PB0–PB15[(1)]	Parallel I/O Controller B	I/O	–	VDDIO	Pulled-up input at reset
Wakeup Pins					
WKUP0–15	Wakeup Pin / External Interrupt	I/O	–	VDDIO	Wakeup pins are used also as External Interrupt
Serial Peripheral Interface - SPIx					
MISOx	Master In Slave Out	I/O	–	–	–
MOSIx	Master Out Slave In	I/O	–	–	–
SPCKx	SPI Serial Clock	I/O	–	–	High Speed Pad
NPCS0x	SPI Peripheral Chip Select 0	I/O	Low	–	–
NPCS1x	SPI Peripheral Chip Select	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
Universal Synchronous Asynchronous Receiver Transmitter USARTx					
SCKx	USART Serial Clock	I/O	–	–	–
TXDx	USART Transmit Data	I/O	–	–	–
RXDx	USART Receive Data	Input	–	–	–
RTSx	USART Request To Send	Output	–	–	–
CTSx	USART Clear To Send	Input	–	–	–
Timer/Counter - TCx					
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	–	–	–
12-bit Analog-to-Digital Converter - ADC					
AD0–AD7	Analog Inputs	Analog	–	–	–
ADTRG	ADC Trigger	Input	–	–	–
ADVREF	ADC Voltage Reference	Input	–	–	Only available on 64-pin package
Inter-IC Sound Controller - I2SCx					
I2SMCKx	Master Clock	Output	–	–	–
I2SCKx	Serial Clock	I/O	–	–	–
I2SWSx	I^2S Word Select	I/O	–	–	–
I2SDIx	Serial Data Input	Input	–	–	–
I2SDOx	Serial Data Output	Output	–	–	–

Table 3-1. Signal Description List (Continued)

Signal Name	Function	Type	Active Level	Voltage Reference	Comments
Pulse Density Modulation Interface Controller - PDMICx					
PDMIC_CLK	Pulse Density Modulation Clock	Output	–	–	–
PDMIC_DAT	Pulse Density Modulation Data	Input	–	–	–
USB OHCI/FS - USB					
DM	USB Data -	Analog, Digital	–	WLCSP49: VDDIO 64-pin package: VDDUSB	DM and DP in PIO configuration
DP	USB Data +				

Note: 1. Pull-up disabled on PB8/PB9.

4. Package and Pinout

Table 4-1. SAM G55 Packages

Device	Package
SAM G55G19	WLCSP49
SAM G55J19	QFN64
	LQFP64

4.1 49-ball WLCSP Pinout

Table 4-2. SAM G55G19 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 64-lead QFN/LQFP Pinout

4.2.1 64-lead QFN / LQFP Pinout

Table 4-3. SAM G55J19 64-pin LQFP and QFN Pinout

1	VDDIO	17	PA6	33	PA17	49	PA9
2	NRST	18	PA16	34	PA18	50	PB5
3	PB12	19	PA30	35	PA19	51	PA27
4	PA4	20	PA29	36	PA20	52	PA26
5	PA3	21	PA28	37	PB0	53	GND
6	PA0	22	PA15	38	PB1	54	PB6
7	PA1	23	PA23	39	PB2	55	PB7
8	PA5	24	PA22	40	PB3	56	PA25
9	VDDCORE	25	PA21	41	PA14	57	PB13
10	TEST	26	VDDUSB	42	PA13	58	PA24
11	PA7	27	VDDIO	43	PA12	59	PB8/XOUT
12	PA8	28	ADVREF	44	PA11	60	PB9/XIN
13	GND	29	GND	45	VDDCORE	61	PA2
14	PB15	30	VDDOUT	46	PB10	62	PB4
15	PB14	31	VDDIO	47	PB11	63	JTAGSEL
16	PA31	32	VDDIO	48	PA10	64	VDDIO

Note: 1. The bottom pad of the QFN package must be tied to ground.

5. Power Considerations

5.1 Power Supplies

The SAM G55 has the following power supply pins:

- VDDCORE pins: Power the core, including the processor, the embedded memories and the peripherals, except the RTC, RTT, and Supply controller (SUPC). It is recommended to connect VDDCORE to VDDOUT.
- VDDIO pins: Power the peripheral I/O lines, RTC, RTT, and SUPC peripherals, voltage regulator and ADC; voltage ranges from 1.62V to 3.6V.
- VDDUSB pins: Power the USB (only for devices with 64-pin package), voltage ranges from 3.0V to 3.6V.

The ground pins GND are common to VDDCORE and VDDIO.

5.2 Powerup Considerations

In order to prevent any overcurrent at powerup, it is recommended to connect pin ADVREF to VDDIO, or to get ADVREF to rise as much as possible at the same time as VDDIO.

Note: Pin ADVREF is only available on 64-pin packages QFN and LQFP.

5.2.1 VDDIO Versus VDDCORE

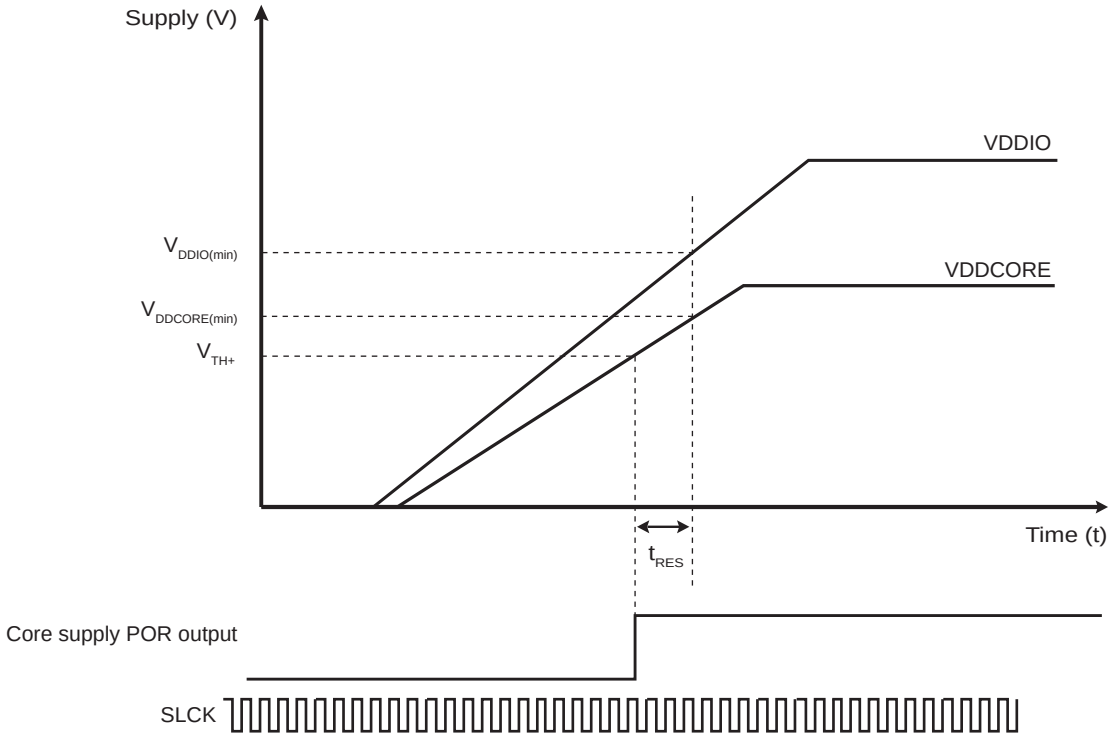
V_{DDIO} must always be higher than or equal to V_{DDCORE}.

V_{DDIO} must reach its minimum operating voltage (1.62 V) before V_{DDCORE} has reached $V_{DDCOREXT(min)}$. The minimum slope for V_{DDCORE} is defined by ($V_{DDCOREXT(min)}$ - V_{TH+}) / t_{RES}.

If $V_{DDCOREXT}$ rises at the same time as V_{DDIO}, the V_{DDIO} rising slope must be higher than or equal to 7V/ms.

If VDDCORE is powered by the internal regulator, all powerup considerations are met.

Figure 5-1. VDDCORE and VDDIO Constraints at Startup



At powerdown, there is no constraint on VDDCORE and VDDIO as the regulator must be enabled.

5.3 Voltage Regulator

The SAM G55 embeds 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. An internal adaptive biasing adjusts the regulator quiescent current depending on the required load current.

For adequate input and output power supply decoupling/bypassing, refer to Table 39-4 “VDDCORE Voltage Regulator Characteristics” in section “Electrical Characteristics”.

In case of dual supply, the voltage regulator must be enabled and VDDOUT must be used as input control of the external DC/DC. This will allow a correct slope at first startup and for low power mode.

5.4 Typical Powering Schematics

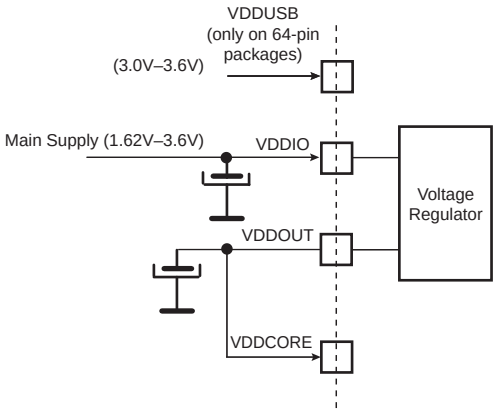
The SAM G55 supports single and dual voltage supply, with VDDIO from 1.62V to 3.6V and VDDCORE from external DC/DC controlled by the internal regulator. Figure 5-2 and Figure 5-3 illustrate the power schematics.

To achieve system performances, the internal voltage regulator must be used.

5.4.1 Single Supply

The SAM G55 supports a 1.62V to 3.6V single supply mode. The internal voltage regulator input is connected to the source and its output feeds VDDCORE. Figure 5-2 illustrates the power schematics.

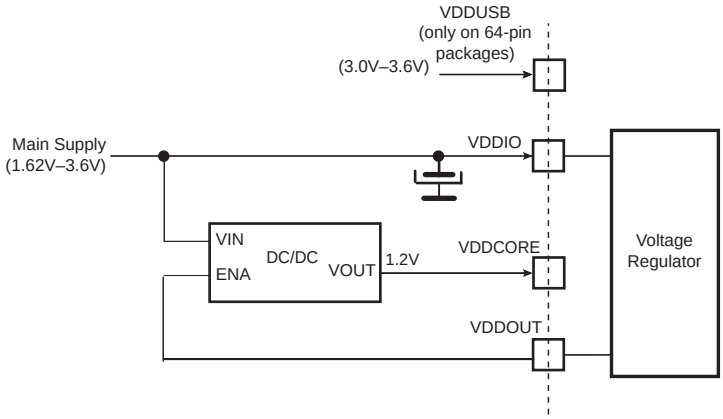
Figure 5-2. Single Supply



5.4.2 Dual Supply

In dual voltage supply, the voltage regulator must always be enabled and must be used as input control of the external DC/DC, to control the slope of VDDCORE after low power mode.

Figure 5-3. Dual Supply



In Wait mode, by default, the voltage regulator is down to V_{DDOUT} Wait mode minimum value. Consequently, it must be configured to keep VDDOUT in Running mode. To avoid any issue, the regulator must be configured by software to deliver the correct supply voltage. To do this, use the following procedure:

- Read the unique identifier bytes [65..64]
- Write the four LSB bits of unique identifier bytes [65..64] in SUPC_PWMR.LPOWER0–LPOWER3
- Enable SUPC_PWMR.LPOWERS

5.5 Functional Modes

5.5.1 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.

5.5.2 Backup Mode

The purpose of Backup mode is to achieve the lowest power consumption possible in a system which is performing periodic wakeups to perform tasks but not requiring fast startup time.

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

The SAMG55 can be awakened from this mode using the pins WKUP0–15, the supply monitor (SM), the RTT, or the RTC.

Backup mode is entered by writing a 1 to the VROFF bit of the Supply Controller Control Register (SUPC_CR) (a key is needed to write the VROFF bit, refer to section Supply Controller SUPC) and with the SLEEPDEEP bit in the Cortex-M4 System Control Register set to 1 (see power management description in section ARM Cortex-M4 Processor). To reduce consumption, the supply monitor on VDDIO can be disabled.

To enter Backup mode using the VROFF bit:

- Write a 1 to the VROFF bit of SUPC_CR.

To enter Backup mode using the WFE instruction:

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

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

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

5.5.3 Wait Mode

Wait mode allows the device to achieve very low power consumption levels while remaining in a powered state with a wakeup time of less than a few µs. In Wait mode, the clocks of the core, the peripherals and memories are stopped. However, power supplies are maintained to ensure memory and CPU context retention.

The wakeup time is achieved when entry into and exit from Wait mode are performed in internal SRAM. The wakeup time increases to 6.9 µs if entry into Wakeup mode is performed in internal Flash.

Wait mode is entered using either the WAITMODE bit in the PMC Clock Generator Main Oscillator register (CKGR_MOR) or the Wait for Event (WFE) instruction. Before entering Wait mode, the POR core must be disabled. Detailed sequences are provided below.

Note that the WFE instruction can add complexity in application state machines due to the fact that the WFE instruction goes along with an event flag of the Cortex core (cannot be managed by the software application). The event flag can be set by interrupts, a debug event or an event signal from another processor. Since an interrupt can 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, follow the sequence using the WAITMODE bit described below.

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 startup wakeup pins (refer to Section 5.6 “Fast Startup”) or the RTT and RTC alarms, USB interrupt line or SleepWalking for FLEXCOM0–7 (USART/SPI/TWI) for internal events.

To enter Wait mode using the WAITMODE bit:

1. Select the 8/16/24 MHz fast RC oscillator as the Main Clock. If frequency of 24 MHz is selected and the code is running from the SRAM.
2. Program the FLPM field in the PMC Fast Startup Mode Register (PMC_FSMR)(1).
3. Set the number of Flash wait states to 1 by writing a one to the FWS field in the EEFC Flash Mode Register (EEFC_MR).
4. Write a one to the WAITMODE bit in the CKGR_MOR.
5. Wait for MCKRDY = 1 in the PMC Status Register (PMC_SR).

To enter Wait mode using the WFE instruction:

1. Select the 8/16/24 MHz fast RC oscillator as the Main Clock. If 24 MHz is selected and the code is running on the SRAM.
2. Program the FLPM field in the PMC Fast Startup Mode Register (PMC_FSMR)(1).
3. Set the number of Flash wait states to 1 by writing a one to the FWS field in the EEFC Flash Mode Register (EEFC_MR).
4. Write a one to the LPM bit in PMC_FSMR.
5. Execute the Wait For Event (WFE) instruction of the processor.

Note: 1. Depending on the value of the field FLPM, the Flash enters on of three different modes:
FLPM = 0: Flash in Standby mode (low power consumption levels)
FLPM = 1: Flash in Deep-powerdown mode (extra low power consumption levels)
FLPM = 2: Flash in Idle mode. Memory ready for Read access.

5.5.4 Sleep Mode

In Sleep mode, power consumption of the device versus response time is optimized. 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) instructions.

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

5.5.5 Low-power Mode Configuration Summary

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

Table 5-1. Low-power Mode Configuration Summary

Component or Parameter	Low-power Mode		
	Backup Mode	Wait Mode with Flash in Deep-powerdown mode	Sleep Mode
SUPC, 32 kHz Oscillator, RTT, POR, Voltage Regulator	ON	ON	ON
POR, Supply Monitor on VDDIO	OFF(1)	OFF	ON
RAM Power Switch	Not powered	From all RAM powered to 8 Kbytes RAM powered	Powered
Core, Memory, Peripherals	Not powered	Powered (Not clocked)	Powered (Not clocked)
Mode Entry	SUPC_CR.VROFF = 1 + SCB_SCR.SLEEPDEEP = 1	PMC_FSMR.FLPM = 1 + CKGR_MOR.WAITMODE = 1 or SCB_SCR.SLEEPDEEP = 0 + PMC_FSMR.FLPM = 1 + PMC_FSMR.LPM = 1 + WFE	WFI + SCB_SCR.SLEEPDEEP = 0 + PMC_FSMR.LPM = 0
Potential Wakeup Sources	Pins WKUP0–15 RTC alarm RTT alarm	Any event from: - Fast startup through pins WKUP0–15 - RTT alarm - RTC alarm - USB device interrupt line - FLEXCOM0–7 and ADC SleepWalking	Entry mode = WFI interrupt only; any enabled interrupt
Core at Wakeup	Reset	Clocked back	Clocked back
PIO State while in Low-power Mode	Previous state saved	Previous state saved	Previous state saved
PIO State at Wakeup	PIOA & PIOB Inputs with pull-ups	Unchanged	Unchanged
Consumption(2) (3)	Refer to Table 39-9	Refer to Table 39-10	(4)
Wakeup Time(5)	–	Refer to Table 39-11	–

Notes:
1. If the supply monitor is enabled, the Wakeup can be done through the SM.
2. The external loads on PIOs are not taken into account in the calculation.
3. BOD current consumption is not included.
4. Refer to Section 39.4 “Power Consumption” in the electrical characteristics.
5. When considering wakeup 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 startup time if it is needed in the system. The wakeup time is defined as the time taken for wake up until the first instruction is fetched.

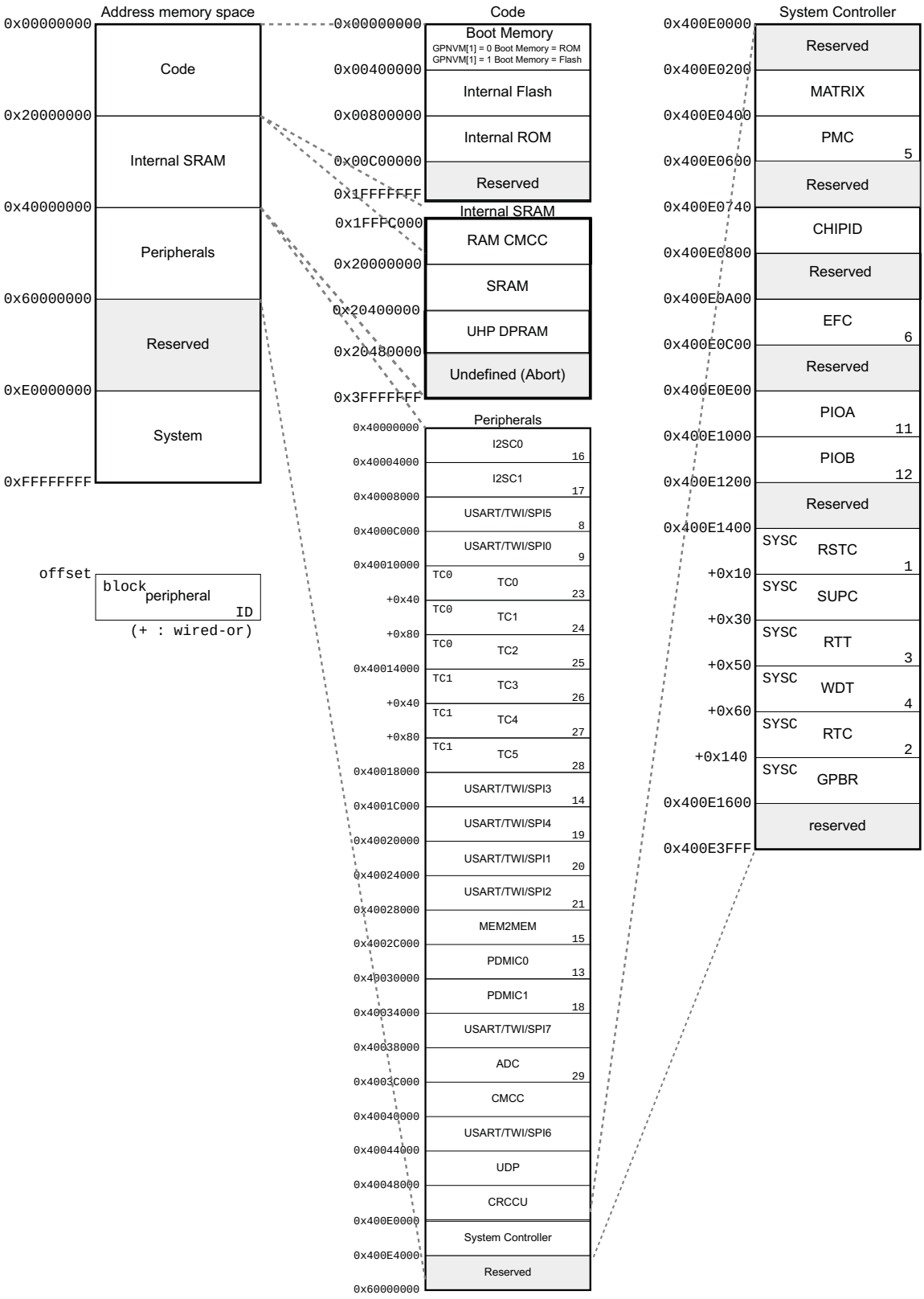
5.6 Fast Startup

The SAM G55 allows the processor to restart in a few microseconds while the processor is in Wait mode. A fast startup can occur upon detection of a low level on one of the 18 wakeup inputs.

The fast restart circuitry is fully asynchronous and provides a fast startup signal to the Power Management Controller. As soon as the fast startup signal is asserted, the PMC 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 the wakeup of Wait mode, the code is executed in the SRAM.

6. Product Mapping

Figure 6-1. SAM G55 Product Mapping



7. Bootloader

The SAM G55 devices ship with a bootloader in ROM, used to download code, in internal Flash, either through the SPI or through the TWI3.

The Bootloader mode is entered automatically on powerup if no valid firmware is detected in the Flash. A valid firmware is detected by performing a CRC on the content of the Flash. If the CRC is correct, the application is started. Otherwise, the Bootloader mode is entered.

Alternatively, the Bootloader mode can be forced by applying low pulses on the NRST line. The NRST should be asserted 10 times for a minimum of 1 µs at an interval less than 50 ms. When the bootloader detects this sequence, it asserts the pin PA01 (NCHG) low as an acknowledge.

The Bootloader mode initializes the TWI3 in Slave Mode with the I2C address 0x26 and the SPI in Slave Mode, 8-bit data length, SPI Mode 1.

Table 7-1 provides information on the pins used by the bootloader.

Table 7-1. Boot Loader Pin Description

Pin Name	Function	Bootloader Use	Description
PA01	NCHG	Driven at 0 or pulled up	Boot loader handshake
PA03	TWD	Open drain input/output	TWI/I2C data line
PA04	TWCK	Open drain input/output	TWI/I2C clock
PA11	NPCS0/NSS	Input	NSS, SPI slave select
PA12	MISO	Push-pull output	SPI master in slave out
PA13	MOSI	Input	SPI master out slave in
PA14	SPCK	Input	SPI clock

For further details on bootloader operations, refer to the application note *AT09002: Atmel SAM I^2C - SPI Bootloader* on www.atmel.com.

8. Memories

8.1 Internal SRAM

The SRAM G55 embeds a total of 176 Kbytes of high-speed SRAM, accessible at address 0x1FFF_C000.

The 160 Kbytes of SRAM are accessible over the Cortex-M4 system 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. The SRAM is composed of five blocks of 32 Kbytes. The five blocks have a power switch. Each power switch controls the supply of the SRAM block to save power. The power switch control (SRAMxON) is in the SUPC_PWMR register (refer table 8-1).

The SRAM G55 also embeds up to 16 Kbytes of SRAM accessible at address 0x1FC0_0000. The 16 Kbytes of SRAM can be assigned by the customer to Data Cache RAM and/or Tightly Coupled Memory (TCM) RAM (CMCC) and on I/D bus following this configuration (PRGCSIZE) in the CMCC_CFG register:

- 2 Kbytes of Data RAM Cache and 14 Kbytes TCM RAM on I/D Bus (CMCC)
- 4 Kbytes of Data RAM Cache and 12 Kbytes TCM RAM on I/D Bus (CMCC)
- 8 Kbytes of Data RAM Cache and 8 Kbytes TCM RAM on I/D Bus (CMCC)

The 16 Kbytes of SRAM (Data Cache/TCM SRAM) also has a power switch (CDPSWITCH) on SUPC_MR which controls the supply of the block.

Table 8-1. SRAM Power Switch vs SRAM Block

Power Switch	SRAM Block	SRAM Size	Address	SUPC_PWMR
0	Block 0	8 Kbytes	0x2000_0000	SRAM0ON
1	Block 0	8 Kbytes	0x2000_2000	SRAM1ON
2	Block 0	16 Kbytes	0X2000_4000	SRAM2ON
3	Block 1	32 Kbytes	0x2000_8000	SRAM3ON
4	Block 2	32 Kbytes	0x2001_0000	SRAM4ON
5	Block 3	32 Kbytes	0x2001_8000	SRAM5ON
6	Block 4	32 Kbytes	0x2002_0000	SRAM6ON
7	USB DPRAM	–	–	DPRAMON

8.2 Internal ROM

The SAMG55 product embeds an Internal ROM.

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

8.3 Embedded Flash

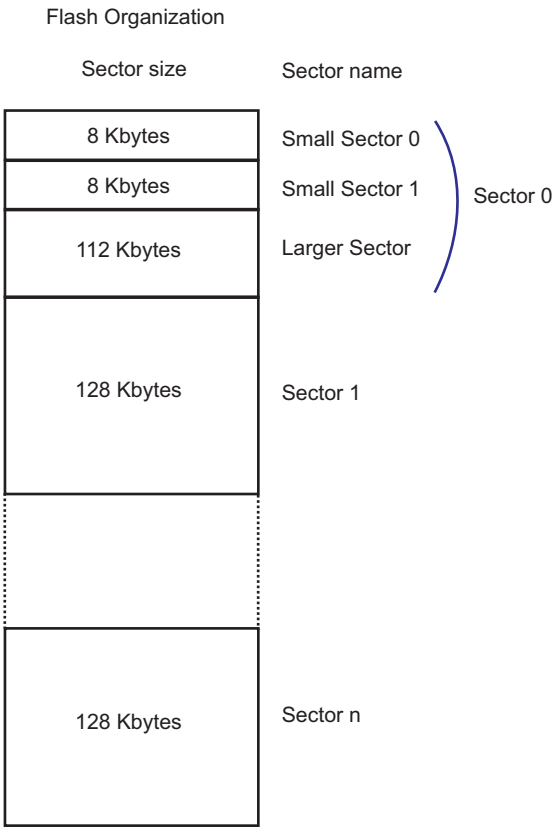
8.3.1 Flash Overview

The memory is organized in sectors. Each sector comprises 128 Kbytes. The first sector of 128 Kbytes is divided into three smaller sectors.

The three smaller sectors are comprised of two sectors of 8 Kbytes and one sector of 112 Kbytes.

Refer to Figure 8-1.

Figure 8-1. Global Flash Organization



Each sector is organized in pages of 512 bytes.

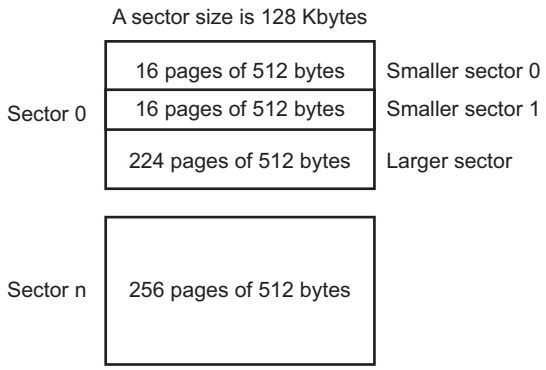
For sector 0:

- The smaller sector 0 has 16 pages of 512 bytes.
- The smaller sector 1 has 16 pages of 512 bytes.
- The larger sector has 224 pages of 512 bytes.

From sector 1 to n:

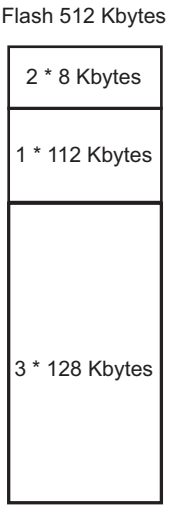
- The rest of the array is composed of 128-Kbyte sectors of 256 pages of 512 bytes each. Refer to Figure 8-2.

Figure 8-2. Flash Sector Organization



The SAM G55 Flash size is 512 Kbytes. Refer to Figure 8-3 for the organization of the Flash.

Figure 8-3. Flash Size



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

- 8 Kbyte small sector
  - Erase and write page (EWP)
  - Erase and write page and lock (EWPL)
  - Erase sector (ES) with FARG set to a page number in the sector to erase
  - Erase pages (EPA) with FARG [1:0] = 0 to erase 4 pages, FARG [1:0] = 1 to erase 8 pages or FARG [1:0] = 2 to erase 16 pages. FARG [1:0] = 3 must not be used.
- 112 Kbyte and 128 Kbyte sectors
  - 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.

8.3.1.1 Enhanced Embedded Flash Controller

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

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

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

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

8.3.1.2 Flash Speed

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

For more details, refer to Section 39.9 “AC Characteristics”.

8.3.1.3 Lock Regions

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

Table 8-2. Lock Bit Number

Product	Number of Lock Bits	Lock Region Size
SAM G55	64	8 Kbytes

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

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

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

8.3.1.4 User Signature

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

Read, write and erase of this area is allowed.

8.3.1.5 Unique Identifier

The G55 Flash contains two pages of 512 bytes called unique identifier. These two pages are read-only and cannot be erased even by the Erase pin. Each device integrates its own 128-bit unique identifier. These bits are factory-configured and cannot be changed by the user.

The sequence to read the unique identifier area is described in Section 23.4.3.8 “Unique Identifier Area”.

Some bytes within the unique identifier pages are reserved for the trimming information of the 32 kHz RC oscillator and the internal voltage regulator.

The mapping is as follows:

- Bytes [15..0]: 128 bits for unique identifier
- Bytes [47..16]: Atmel reserved
- Bytes [49..48]: Measured frequency (on tester) of the internal 32 kHz RC when V_{DDIO}= 3.3V (measurement performed at 25°C). These two bytes contain the frequency in hertz.
- Bytes [51..50]: Measured frequency (on tester) of the internal 32 kHz RC when V_{DDIO}= 1.8V (measurement performed at 25°C). These two bytes contain the frequency in hertz.
- Bytes [63..52]: Atmel reserved
- Bytes [65..64]: Trimmed code of the internal regulator which allows the device to run at up to 120 MHz. The four LSB bits must be written in the SUPC_PWMR.ECPWRx.
- Bytes [67..66]: Trimmed code of the internal regulator which allows the device to run at up to 100 MHz. Only the four LSB bits are used. They must be written in the SUPC_PWMR.ECPWRx. It is the default value after reset.
- Bytes [67..511]: Atmel reserved

8.3.1.6 General-Purpose Non-Volatile Memory Bits

The SAM G55 features three GPNVM bits that can be cleared or set, respectively, through the commands “Clear GPNVM Bit” and “Set GPNVM Bit” of the EEFC User Interface.

Table 8-3. General-purpose Non-volatile Memory Bits

GPNVM Bit	Function
0	Security bit
1	Boot Mode Selection
2	Reserved (do not use)

8.3.1.7 Boot Strategies

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

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

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

Setting GPNVM1 selects the boot from the Flash. Clearing it selects the boot from the ROM. Asserting ERASE clears the GPNVM1 and thus selects the boot from the ROM by default.

8.3.1.8 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.

See Section 23.4.3.6 “Calibration Bit” for more information.

8.3.1.9 Security Bit

The SAM G55 features a security bit, based on a specific general-purpose NVM bit (GPNVM bit 0). When the security is enabled, any access to the Flash, SRAM, core registers and internal peripherals through the ICE interface is forbidden. This ensures the confidentiality of the code programmed in the Flash.

The security bit can only be enabled with the command “Set GPNVM Bit 0” of the EEFC User Interface. Disabling the security bit can only be done by asserting the ERASE pin to 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.

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

To avoid unexpected erase at powerup, a minimum ERASE pin assertion time is required. This time is defined in Table 39-50 “AC Flash Characteristics”.

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

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

The following sequence details the steps of the erase operation:

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

9. Peripherals

9.1 Peripheral Identifiers

Table 9-1 defines the peripheral identifiers of the SAM G55. A peripheral identifier is required:

- for the control of the peripheral interrupts by the Nested Vectored Interrupt Controller
- to enable/disable the peripheral clocks by means of the Peripheral Clock Enable and Disable registers (PMC_PCERx, PMC_PCDRx) in the Power Management Controller.

The external interrupts are connected to WKUP pins (level detection managed by the SUPC) and the default detection is in low level.

Table 9-1. Peripheral Identifiers

Instance ID	Instance Name	NVIC Interrupt	PMC Clock Control	Instance Description
0	SUPC	X	–	Supply Controller
1	RSTC	X	–	Reset Controller
2	RTC	X	–	Real-time Clock
3	RTT	X	–	Real-time Timer
4	WDT	X	–	Watchdog Timer
5	PMC	X	–	Power Management Controller
6	EFC	X	–	Enhanced Flash Controller
7	USART7, SPI7, TWI7	X	X	USART/SPI/TWI 7
8	USART0, SPI0, TWI0	X	X	USART/SPI/TWI 0
9	USART1, SPI1, TWI1	X	X	USART/SPI/TWI 1
10	Reserved	–	–	–
11	PIOA	X	X	Parallel I/O Controller A
12	PIOB	X	X	Parallel I/O Controller B
13	PDMIC0	X	X	Pulse Density Modulation Interface Controller 0
14	USART2, SPI2, TWI2	X	X	USART/SPI/TWI 2
15	MEM2MEM	X	X	Memory to Memory
16	I2SC0	X	X	Inter-IC Sound Controller 0
17	I2SC1	X	X	Inter-IC Sound Controller 1
18	PDMIC1	X	X	Pulse Density Modulation Interface Controller 1
19	USART3, SPI3, TWI3	X	X	USART/SPI/TWI 3
20	USART4, SPI4, TWI4	X	X	USART/SPI/TWI 4
21	USART5, SPI5, TWI5	X	X	USART/SPI/TWI 5
22	USART6, SPI6, TWI6	X	X	USART/SPI/TWI 6
23	TC0	X	X	Timer/Counter 0
24	TC1	X	X	Timer/Counter 1
25	TC2	X	X	Timer/Counter 2
26	TC3	X	X	Timer/Counter 3
27	TC4	X	X	Timer/Counter 4

Table 9-1. Peripheral Identifiers (Continued)

Instance ID	Instance Name	NVIC Interrupt	PMC Clock Control	Instance Description
28	TC5	X	X	Timer/Counter 5
29	ADC	X	X	Analog-to-Digital Converter
30	ARM	X	–	FPU
31	WKUP0	X	–	External interrupt 0
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
47	UHP	X	X	USB OHCI
48	UDP	X	X	USB Device FS
49	CRCCU	X	X	Cyclic Redundancy Check Calculation Unit

9.2 Peripheral Signal Multiplexing on I/O Lines

The SAM G55 features two PIO controllers, PIOA and PIOB, which multiplex the I/O lines of the peripheral set.

Each line can be assigned to one of two peripheral functions: A or B. The multiplexing tables in this section 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.

Table 9-2. Available PIO Lines per Package

PIO Controller	PIO Lines per Package	
	49-Lead Package	64-Lead Package
PIOA	25	32
PIOB	13	16

9.2.1 PIO Controller A Multiplexing

Table 9-3. Multiplexing on PIO Controller A (PIOA)

I/O Line	Peripheral A	Peripheral B	Extra Function	System Function
PA0	I2SCK0	TIOA0	WKUP0(1)	–
PA1	I2SWS0	TIOB0	WKUP1(1)	–
PA2	TCLK0	I2SDI0	WKUP2(1)	–
PA3	TXD3/SPI3_MOSI/TWD3	I2SDO0	WKUP9(1)	–
PA4	RXD3/SPI3_MISO/TWCK3	I2SMCK0	WKUP10(1)	–
PA5	RXD2/SPI2_MISO/TWCK2	SPI5_NPCS1/RTS5	WKUP4(1)	–
PA6	TXD2/SPI2_MOSI/TWD2	PCK0	–	–
PA7	–	–	–	XIN32(2)
PA8	–	ADTRG	WKUP5(1)	XOUT32(2)
PA9	RXD0/SPI0_MISO/TWCK0	PDMIC_DAT	WKUP6(1)	–
PA10	TXD0/SPI0_MOSI/TWD0	PDMIC_CLK	–	–
PA11	SPI5_NPCS0/CTS5	–	–	–
PA12	RXD5/SPI5_MISO/TWCK5	–	–	–
PA13	TXD5/SPI5_MOSI/TWD5	–	–	–
PA14	SCK5/SPI5_SPCK	–	WKUP8(1)	–
PA15	SPI2_NPCS1/RTS2	SCK2/SPI2_SPCK	–	–
PA16	SPI2_NPCS0/CTS2	TIOB1	WKUP7(1)	–
PA17	I2SDO0	PCK1	AD0(3)	–
PA18	I2SMCK0	PCK2	AD1(3)	–
PA19	TCLK1	I2SCK1	AD2(3)	–
PA20	TCLK2	I2SWS1	AD3(3)	–
PA21	TIOA2	PCK1	–	DM
PA22	TIOB2	I2SDI1	–	DP
PA23	I2SDO1	TIOA1	WKUP3(1)	–
PA24	I2SMCK1	SCK2/SPI2_SPCK	WKUP11(1)	–
PA25	SPI0_NPCS0/CTS0	I2SDO1	–	–
PA26	SPI0_NPCS1/RTS0	I2SMCK1	–	–
PA27	SCK1/SPI1_SPCK	RXD7/SPI7_MISO/TWCK7	–	–
PA28	SPI1_NPCS0/CTS1	TXD7/SPI7_MOSI/TWD7	–	–
PA29	SPI1_NPCS1/RTS1	SCK7/SPI7_SPCK	–	–
PA30	PCK1	SPI7_NPCS0/CTS7	–	–
PA31	PCK2	SPI7_NPCS1/RTS7	–	–

Notes: 1. WKUPx can be used if PIO controller defines the I/O line as "input".
2. Refer to Section 9.3 “System I/O Lines”.
3. To select this extra function, refer to Section 39.5.3 “I/O Lines”.

9.2.2 PIO Controller B Multiplexing

Table 9-4. Multiplexing on PIO Controller B (PIOB)

I/O Line	Peripheral A	Peripheral B	Extra Function	System Function
PB0	SCK0/SPI0_SPCK	TXD6/SPI6_MOSI/TWD6	AD4[2]	–
PB1	SCK4/SPI4_SPCK	RXD6/SPI6_MISO/TWCK6	AD5[2]	–
PB2	RXD1/SPI1_MISO/TWCK1	SPI5_NPCS1/RTS5	AD6/WKUP12[3]	–
PB3	TXD1/SPI1_MOSI/TWD1	PCK2	AD7//WKUP13[3]	–
PB4	–	–	–	TDI[5]
PB5	–	–	–	TDO/ TRACESWO[5]
PB6	–	–	–	TMS/SWDIO[5]
PB7	–	–	–	TCK/SWCLK[5]
PB8	TXD4/SPI4_MOSI/TWD4	SPI4_NPCS0/CTS4	WKUP14[4]	XOUT[5]
PB9	RXD4/SPI4_MISO/TWCK4	SPI4_NPCS1/RTS4	WKUP15[4]	XIN[5]
PB10	TXD4/SPI4_MOSI/TWD4[1]	TXD6/SPI6_MOSI/TWD6[1]	–	–
PB11	RXD4/SPI4_MISO/TWCK4[1]	RXD6/SPI6_MISO/TWCK6[1]	–	–
PB12	–	–	–	ERASE[5]
PB13	SCK3/SPI3_SPCK	SCK6/SPI6_SPCK	–	–
PB14	SPI3_NPCS0/CTS3	SPI6_NPCS0/CTS6	–	–
PB15	SPI3_NPCS1/RTS3	SPI6_NPCS1/RTS6	–	–

Note:
1. Each TWI (TWI4,TWI6) 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 determines which TWI is selected.
2. To select this extra function, refer to Section 39.5.3 “I/O Lines”.
3. Analog input has priority over WKUPx pin.
4. WKUPx can be used if PIO controller defines the I/O line as “input”.
5. Refer to Section 9.3 “System I/O Lines”.

9.2.2.1 TWI Multiplexing

The TWI function must first be selected through the FLEXCOM interface (OPMODE field in FLEXCOM_MR).

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 TWI4 and Alternative Mode TWI6.

Normal Mode

- Only TWI4 used: PB09 and PB08 must be configured as PIO Peripheral A
- Only TWI6 used: PB00 and PB01 must be configured as PIO Peripheral B
- Both TWI4 and TWI6 used: PB09 and PB08 must be configured as PIO Peripheral A and PB00 and PB01 must be configured as PIO Peripheral B.

Alternative Mode TWI4

TWI4 is multiplexed 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 TWI6

TWI6 is multiplexed 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.

Example of Alternative Mode TWI4:

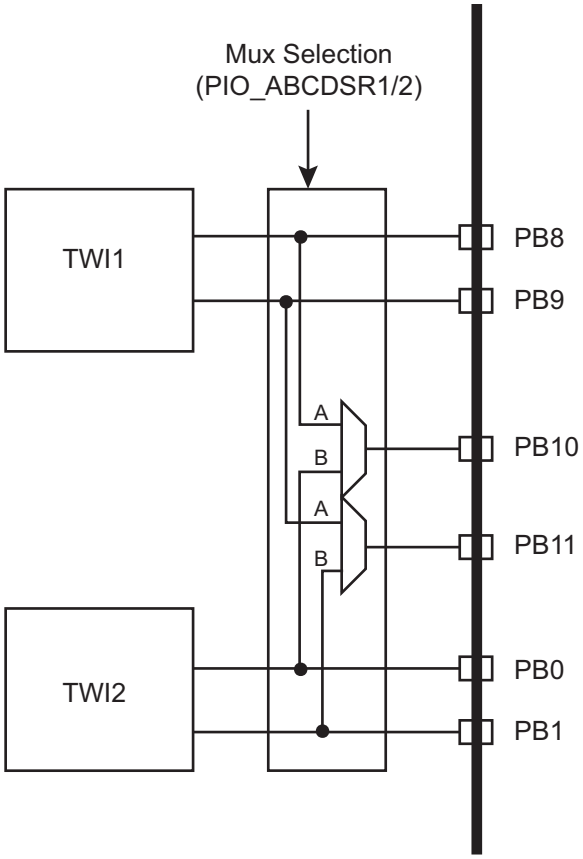
- PB10 is driven by TWD4 signal if the PB10 is configured as peripheral (PIO_PSR[10] = 1 and PIO_ABCDSR1[10] = PIO_ABCDSR2[10] = 0).
- PB11 is driven by TWCK4 signal if the PB11 is configured as peripheral (PIO_PSR[11] = 1 and PIO_ABCDSR1[11] = PIO_ABCDSR2[11] = 0).

Table 9-5. TWI Multiplexing

Required Configuration	PIO Configuration		PB0	PB1	PB8	PB9	PB10	PB11
	PB10	PB11						
Normal Mode TWI1 and/or TWI2 Used	Enable PIO_PER[10]	Enable PIO_PER[11]	TWD6	TWCK6	TWD4	TWCK4	GPIO	GPIO
Alternative Mode TWI1	Config PB10 as Peripheral A	Config PB11 as Peripheral A	TWD6	TWCK6	GPIO or PIO_periphB or WKUP pin(1)	GPIO or PIO_periphB or WKUP pin(1)	TWD4	TWCK4
Alternative Mode TWI2	Config PB10 as Peripheral B	Config PB10 as Peripheral B	GPIO or PIO_periphB or AD Input(1)	GPIO or PIO_periphB or AD Input(1)	TWD4	TWCK4	TWD6	TWCK6

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

Figure 9-1. TWI Master PIO Muxing Selection



9.3 System I/O Lines

Table 9-6 lists the SAMG55 system I/O lines shared with PIO lines. These pins are software configurable as general purpose I/O or system pins. At startup, the default function of these pins is always used.

Table 9-6. System I/O Configuration Pin List

CCFG_SYSIO Bit No.	Default Function after Reset	Other Function	Constraints for Normal Start	Configuration
12	ERASE	PB12	Low Level at startup(1)	In Matrix User Interface Registers (Refer to the System I/O Configuration Register in Section 15. “Bus Matrix (MATRIX)”.)
11	DP	PA22	–	
10	DM	PA21	–	
7	TCK/SWCLK	PB7	–	
6	TMS/SWDIO	PB6	–	
5	TDO/TRACESWO	PB5	–	
4	TDI	PB4	–	
–	PA7	XIN32	–	(2)
–	PA8	XOUT32	–	
–	PB9	XIN	–	(3)
–	PB8	XOUT	–	

Notes:
1. If PB12 is used as PIO input in user applications, a low level must be ensured at startup to prevent Flash erase before the user application sets PB12 into PIO mode.
2. Refer to Section 26.4.2 “Slow Clock Generator”.
3. Refer to Section 17.5.3 “3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator”.

10. 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 used to select the required peripheral.

10.1 Embedded Characteristics

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

10.2 Real-time Event Mapping

Table 10-1. Real-time Event Mapping List

Function	Application	Description	Event Source	Event Destination
Safety	General-purpose	Automatic switch to reliable main RC oscillator in case of main crystal clock failure(1)	Power Management Controller (PMC)	PMC
Security	General-purpose	Immediate GPBR clear (asynchronous) on tamper detection through WKUP0/1 IO pins(2)	PIO: WKUP0/1	GPBR
Measurement trigger	General-purpose	Trigger source selection in ADC(3)	PIO: ADTRG	ADC
			TC: TIOA0	ADC
			TC: TIOA1	ADC
			TC: TIOA2	ADC
			RTC: RTCOUT0(4)	ADC
			RTT: 16-bit prescaler output(5)	ADC
		Last Channel Specific Measurement Trigger(6)	RTC: RTCOUT1(4)	ADC
	SleepWalking	Trigger source selection in ADC(3)	RTT: RTTEVENT(7)	ADC
Direct Memory Access	General-purpose	Peripheral trigger event generation to transfer data to/from system memory(8)	FLEXCOM (USART/TWI/SPI) 0/1/2/3/4/5/6/7, ADC, TC, I2SC0/1, PDMIC0/1	PDC

Notes:
1. Refer to “Main Clock Failure Detection” in section “Power Management Controller (PMC)”
2. Refer to “Low-power Tamper Detection and Anti-Tampering” in section “Supply Controller (SUPC)” and “General Purpose Backup Register x” in section “General Purpose Backup Register (GPBR)”
3. Refer to “ADC Mode Register (ADC_MR)” in section “Analog-to-Digital Converter (ADC)”.
4. Refer to “Waveform Generation” in section “Real-time clock (RTC)”
5. Refer to “Block Diagram” in section “Real-time Timer (RTT)”
6. Refer to “Last Channel Specific Measurement Trigger” in section “Analog-to-Digital Converter (ADC)”
7. Refer to “Block Diagram” and “Real-time Timer Modulo Selection Register (RTT_MODR)” in section “Real-time Timer (RTT)”
8. Refer to “Peripheral DMA Controller (PDC)”.

11. Debug and Test Features

11.1 Description

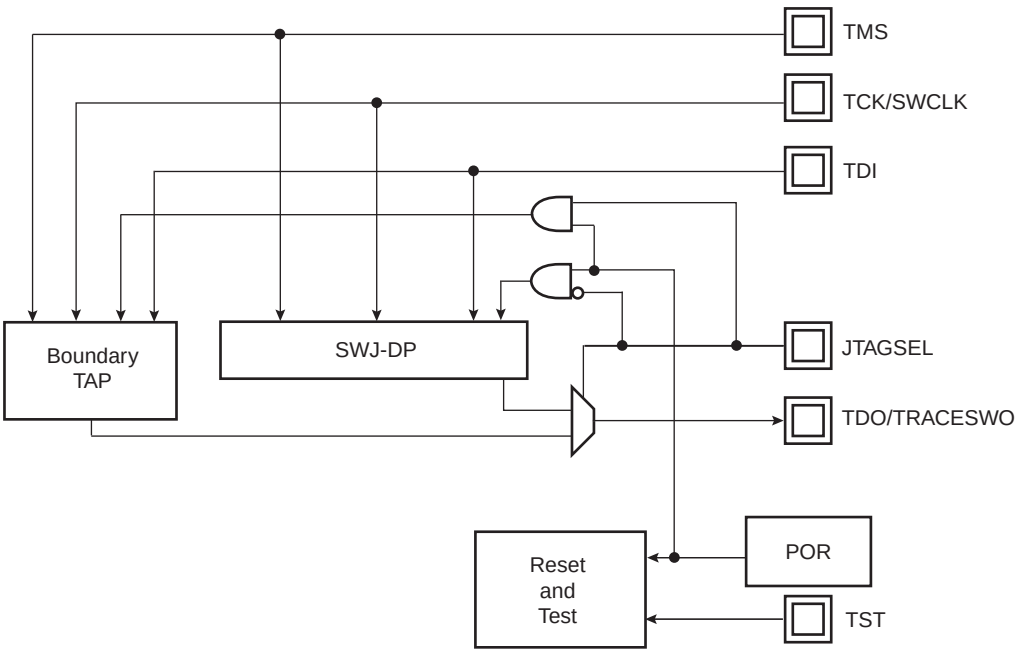
The SAM G55 features a number of complementary debug and test capabilities. The Serial Wire/JTAG Debug Port (SWJ-DP) combining a Serial Wire Debug Port (SW-DP) and JTAG Debug Port (JTAG-DP) is used for standard debugging functions, such as downloading code and single-stepping through programs. It also 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.

11.3 Debug and Test Block Diagram

Figure 11-1. Debug and Test Block Diagram

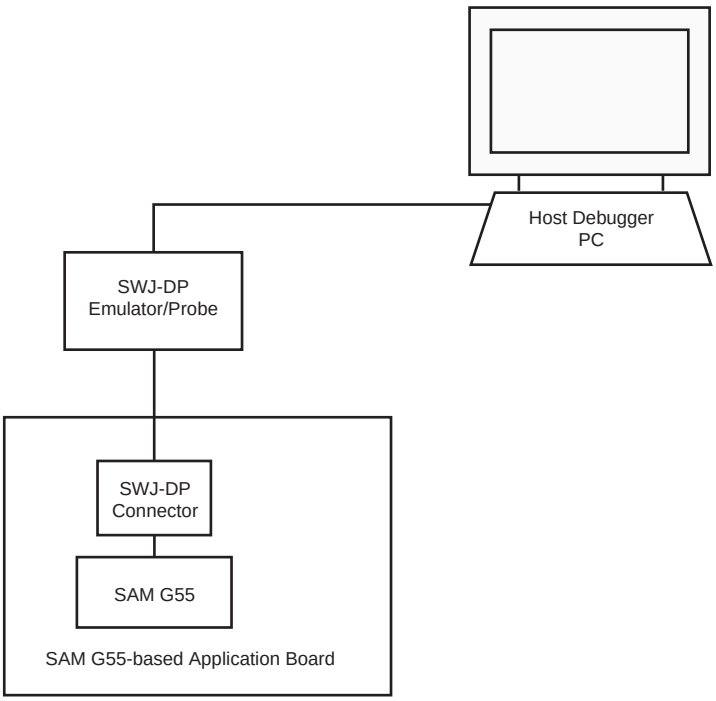


11.4 Application Examples

11.4.1 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.

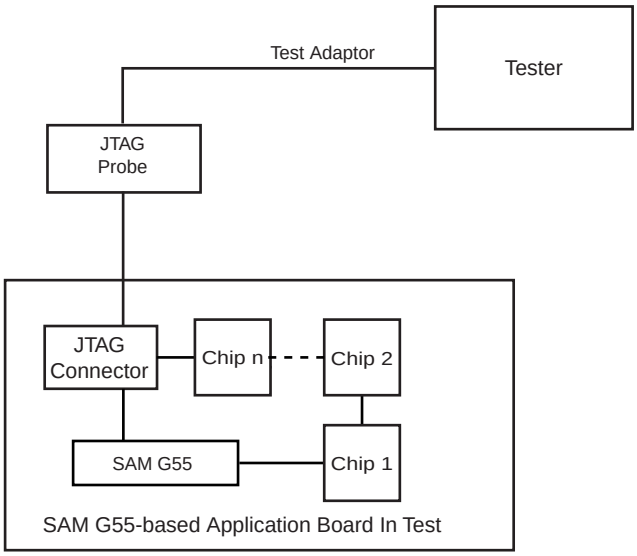
Figure 11-2. Application Debug Environment Example



11.4.2 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



11.5 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.

11.6 Functional Description

11.6.1 Test Pin

One dedicated pin, TST, is used to define the device operating mode. When this pin is at low level during powerup, the device is in normal operating mode. When at high level, the device is in test 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.

11.6.2 NRST Pin

The NRST pin is bidirectional. It is handled by the on-chip reset controller and can be driven low to provide a reset signal to the external components or asserted low externally to reset the microcontroller. It will reset the Core and the peripherals except the Backup region (RTC, RTT and Supply Controller). There is no constraint on the length of the reset pulse and the reset controller can guarantee a minimum pulse length. The NRST pin integrates a permanent pull-up resistor to VDDIO of about 100 kΩ. By default, the NRST pin is configured as an input.

11.6.3 ERASE Pin

The ERASE pin is used to reinitialize the Flash content (and some of its NVM bits) to an erased state (all bits read as logic level 1). The ERASE pin and the ROM code ensure an in-situ reprogrammability of the Flash content without the use of a debug tool. When the security bit is activated, the ERASE pin provides the capability to reprogram the Flash content. It integrates a pull-down resistor of about 100 kΩ to GND, so that it can be left unconnected for normal operations.

This pin is debounced by SCLK to improve the glitch tolerance. To avoid unexpected erase at powerup, a minimum ERASE pin assertion time is required. This time is defined in Table 39-50 “AC Flash Characteristics”.

The ERASE pin is a system I/O pin and can be used as a standard I/O. At startup, the ERASE pin is not configured as a PIO pin. If the ERASE pin is used as a standard I/O, startup level of this pin must be low to prevent unwanted erasing. Also, if the ERASE pin is used as a standard I/O output, asserting the pin to low does not erase the Flash. For details, please refer to Section 9.2 “Peripheral Signal Multiplexing on I/O Lines”.

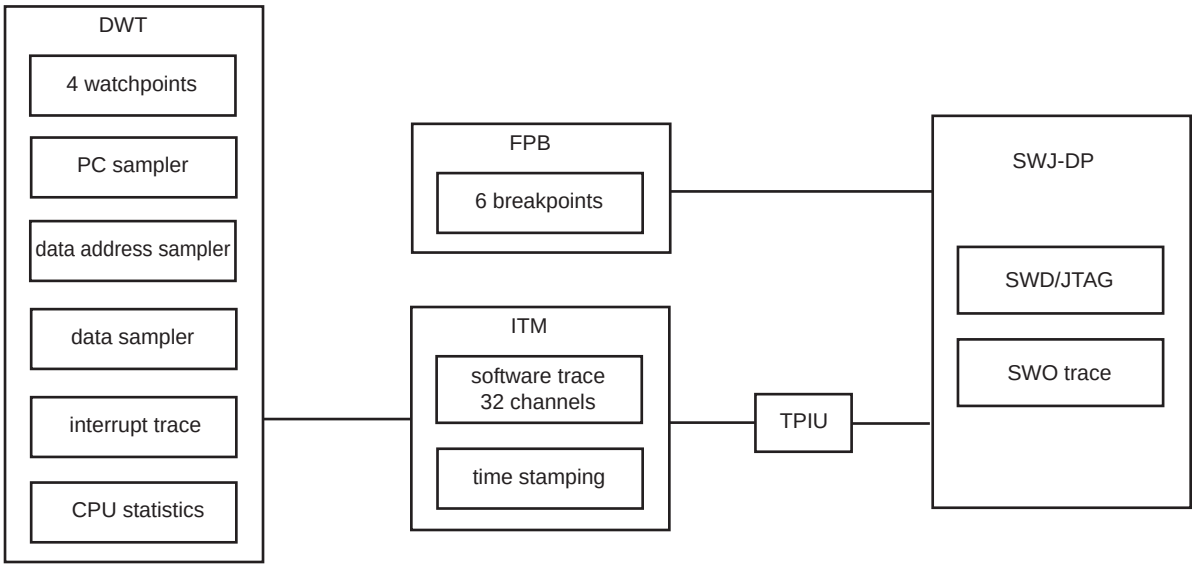
11.6.4 Debug Architecture

Figure 11-4 shows the Debug Architecture used in the SAM G55. The Cortex-M4 embeds five functional units for debug:

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

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

Figure 11-4. Debug Architecture



11.6.5 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.

11.6.5.1 SW-DP and JTAG-DP Selection Mechanism

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

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

- Send the 16-bit sequence on SWDIOTMS = 0011110011100111 (0x3CE7 MSB first)
- Send more than 50 SWCLKTCK cycles with SWDIOTMS = 1

11.6.6 FPB (Flash Patch Breakpoint)

The FPB:

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

The FPB unit contains:

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

11.6.7 DWT (Data Watchpoint and Trace)

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

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

The DWT contains counters for the items that follow:

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

11.6.8 ITM (Instrumentation Trace Macrocell)

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

- Software trace: Software can write directly to ITM stimulus registers. This can be done thanks to the “printf” function. For more information, refer to Section 11.6.8.1 “How to Configure the ITM”.
- Hardware trace: The ITM emits packets generated by the DWT.
- Time stamping: Timestamps are emitted relative to packets. The ITM contains a 21-bit counter to generate the timestamp.

11.6.8.1 How to Configure the ITM

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

- Configure the TPIU for asynchronous trace mode (refer to Section 11.6.8.3 “How to Configure the TPIU”).
- Enable the write accesses into the ITM registers by writing “0xC5ACCE55” into the Lock Access Register (address: 0xE0000FB0).
- Write 0x00010015 into the Trace Control Register:
  - Enable ITM.
  - Enable synchronization packets.

- Enable SWO behavior.
- Fix the ATB ID to 1.

- Write 0x1 into the Trace Enable Register:
  - Enable the stimulus port 0.
- Write 0x1 into the Trace Privilege Register:
  - Stimulus port 0 only accessed in privileged mode (clearing a bit in this register will result in the corresponding stimulus port being accessible in user mode).
- Write into the Stimulus Port 0 Register: TPIU (Trace Port Interface Unit).
  - The TPIU acts as a bridge between the on-chip trace data and the Instruction Trace Macrocell (ITM).
  - The TPIU formats and transmits trace data off-chip at frequencies asynchronous to the core.

11.6.8.2 Asynchronous Mode

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

Two encoding formats are available for the single pin output:

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

11.6.8.3 How to Configure the TPIU

This example only concerns the asynchronous trace mode.

- Set the TRCENA bit to 1 into the Debug Exception and Monitor Register (0xE000EDFC) to enable the use of trace and debug blocks.
- Write 0x2 into the Selected Pin Protocol Register.
  - Select the Serial Wire Output – NRZ.
- Write 0x100 into the Formatter and Flush Control Register.
- Set the suitable clock prescaler value into the Async Clock Prescaler Register to scale the baud rate of the asynchronous output (this can be done automatically by the debugging tool).

11.6.9 IEEE 1149.1 JTAG Boundary Scan

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

IEEE1149.1 JTAG Boundary Scan is enabled when TST is tied to high, PD0 tied to low, and JTAGSEL tied to high during powerup. These pins must be maintained in their respective states for the duration of the boundary scan operation. The SAMPLE, EXTEST and BYPASS functions are implemented. In SWD/JTAG debug mode, the ARM processor responds with a non-JTAG chip ID that identifies the processor. This is not IEEE 1149.1 JTAG-compliant.

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

11.6.9.1 JTAG Boundary-scan Register

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

Each SAM G55 input/output pin corresponds to a 3-bit field 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 SAM G55.

11.6.10 ID Code Register

Access: Read-only

31	30	29	28	27	26	25	24
VERSION				PART NUMBER			

23	22	21	20	19	18	17	16
PART NUMBER							

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

7	6	5	4	3	2	1	0
MANUFACTURER IDENTITY							1

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

Set to 0x0.

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

PART NUMBER
0x05B3E

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

Set to 0x01F.

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

Set to 0x1.

JTAG ID Code
0x05B3_E03F

12. Chip Identifier (CHIPID)

12.1 Description

Chip Identifier (CHIPID) registers are used to recognize the device and its revision. These registers provide the sizes and types of the on-chip memories, as well as the set of embedded peripherals.

Two CHIPID registers are embedded: Chip ID Register (CHIPID_CIDR) and Chip ID Extension Register (CHIPID_EXID). Both registers contain a hard-wired value that is read-only.

The CHIPID_CIDR register contains the following fields:

- VERSION: Identifies the revision of the silicon
- EPROC: Indicates the embedded ARM processor
- NVPTYP and NVPSIZ: Identify the type of embedded non-volatile memory and the size
- SRAMSIZ: Indicates the size of the embedded SRAM
- ARCH: Identifies the set of embedded peripherals
- EXT: Shows the use of the extension identifier register

The CHIPID_EXID register is device-dependent and reads 0 if CHIPID_CIDR.EXT = 0.

12.2 Embedded Characteristics

- Chip ID Registers
  - Identification of the Device Revision, Sizes of the Embedded Memories, Set of Peripherals, Embedded Processor

Table 12-1. SAM G55 Chip ID Registers

Chip Name	CHIPID_CIDR	CHIPID_EXID
SAM G55G19 Rev. A	0x2447_0AE0	0x0
SAM G55J19 Rev. A	0x2457_0AE0	0x0
SAM G55G19 Rev. B	0x2447_0AE1	0x0
SAM G55J19 Rev. B	0x2457_0AE1	0x0

12.3 Chip Identifier (CHIPID) User Interface

Table 12-2. Register Mapping

Offset	Register	Name	Access	Reset
0x0	Chip ID Register	CHIPID_CIDR	Read-only	–
0x4	Chip ID Extension Register	CHIPID_EXID	Read-only	–

12.3.1 Chip ID Register

Name: CHIPID_CIDR

Address: 0x400E0740

Access: Read-only

31	30	29	28	27	26	25	24
EXT	NVPTYP			ARCH			

23	22	21	20	19	18	17	16
ARCH				SRAMSIZ			

15	14	13	12	11	10	9	8
NVPSIZ2				NVPSIZ			

7	6	5	4	3	2	1	0
EPROC			VERSION				

- **VERSION: Version of the Device**

Current version of the device.

- **EPROC: Embedded Processor**

Value	Name	Description
0	SAM x7	Cortex-M7
1	ARM946ES	ARM946ES
2	ARM7TDMI	ARM7TDMI
3	CM3	Cortex-M3
4	ARM920T	ARM920T
5	ARM926EJS	ARM926EJS
6	CA5	Cortex-A5
7	CM4	Cortex-M4

- **NVPSIZ: Nonvolatile Program Memory Size**

Value	Name	Description
0	NONE	None
1	8K	8 Kbytes
2	16K	16 Kbytes
3	32K	32 Kbytes
4	–	Reserved
5	64K	64 Kbytes
6	–	Reserved
7	128K	128 Kbytes
8	160K	160 Kbytes
9	256K	256 Kbytes
10	512K	512 Kbytes

Value	Name	Description
11	–	Reserved
12	1024K	1024 Kbytes
13	–	Reserved
14	2048K	2048 Kbytes
15	–	Reserved

- **NVPSIZ2: Second Nonvolatile Program Memory Size**

Value	Name	Description
0	NONE	None
1	8K	8 Kbytes
2	16K	16 Kbytes
3	32K	32 Kbytes
4	–	Reserved
5	64K	64 Kbytes
6	–	Reserved
7	128K	128 Kbytes
8	–	Reserved
9	256K	256 Kbytes
10	512K	512 Kbytes
11	–	Reserved
12	1024K	1024 Kbytes
13	–	Reserved
14	2048K	2048 Kbytes
15	–	Reserved

- **SRAMSIZ: Internal SRAM Size**

Value	Name	Description
0	48K	48 Kbytes
1	192K	192 Kbytes
2	384K	384 Kbytes
3	6K	6 Kbytes
4	24K	24 Kbytes
5	4K	4 Kbytes
6	80K	80 Kbytes
7	160K	160 Kbytes
8	8K	8 Kbytes
9	16K	16 Kbytes
10	32K	32 Kbytes
11	64K	64 Kbytes

Value	Name	Description
12	128K	128 Kbytes
13	256K	256 Kbytes
14	96K	96 Kbytes
15	512K	512 Kbytes

- **ARCH: Architecture Identifier**

Value	Name	Description
0x44	SAM G55	SAM G55 (49-lead version)
0x45	SAM G55	SAM G55 (64-lead version)

- **NVPTYP: Nonvolatile Program Memory Type**

Value	Name	Description
0	ROM	ROM
1	ROMLESS	ROMless or on-chip Flash
2	FLASH	Embedded Flash Memory
3	ROM_FLASH	ROM and Embedded Flash Memory • NVPSIZ is ROM size • NVPSIZ2 is Flash size
4	SRAM	SRAM emulating ROM

- **EXT: Extension Flag**

0: Chip ID has a single register definition without extension.

1: An extended Chip ID exists.

12.3.2 Chip ID Extension Register

Name: CHIPID_EXID

Address: 0x400E0744

Access: Read-only

31	30	29	28	27	26	25	24
EXID							

23	22	21	20	19	18	17	16
EXID							

15	14	13	12	11	10	9	8
EXID							

7	6	5	4	3	2	1	0
EXID							

- **EXID: Chip ID Extension**

This field is cleared if CHIPID_CIDR.EXT = 0.

13. ARM Cortex-M4 Processor

13.1 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.

13.1.1 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.

13.1.2 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.

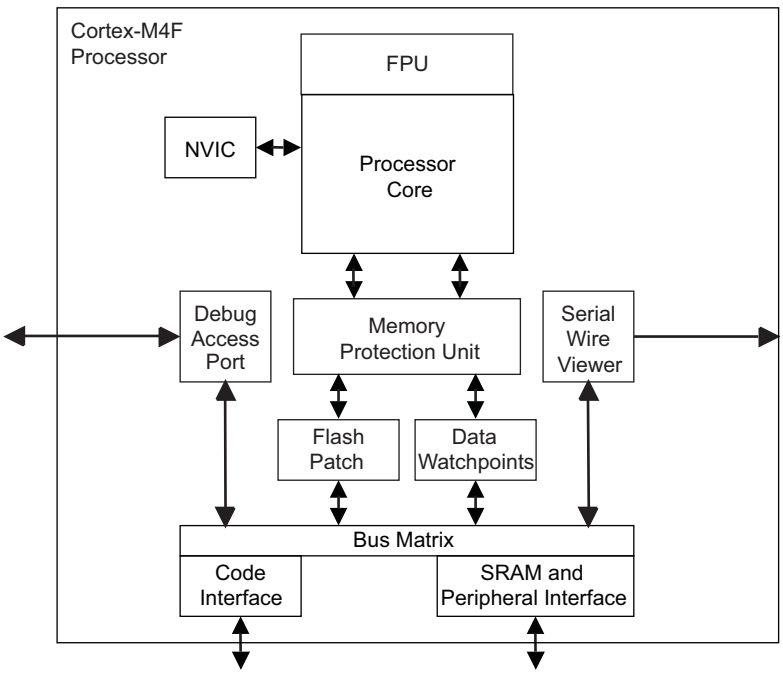
The Flash Patch and Breakpoint Unit (FPB) provides up to eight hardware breakpoint comparators that debuggers can use. The comparators in the FPB also provide remap functions of up to eight words in the program code in the CODE memory region. This enables applications stored on a non-erasable, ROM-based 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.

13.2 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.

13.3 Block Diagram

Figure 13-1. Typical Cortex-M4F Implementation



13.4 Cortex-M4 Models

13.4.1 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.

13.4.1.1 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.

13.4.1.2 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 13-1. 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".

13.4.1.3 Core Registers

Figure 13-2. Processor Core Registers

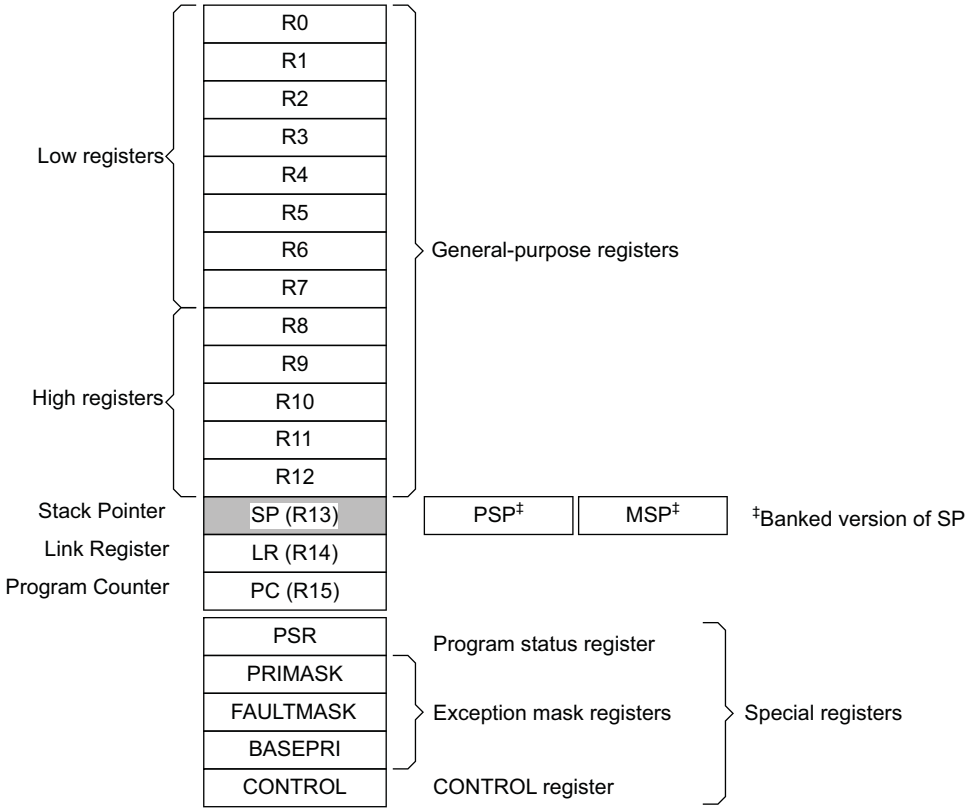


Table 13-2. Core Processor Registers

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.

13.4.1.4 General-purpose Registers

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

13.4.1.5 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.

13.4.1.6 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.

13.4.1.7 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.

13.4.1.8 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.

13.4.1.9 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.

13.4.1.10 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 = IRQ49

See “Exception Types” for more information.

13.4.1.11 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.

13.4.1.12 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.

13.4.1.13 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.

13.4.1.14 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.

13.4.1.15 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.

13.4.1.16 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 13-10.

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”.

13.4.1.17 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.

13.4.1.18 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.

13.4.1.19 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.

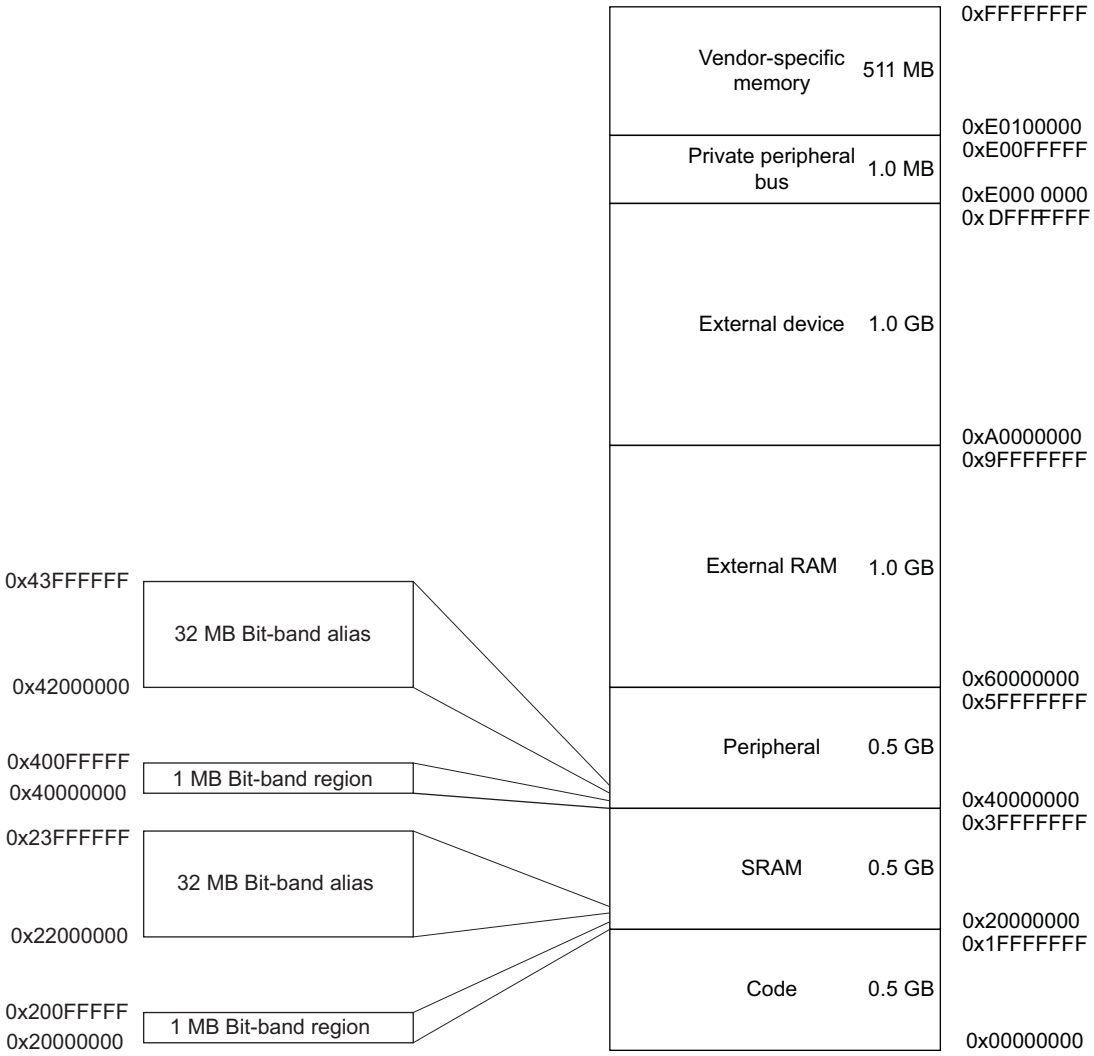
The following sections give more information about the CMSIS:

- Section 13.5.3 "Power Management Programming Hints"
- Section 13.6.2 "CMSIS Functions"
- Section 13.8.2.1 "NVIC Programming Hints".

13.4.2 Memory Model

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

Figure 13-3. Memory Map



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

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

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

13.4.2.1 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.

13.4.2.2 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 13-3. 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.

13.4.2.3 Behavior of Memory Accesses

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

Table 13-4. 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 13-6.
0x40000000–0x5FFFFFFF	Peripheral	Device(1)	XN	This region includes bit band and bit band alias areas, see Table 13-6.
0x60000000–0x9FFFFFFF	External RAM	Normal(1)	–	Executable region for data
0xA0000000–0xDFFFFFFF	External device	Device(1)	XN	External Device memory
0xE0000000–0xE00FFFFF	Private Peripheral Bus	Strongly-ordered(1)	XN	This region includes the NVIC, system timer, and system control block.
0xE0100000–0xFFFFFFFF	Reserved	Device(1)	XN	Reserved

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

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

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

Additional Memory Access Constraints For Caches and Shared Memory

When a system includes caches or shared memory, some memory regions have additional access constraints, and some regions are subdivided, as Table 13-5 shows.

Table 13-5. Memory Region Shareability and Cache Policies

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

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

2. WT = Write through, no write allocate. WBWA = Write back, write allocate. See the "Glossary" for more information.

Instruction Prefetch and Branch Prediction

The Cortex-M4 processor:

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

13.4.2.4 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.

13.4.2.5 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 13-6.
- Accesses to the 32 MB peripheral alias region map to the 1 MB peripheral bit-band region, as shown in Table 13-7.

Table 13-6. 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 13-7. 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.

Table 13-7. Peripheral Memory Bit-banding Regions (Continued)

Address Range	Memory Region	Instruction and Data Accesses
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
```

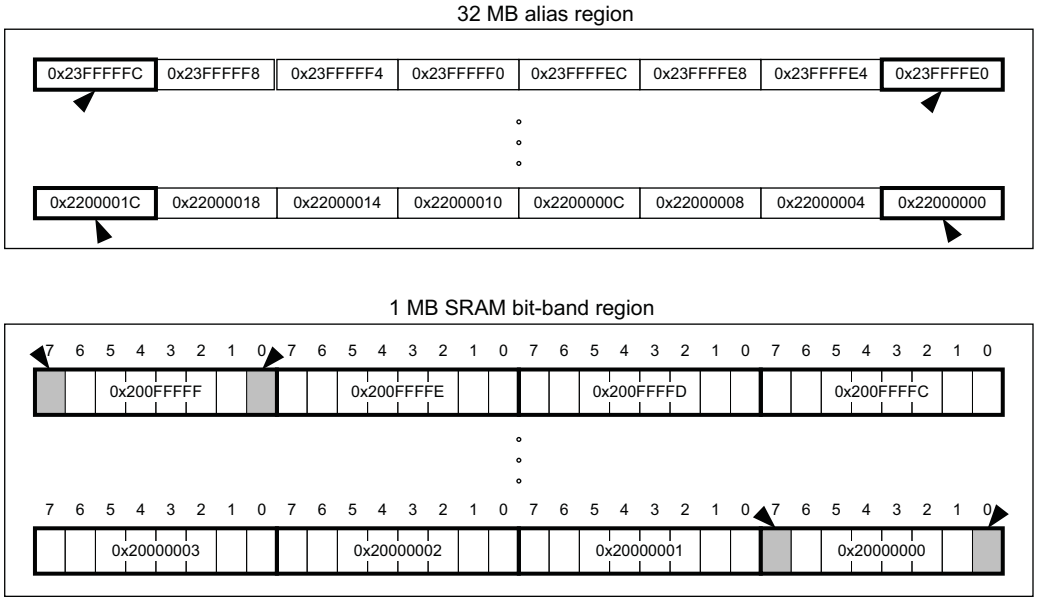
where:

- `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 13-4 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 13-4. 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.

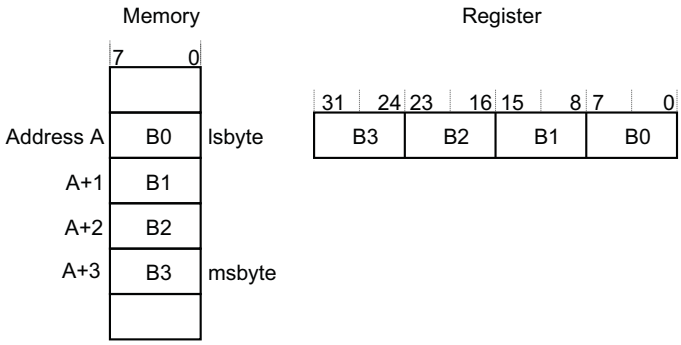
13.4.2.6 Memory Endianness

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 13-5. *Little-endian Format*



13.4.2.7 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”.

13.4.2.8 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 13-8. 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);
```

13.4.3 Exception Model

This section describes the exception model.

13.4.3.1 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.

13.4.3.2 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 13-9. 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 13-9 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".

13.4.3.3 Exception Handlers

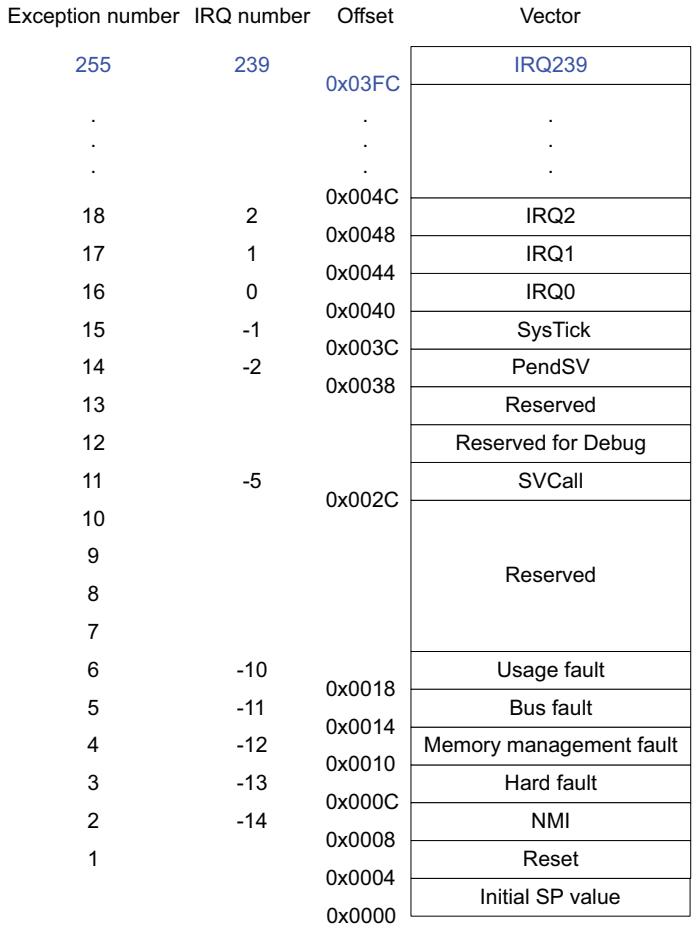
The processor handles exceptions using:

- Interrupt Service Routines (ISRs)
  Interrupts IRQ0 to IRQ49 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.

13.4.3.4 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 13-6 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 13-6. 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".

13.4.3.5 Exception Priorities

As Table 13-9 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.

13.4.3.6 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”.

13.4.3.7 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.

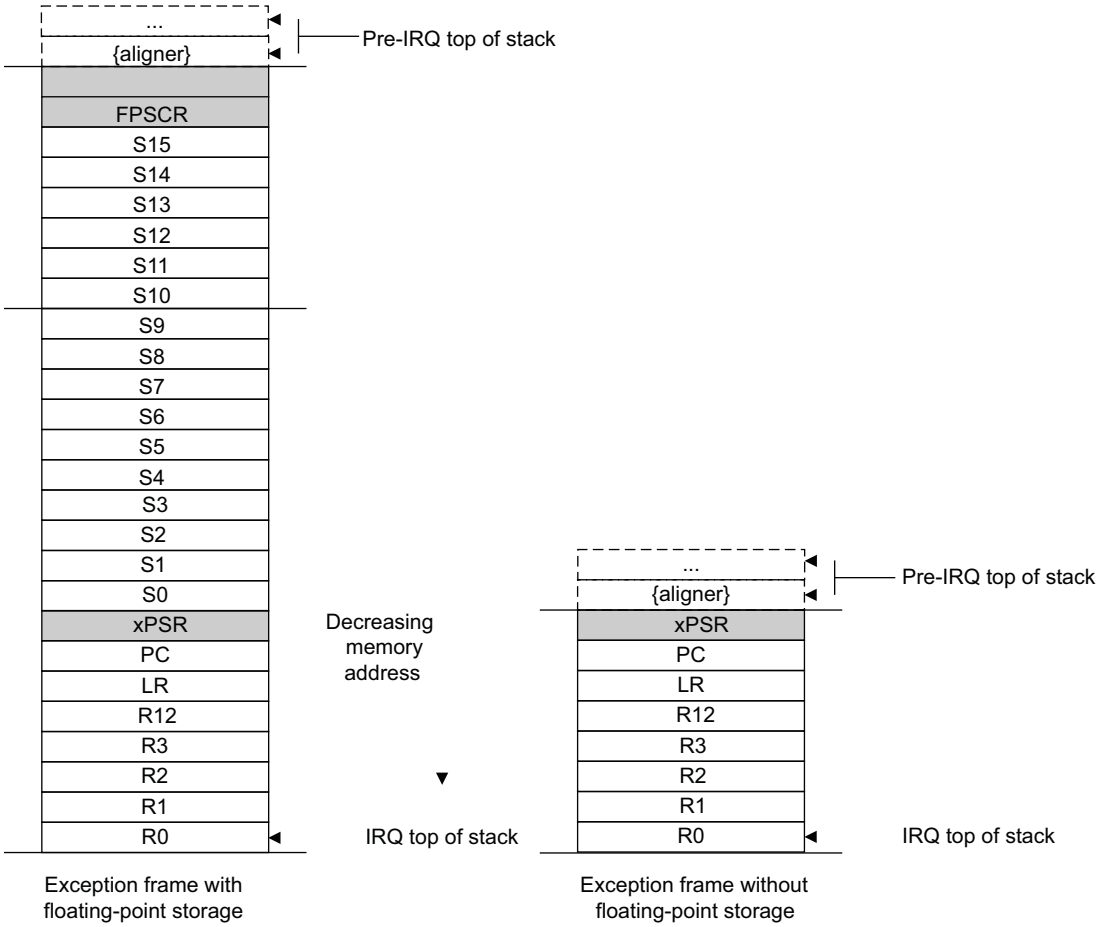
Sufficient priority means that the exception has more priority than any limits set by the mask registers, see "Exception Mask Registers". An exception with less priority than this is pending but is not handled by the processor.

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 13-7 shows the Cortex-M4 stack frame layout when floating-point state is preserved on the stack as the result of an interrupt or an exception.

Note: Where stack space for floating-point state is not allocated, the stack frame is the same as that of ARMv7-M implementations without an FPU. Figure 13-7 shows this stack frame also.

Figure 13-7. 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 13-10 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 13-10. 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.

13.4.3.8 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 13-11 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 13-11. Faults

Fault	Handler	Bit Name	Fault Status Register
Bus error on a vector read	Hard fault	VECTTBL	"Hard Fault Status Register"
Fault escalated to a hard fault		FORCED	
MPU or default memory map mismatch:	Memory management fault	–	–
on instruction access		IACCVIOL(1)	"MMFSR: Memory Management Fault Status Subregister"
on data access		DACCVIOL(2)	
during exception stacking		MSTKERR	
during exception unstacking		MUNSTKERR	
during lazy floating-point state preservation		MLSPERR(3)	
Bus error:	Bus fault	–	–
during exception stacking		STKERR	"BFSR: Bus Fault Status Subregister"
during exception unstacking		UNSTKERR	
during instruction prefetch		IBUSERR	
during lazy floating-point state preservation		LSPERR(3)	
Precise data bus error		PRECISERR	
Imprecise data bus error		IMPRECISERR	
Attempt to access a coprocessor	Usage fault	NOCP	"UFSR: Usage Fault Status Subregister"
Undefined instruction		UNDEFINSTR	
Attempt to enter an invalid instruction set state		INVSTATE	
Invalid EXC_RETURN value		INVPC	
Illegal unaligned load or store		UNALIGNED	
Divide By 0		DIVBYZERO	

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

Fault Escalation and Hard Faults

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

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

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

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

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

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

Fault Status Registers and Fault Address Registers

The fault status registers indicate the cause of a fault. For bus faults and memory management faults, the fault address register indicates the address accessed by the operation that caused the fault, as shown in Table 13-12.

Table 13-12. 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.

13.5 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.

13.5.1 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.

13.5.1.1 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.

13.5.1.2 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.

13.5.1.3 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.

13.5.2 Wakeup from Sleep Mode

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

13.5.2.1 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".

13.5.2.2 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”.

13.5.2.3 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.

13.5.3 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
```

13.6 Cortex-M4 Instruction Set

13.6.1 Instruction Set Summary

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

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

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

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

13.6.2 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 13-14. 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 13-15. 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)

13.6.3 Instruction Descriptions

13.6.3.1 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”.

13.6.3.2 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.

13.6.3.3 Flexible Second Operand

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

Operand2 can be a:

- “Constant”
- “Register with Optional Shift”

Constant

Specify an Operand2 constant in the form:

```
#constant
```

where *constant* can be:

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

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

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

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

Instruction Substitution

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

Register with Optional Shift

Specify an Operand2 register in the form:

```
Rm {, shift}
```

where:

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

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

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

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

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

13.6.3.4 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

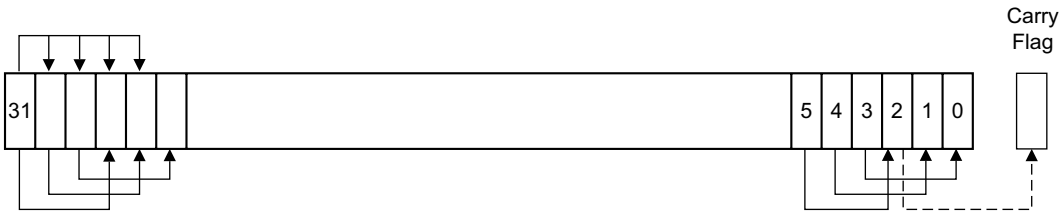
Arithmetic shift right by *n* bits moves the left-hand 32-n bits of the register, *Rm*, to the right by *n* places, into the right-hand 32-n bits of the result. And it copies the original bit[31] of the register into the left-hand *n* bits of the result. See Figure 13-8.

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 13-8. ASR #3



LSR

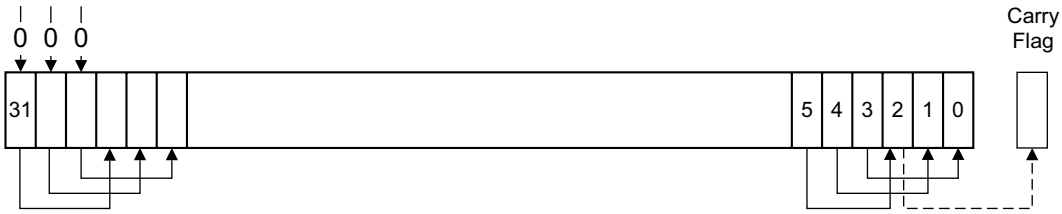
Logical shift right by n bits moves the left-hand 32-n bits of the register Rm, to the right by *n* places, into the right-hand 32-n bits of the result. And it sets the left-hand n bits of the result to 0. See Figure 13-9.

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 13-9. LSR #3



LSL

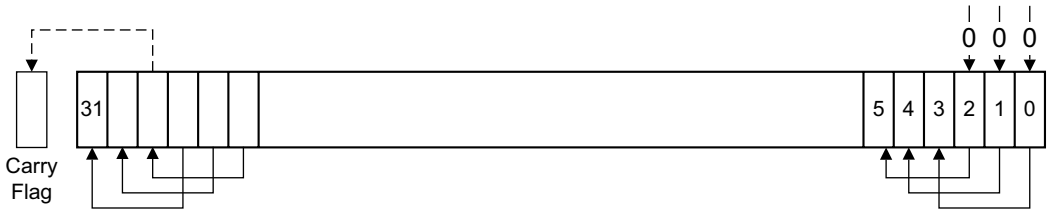
Logical shift left by *n* bits moves the right-hand 32-n bits of the register Rm, to the left by *n* places, into the left-hand 32-n bits of the result; and it sets the right-hand *n* bits of the result to 0. See Figure 13-10.

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 13-10. LSL #3



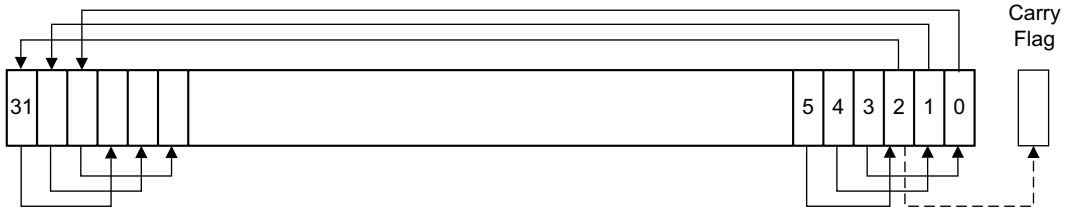
ROR

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 13-11.

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 13-11. ROR #3

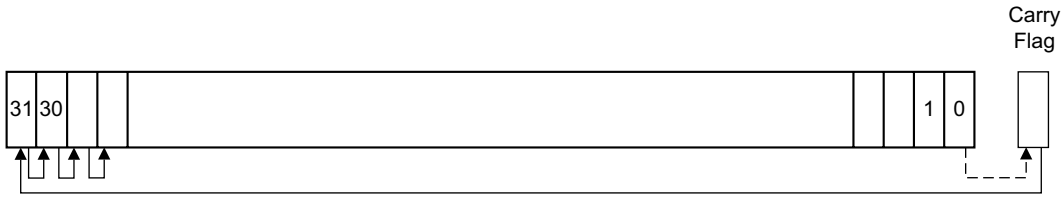


RRX

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 13-12.

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 13-12. RRX



13.6.3.5 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".

13.6.3.6 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].

13.6.3.7 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 13-16 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 13-16 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 13-16 also shows the relationship between condition code suffixes and the N, Z, C, and V flags.

Table 13-16. 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
```

13.6.3.8 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
```

13.6.4 Memory Access Instructions

The table below shows the memory access instructions.

Table 13-17. 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

13.6.4.1 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
```

13.6.4.2 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 13-18. 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.
```

13.6.4.3 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
```

13.6.4.4 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
```

13.6.4.5 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 13-19. 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
```

13.6.4.6 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
```

13.6.4.7 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}
```

13.6.4.8 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
```

13.6.4.9 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
```

13.6.5 General Data Processing Instructions

The table below shows the data processing instructions.

Table 13-20. Data Processing Instructions

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

Table 13-20. Data Processing Instructions (Continued)

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

13.6.5.1 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
```

13.6.5.2 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
```

13.6.5.3 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.
```

13.6.5.4 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
```

13.6.5.5 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
```

13.6.5.6 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.
```

13.6.5.7 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.
```

13.6.5.8 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.
```

13.6.5.9 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.
```

13.6.5.10 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.
```

13.6.5.11 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.
```

13.6.5.12 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.
```

13.6.5.13 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.
```

13.6.5.14 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.
```

13.6.5.15 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.
```

13.6.5.16 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.
```

13.6.5.17 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.
```

13.6.5.18 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.
```

13.6.5.19 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.
```

13.6.5.20 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.
```

13.6.5.21 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.
```

13.6.5.22 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.
```

13.6.5.23 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.
```

13.6.5.24 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.
```

13.6.6 Multiply and Divide Instructions

The table below shows the multiply and divide instructions.

Table 13-21. 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

13.6.6.1 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)
```

13.6.6.2 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.
```

13.6.6.3 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.
```

13.6.6.4 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.
```

13.6.6.5 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.
```

13.6.6.6 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.
```

13.6.6.7 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.
```

13.6.6.8 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.
```

13.6.6.9 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.
```

13.6.6.10 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 multiply operand.
If *Y* is B, then the bottom halfword (bits [15:0]) of *Rm* is used.
If *Y* is T, then the top halfword (bits [31:16]) of *Rm* is used.

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

Rd is the destination register.

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

Operation

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

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

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

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

Restrictions

In these instructions:

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

Examples

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

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

13.6.6.11 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
```

13.6.6.12 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
```

13.6.7 Saturating Instructions

The table below shows the saturating instructions.

Table 13-22. 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”.

13.6.7.1 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.
```

13.6.7.2 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.
```

13.6.7.3 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.
```

13.6.7.4 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.
```

13.6.7.5 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.
```

13.6.7.6 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.
```

13.6.7.7 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.
```

13.6.8 Packing and Unpacking Instructions

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

Table 13-23. 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

13.6.8.1 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.
```

13.6.8.2 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.
```

13.6.8.3 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.
```

13.6.9 Bitfield Instructions

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

Table 13-24. 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

13.6.9.1 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.
```

13.6.9.2 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.
```

13.6.9.3 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.
```

13.6.10 Branch and Control Instructions

The table below shows the branch and control instructions.

Table 13-25. 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

13.6.10.1 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 13-26. 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.
```

13.6.10.2 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
```

13.6.10.3 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
```

13.6.10.4 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
```

13.6.11 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 13-27. 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 13-27. 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

13.6.11.1 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
```

13.6.11.2 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
```

13.6.11.3 VCMP, VCMPE

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

Syntax

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

where:

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

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

Sd — is the floating-point operand to compare.

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

Operation

This instruction:

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

Restrictions

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

Condition Flags

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

Examples

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

13.6.11.4 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
   - Convert a value in a register from floating-point value to a 32-bit integer.
   - Convert from a 32-bit integer to floating-point value.
2. Place the result in a second register.

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

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

Restrictions

There are no restrictions.

Condition Flags

These instructions do not change the flags.

13.6.11.5 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.

13.6.11.6 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.

13.6.11.7 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.

13.6.11.8 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.

13.6.11.9 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.

13.6.11.10 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.

13.6.11.11 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.

13.6.11.12 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.

13.6.11.13 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.

13.6.11.14 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.

13.6.11.15 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.

13.6.11.16 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.

13.6.11.17 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.

13.6.11.18 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.

13.6.11.19 VMRS

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

Syntax

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

where:

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

Rt is the destination ARM core register. This register can be R0–R14.

APSR_nzcv transfers floating-point flags to the APSR flags.

Operation

This instruction performs one of the following actions:

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

Restrictions

Rt cannot be PC or SP.

Condition Flags

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

13.6.11.20 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.

13.6.11.21 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.

13.6.11.22 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.

13.6.11.23 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.

13.6.11.24 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.

13.6.11.25 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 sur
rounded 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.

13.6.11.26 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.

13.6.11.27 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.

13.6.11.28 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.

13.6.11.29 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.

13.6.12 Miscellaneous Instructions

The table below shows the remaining Cortex-M4 instructions.

Table 13-28. 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

13.6.12.1 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.

13.6.12.2 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)
```

13.6.12.3 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
```

13.6.12.4 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
```

13.6.12.5 ISB

Instruction Synchronization Barrier.

Syntax

```
ISB{cond}
```

where:

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

Operation

ISB acts as an instruction synchronization barrier. It flushes the pipeline of the processor, so that all instructions following the ISB are fetched from cache or memory again, after the ISB instruction has been completed.

Condition Flags

This instruction does not change the flags.

Examples

```
ISB  ; Instruction Synchronisation Barrier
```

13.6.12.6 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
```

13.6.12.7 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
```

13.6.12.8 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
```

13.6.12.9 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
```

13.6.12.10 SVC

Supervisor Call.

Syntax

`SVC{cond} #imm`

where:

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

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

Operation

The SVC instruction causes the SVC exception.

imm is ignored by the processor. If required, it can be retrieved by the exception handler to determine what service is being requested.

Condition Flags

This instruction does not change the flags.

Examples

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

13.6.12.11 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
```

13.6.12.12 WFI

Wait for Interrupt.

Syntax

`WFI{cond}`

where:

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

Operation

WFI is a hint instruction that suspends execution until one of the following events occurs:

- An exception
- A Debug Entry request, regardless of whether Debug is enabled.

Condition Flags

This instruction does not change the flags.

Examples

```
WFI ; Wait for interrupt
```

13.7 Cortex-M4 Core Peripherals

13.7.1 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 13.8 ”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 13.9 ”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 13.10 ”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 13.11 ”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 13.12 ”Floating Point Unit (FPU)”.

13.7.2 Address Map

The address map of the *Private peripheral bus* (PPB) is given in the following table.

Table 13-29. 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.

13.8 Nested Vectored Interrupt Controller (NVIC)

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

- Up to 50 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.

13.8.1 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.

13.8.1.1 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.

13.8.2 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”.

13.8.2.1 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 13-30. CMSIS Functions for NVIC Control

CMSIS Interrupt Control Function	Description
void NVIC_SetPriorityGrouping(uint32_t priority_grouping)	Set the priority grouping
void NVIC_EnableIRQ(IRQn_t IRQn)	Enable IRQn
void NVIC_DisableIRQ(IRQn_t IRQn)	Disable IRQn
uint32_t NVIC_GetPendingIRQ (IRQn_t IRQn)	Return true (IRQ-Number) if IRQn is pending
void NVIC_SetPendingIRQ (IRQn_t IRQn)	Set IRQn pending
void NVIC_ClearPendingIRQ (IRQn_t IRQn)	Clear IRQn pending status
uint32_t NVIC_GetActive (IRQn_t IRQn)	Return the IRQ number of the active interrupt
void NVIC_SetPriority (IRQn_t IRQn, uint32_t priority)	Set priority for IRQn
uint32_t NVIC_GetPriority (IRQn_t IRQn)	Read priority of IRQn
void NVIC_SystemReset (void)	Reset the system

The input parameter IRQn is the IRQ number. For more information about these functions, see the CMSIS documentation.

To improve software efficiency, the CMSIS simplifies the NVIC register presentation. In the CMSIS:

- The Set-enable, Clear-enable, Set-pending, Clear-pending and Active Bit registers map to arrays of 32-bit integers, so that:
  - The array ISER[0] to ISER[1] corresponds to the registers ISER0–ISER1
  - The array ICER[0] to ICER[1] corresponds to the registers ICER0–ICER1
  - The array ISPR[0] to ISPR[1] corresponds to the registers ISPR0–ISPR1
  - The array ICPR[0] to ICPR[1] corresponds to the registers ICPR0–ICPR1
  - The array IABR[0] to IABR[1] corresponds to the registers IABR0–IABR1
- The Interrupt Priority Registers (IPR0–IPR12) provide an 8-bit priority field for each interrupt and each register holds four priority fields.

The CMSIS provides thread-safe code that gives atomic access to the Interrupt Priority Registers. Table 13-31 shows how the interrupts, or IRQ numbers, map onto the interrupt registers and corresponding CMSIS variables that have one bit per interrupt.

Table 13-31. Mapping of Interrupts

Interrupts	CMSIS Array Elements[1]				
	Set-enable	**Clear-enable**	**Set-pending**	**Clear-pending**	**Active Bit**
0–31	ISER[0]	ICER[0]	ISPR[0]	ICPR[0]	IABR[0]
32–50	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.

13.8.3 Nested Vectored Interrupt Controller (NVIC) User Interface

Table 13-32. Nested Vectored Interrupt Controller (NVIC) Register Mapping

Offset	Register	Name	Access	Reset
0xE000E100	Interrupt Set-enable Register 0	NVIC_ISER0	Read/Write	0x00000000
...	...	...	...	...
0xE000E11C	Interrupt Set-enable Register 7	NVIC_ISER7	Read/Write	0x00000000
0XE000E180	Interrupt Clear-enable Register 0	NVIC_ICER0	Read/Write	0x00000000
...	...	...	...	...
0xE000E19C	Interrupt Clear-enable Register 7	NVIC_ICER7	Read/Write	0x00000000
0XE000E200	Interrupt Set-pending Register 0	NVIC_ISPR0	Read/Write	0x00000000
...	...	...	...	...
0xE000E21C	Interrupt Set-pending Register 7	NVIC_ISPR7	Read/Write	0x00000000
0XE000E280	Interrupt Clear-pending Register 0	NVIC_ICPR0	Read/Write	0x00000000
...	...	...	...	...
0xE000E29C	Interrupt Clear-pending Register 7	NVIC_ICPR7	Read/Write	0x00000000
0xE000E300	Interrupt Active Bit Register 0	NVIC_IABR0	Read/Write	0x00000000
...	...	...	...	...
0xE000E31C	Interrupt Active Bit Register 7	NVIC_IABR7	Read/Write	0x00000000
0xE000E400	Interrupt Priority Register 0	NVIC_IPR0	Read/Write	0x00000000
...	...	...	...	...
0xE000E42C	Interrupt Priority Register 12	NVIC_IPR12	Read/Write	0x00000000
0xE000EF00	Software Trigger Interrupt Register	NVIC_STIR	Write-only	0x00000000

13.8.3.1 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.

13.8.3.2 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.

13.8.3.3 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.

13.8.3.4 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.

13.8.3.5 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.

13.8.3.6 Interrupt Priority Registers

Name: NVIC_IPRx [x=0..12]

Access: Read/Write

Reset: 0x000000000

31	30	29	28	27	26	25	24
PRI3							

23	22	21	20	19	18	17	16
PRI2							

15	14	13	12	11	10	9	8
PRI1							

7	6	5	4	3	2	1	0
PRI0							

The NVIC_IPR0–NVIC_IPR12 registers provide a 8-bit priority field for each interrupt. These registers are byte-accessible. Each register holds four priority fields that map up to four elements in the CMSIS interrupt priority array IP[0] to IP[49].

- **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[49] interrupt priority array, that provides the software view of the interrupt priorities, see Table 13-30 “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.

13.8.3.7 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.

13.9 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.

13.9.1 System Control Block (SCB) User Interface

Table 13-33. 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).

13.9.1.1 Auxiliary Control Register

Name: SCB_ACTLR

Access: Read/Write

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

The SCB_ACTLR provides disable bits for the following processor functions:

- IT folding
- Write buffer use for accesses to the default memory map
- Interruption of multi-cycle instructions.

By default, this register is set to provide optimum performance from the Cortex-M4 processor, and does not normally require modification.

- **DISOOFP: Disable Out Of Order Floating Point**

Disables floating point instructions that complete out of order with respect to integer instructions.

- **DISFPCA: Disable FPCA**

Disables an automatic update of CONTROL.FPCA.

- **DISFOLD: Disable Folding**

When set to 1, disables the IT folding.

Note: In some situations, the processor can start executing the first instruction in an IT block while it is still executing the IT instruction. This behavior is called IT folding, and it improves the performance. However, IT folding can cause jitter in looping. If a task must avoid jitter, set the DISFOLD bit to 1 before executing the task, to disable the IT folding.

- **DISDEFWBUF: Disable Default Write Buffer**

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

This bit only affects write buffers implemented in the Cortex-M4 processor.

- **DISMCYCINT: Disable Multiple Cycle Interruption**

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

13.9.1.2 CPUID Base Register

Name: SCB_CPUID

Access: Read/Write

31	30	29	28	27	26	25	24
Implementer							

23	22	21	20	19	18	17	16
Variant				Constant			

15	14	13	12	11	10	9	8
PartNo							

7	6	5	4	3	2	1	0
PartNo				Revision			

The SCB_CPUID register contains the processor part number, version, and implementation information.

- **Implementer: Implementer Code**

0x41: ARM.

- **Variant: Variant Number**

It is the r value in the rnpn product revision identifier:

0x0: Revision 0.

- **Constant: Reads as 0xF**

Reads as 0xF.

- **PartNo: Part Number of the Processor**

0xC24 = Cortex-M4.

- **Revision: Revision Number**

It is the p value in the rnpn product revision identifier:

0x0: Patch 0.

13.9.1.3 Interrupt Control and State Register

Name: SCB_ICSR

Access: Read/Write

31	30	29	28	27	26	25	24
NMIPENDSET	–		PENDSVSET	PENDSVCLR	PENDSTSET	PENDSTCLR	–
23	22	21	20	19	18	17	16
–	ISRPENDING	VECTPENDING					
15	14	13	12	11	10	9	8
VECTPENDING				RETTOBASE	–	–	VECTACTIVE
7	6	5	4	3	2	1	0
VECTACTIVE							

The SCB_ICSR provides a set-pending bit for the Non-Maskable Interrupt (NMI) exception, and set-pending and clear-pending bits for the PendSV and SysTick exceptions.

It indicates:

- The exception number of the exception being processed, and whether there are preempted active exceptions,
- The exception number of the highest priority pending exception, and whether any interrupts are pending.

- **NMIPENDSET: NMI Set-pending**

Write:

PendSV set-pending bit.

Write:

0: No effect.

1: Changes NMI exception state to pending.

Read:

0: NMI exception is not pending.

1: NMI exception is pending.

As NMI is the highest-priority exception, the processor normally enters the NMI exception handler as soon as it registers a write of 1 to this bit. Entering the handler clears this bit to 0. A read of this bit by the NMI exception handler returns 1 only if the NMI signal is reasserted while the processor is executing that handler.

- **PENDSVSET: PendSV Set-pending**

Write:

0: No effect.

1: Changes PendSV exception state to pending.

Read:

0: PendSV exception is not pending.

1: PendSV exception is pending.

Writing a 1 to this bit is the only way to set the PendSV exception state to pending.

- **PENDSVCLR: PendSV Clear-pending**

Write:

0: No effect.

1: Removes the pending state from the PendSV exception.

- **PENDSTSET: SysTick Exception Set-pending**

Write:

0: No effect.

1: Changes SysTick exception state to pending.

Read:

0: SysTick exception is not pending.

1: SysTick exception is pending.

- **PENDSTCLR: SysTick Exception Clear-pending**

Write:

0: No effect.

1: Removes the pending state from the SysTick exception.

This bit is Write-only. On a register read, its value is Unknown.

- **ISRPENDING: Interrupt Pending Flag (Excluding NMI and Faults)**

0: Interrupt not pending.

1: Interrupt pending.

- **VECTPENDING: Exception Number of the Highest Priority Pending Enabled Exception**

0: No pending exceptions.

Nonzero: The exception number of the highest priority pending enabled exception.

The value indicated by this field includes the effect of the BASEPRI and FAULTMASK registers, but not any effect of the PRIMASK register.

- **RETTOBASE: Preempted Active Exceptions Present or Not**

0: There are preempted active exceptions to execute.

1: There are no active exceptions, or the currently-executing exception is the only active exception.

- **VECTACTIVE: Active Exception Number Contained**

0: Thread mode.

Nonzero: The exception number of the currently active exception. The value is the same as IPSR bits [8:0]. See “Interrupt Program Status Register”.

Subtract 16 from this value to obtain the IRQ number required to index into the Interrupt Clear-Enable, Set-Enable, Clear-Pending, Set-Pending, or Priority Registers, see “Interrupt Program Status Register”.

Note: When the user writes to the SCB_ICSR, the effect is unpredictable if:
- Writing a 1 to the PENDSVSET bit and writing a 1 to the PENDSVCLR bit
- Writing a 1 to the PENDSTSET bit and writing a 1 to the PENDSTCLR bit.

13.9.1.4 Vector Table Offset Register

Name: SCB_VTOR

Access: Read/Write

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

The SCB_VTOR indicates the offset of the vector table base address from memory address 0x00000000.

- **TBLOFF: Vector Table Base Offset**

It contains bits [29:7] of the offset of the table base from the bottom of the memory map.

Bit [29] determines whether the vector table is in the code or SRAM memory region:

0: Code.

1: SRAM.

It is sometimes called the TBLBASE bit.

Note: When setting TBLOFF, the offset must be aligned to the number of exception entries in the vector table. Configure the next statement to give the information required for your implementation; the statement reminds the user of how to determine the alignment requirement. The minimum alignment is 32 words, enough for up to 16 interrupts. For more interrupts, adjust the alignment by rounding up to the next power of two. For example, if 21 interrupts are required, the alignment must be on a 64-word boundary because the required table size is 37 words, and the next power of two is 64.

Table alignment requirements mean that bits[6:0] of the table offset are always zero.

13.9.1.5 Application Interrupt and Reset Control Register

Name: SCB_AIRCR

Access: Read/Write

31	30	29	28	27	26	25	24
VECTKEYSTAT/VECTKEY							
23	22	21	20	19	18	17	16
VECTKEYSTAT/VECTKEY							
15	14	13	12	11	10	9	8
ENDIANNESS	–	–	–	–	PRIGROUP		
7	6	5	4	3	2	1	0
–	–	–	–	–	SYSRESETREQ	VECTCLRACTIVE	VECTRESET

The SCB_AIRCR provides priority grouping control for the exception model, endian status for data accesses, and reset control of the system. To write to this register, write 0x5FA to the VECTKEY field, otherwise the processor ignores the write.

- **VECTKEYSTAT: Register Key (Read)**

Reads as 0xFA05.

- **VECTKEY: Register Key (Write)**

Writes 0x5FA to VECTKEY, otherwise the write is ignored.

- **ENDIANNESS: Data Endianness**

0: Little-endian.

1: Big-endian.

- **PRIGROUP: Interrupt Priority Grouping**

This field determines the split of group priority from subpriority. It shows the position of the binary point that splits the PRI_*n* fields in the Interrupt Priority Registers into separate *group priority* and *subpriority* fields. The table below shows how the PRIGROUP value controls this split.

	Interrupt Priority Level Value, PRI_*N*[7:0]			Number of	
PRIGROUP	**Binary Point**[1]	**Group Priority Bits**	**Subpriority Bits**	**Group Priorities**	**Subpriorities**
0b000	bxxxxxxx.y	[7:1]	None	128	2
0b001	bxxxxxx.yy	[7:2]	[4:0]	64	4
0b010	bxxxxx.yyy	[7:3]	[4:0]	32	8
0b011	bxxxx.yyyy	[7:4]	[4:0]	16	16
0b100	bxxx.yyyyy	[7:5]	[4:0]	8	32
0b101	bxx.yyyyyy	[7:6]	[5:0]	4	64
0b110	bx.yyyyyyy	[7]	[6:0]	2	128
0b111	b.yyyyyyy	None	[7:0]	1	256

Note: 1. PRI_n[7:0] field showing the binary point. x denotes a group priority field bit, and y denotes a subpriority field bit.

Determining preemption of an exception uses only the group priority field.

- **SYSRESETREQ: System Reset Request**

0: No system reset request.

1: Asserts a signal to the outer system that requests a reset.

This is intended to force a large system reset of all major components except for debug. This bit reads as 0.

- **VECTCLRACTIVE: Reserved for Debug use**

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

- **VECTRESET: Reserved for Debug use**

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

13.9.1.6 System Control Register

Name: SCB_SCR

Access: Read/Write

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

- **SEVONPEND: Send Event on Pending Bit**

0: Only enabled interrupts or events can wake up the processor; disabled interrupts are excluded.

1: Enabled events and all interrupts, including disabled interrupts, can wake up the processor.

When an event or an interrupt enters the pending state, the event signal wakes up the processor from WFE. If the processor is not waiting for an event, the event is registered and affects the next WFE.

The processor also wakes up on execution of an SEV instruction or an external event.

- **SLEEPDEEP: Sleep or Deep Sleep**

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

0: Sleep.

1: Deep sleep.

- **SLEEPONEXIT: Sleep-on-exit**

Indicates sleep-on-exit when returning from the Handler mode to the Thread mode:

0: Do not sleep when returning to Thread mode.

1: Enter sleep, or deep sleep, on return from an ISR.

Setting this bit to 1 enables an interrupt-driven application to avoid returning to an empty main application.

13.9.1.7 Configuration and Control Register

Name: SCB_CCR

Access: Read/Write

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	STKALIGN	BFHFNMIGN

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

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

- **STKALIGN: Stack Alignment**

Indicates the stack alignment on exception entry:

0: 4-byte aligned.

1: 8-byte aligned.

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

- **BFHFNMIGN: Bus Faults Ignored**

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

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

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

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

- **DIV_0_TRP: Division by Zero Trap**

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

0: Do not trap divide by 0.

1: Trap divide by 0.

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

- **UNALIGN_TRP: Unaligned Access Trap**

Enables unaligned access traps:

0: Do not trap unaligned halfword and word accesses.

1: Trap unaligned halfword and word accesses.

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

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

- **USERSETMPEND: Unprivileged Software Access**

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

0: Disable.

1: Enable.

- **NONBASETHRDENA: Thread Mode Enable**

Indicates how the processor enters Thread mode:

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

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

13.9.1.8 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 13-34. 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.

13.9.1.9 System Handler Priority Register 1

Name: SCB_SHPR1

Access: Read/Write

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

23	22	21	20	19	18	17	16
PRI_6							

15	14	13	12	11	10	9	8
PRI_5							

7	6	5	4	3	2	1	0
PRI_4							

- **PRI_6: Priority**

Priority of system handler 6, UsageFault.

- **PRI_5: Priority**

Priority of system handler 5, BusFault.

- **PRI_4: Priority**

Priority of system handler 4, MemManage.

13.9.1.10 System Handler Priority Register 2

Name: SCB_SHPR2

Access: Read/Write

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

- **PRI_11: Priority**

Priority of system handler 11, SVCall.

13.9.1.11 System Handler Priority Register 3

Name: SCB_SHPR3

Access: Read/Write

31	30	29	28	27	26	25	24
PRI_15							
23	22	21	20	19	18	17	16
PRI_14							
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

- **PRI_15: Priority**

Priority of system handler 15, SysTick exception.

- **PRI_14: Priority**

Priority of system handler 14, PendSV.

13.9.1.12 System Handler Control and State Register

Name: SCB_SHCSR

Access: Read/Write

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

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

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

7	6	5	4	3	2	1	0
SVCALLACT	–	–	–	USGFAULTACT	–	BUSFAULTACT	MEMFAULTACT

The SHCSR enables the system handlers, and indicates the pending status of the bus fault, memory management fault, and SVC exceptions; it also indicates the active status of the system handlers.

- **USGFAULTENA: Usage Fault Enable**

0: Disables the exception.

1: Enables the exception.

- **BUSFAULTENA: Bus Fault Enable**

0: Disables the exception.

1: Enables the exception.

- **MEMFAULTENA: Memory Management Fault Enable**

0: Disables the exception.

1: Enables the exception.

- **SVCALLPENDED: SVC Call Pending**

Read:

0: The exception is not pending.

1: The exception is pending.

Note: The user can write to these bits to change the pending status of the exceptions.

- **BUSFAULTPENDED: Bus Fault Exception Pending**

Read:

0: The exception is not pending.

1: The exception is pending.

Note: The user can write to these bits to change the pending status of the exceptions.

- **MEMFAULTPENDED: Memory Management Fault Exception Pending**

Read:

0: The exception is not pending.

1: The exception is pending.

Note: The user can write to these bits to change the pending status of the exceptions.

- **USGFAULTPENDED: Usage Fault Exception Pending**

Read:

0: The exception is not pending.

1: The exception is pending.

Note: The user can write to these bits to change the pending status of the exceptions.

- **SYSTICKACT: SysTick Exception Active**

Read:

0: The exception is not active.

1: The exception is active.

Note: The user can write to these bits to change the active status of the exceptions.
- Caution: A software that changes the value of an active bit in this register without a correct adjustment to the stacked content can cause the processor to generate a fault exception. Ensure that the software writing to this register retains and subsequently restores the current active status.
- Caution: After enabling the system handlers, to change the value of a bit in this register, the user must use a read-modify-write procedure to ensure that only the required bit is changed.

- **PENDSVACT: PendSV Exception Active**

0: The exception is not active.

1: The exception is active.

- **MONITORACT: Debug Monitor Active**

0: Debug monitor is not active.

1: Debug monitor is active.

- **SVCALLACT: SVC Call Active**

0: SVC call is not active.

1: SVC call is active.

- **USGFAULTACT: Usage Fault Exception Active**

0: Usage fault exception is not active.

1: Usage fault exception is active.

- **BUSFAULTACT: Bus Fault Exception Active**

0: Bus fault exception is not active.

1: Bus fault exception is active.

- **MEMFAULTACT: Memory Management Fault Exception Active**

0: Memory management fault exception is not active.

1: Memory management fault exception is active.

If the user disables a system handler and the corresponding fault occurs, the processor treats the fault as a hard fault.

The user can write to this register to change the pending or active status of system exceptions. An OS kernel can write to the active bits to perform a context switch that changes the current exception type.

13.9.1.13 Configurable Fault Status Register

Name: SCB_CFSR

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	DIVBYZERO	UNALIGNED
23	22	21	20	19	18	17	16
–	–	–	–	NOCP	INVPC	INVSTATE	UNDEFINSTR
15	14	13	12	11	10	9	8
BFARVALID	–	LSPERR	STKERR	UNSTKERR	IMPRECISERR	PRECISERR	IBUSERR
7	6	5	4	3	2	1	0
MMARVALID	–	MLSPERR	MSTKERR	MUNSTKERR	–	DACCVIOL	IACCVIOL

- **IACCVIOL: Instruction Access Violation Flag**

This is part of "MMFSR: Memory Management Fault Status Subregister".

0: No instruction access violation fault.

1: The processor attempted an instruction fetch from a location that does not permit execution.

This fault occurs on any access to an XN region, even when the MPU is disabled or not present.

When this bit is 1, the PC value stacked for the exception return points to the faulting instruction. The processor has not written a fault address to the SCB_MMFAR.

- **DACCVIOL: Data Access Violation Flag**

This is part of "MMFSR: Memory Management Fault Status Subregister".

0: No data access violation fault.

1: The processor attempted a load or store at a location that does not permit the operation.

When this bit is 1, the PC value stacked for the exception return points to the faulting instruction. The processor has loaded the SCB_MMFAR with the address of the attempted access.

- **MUNSTKERR: Memory Manager Fault on Unstacking for a Return From Exception**

This is part of "MMFSR: Memory Management Fault Status Subregister".

0: No unstacking fault.

1: Unstack for an exception return has caused one or more access violations.

This fault is chained to the handler. This means that when this bit is 1, the original return stack is still present. The processor has not adjusted the SP from the failing return, and has not performed a new save. The processor has not written a fault address to the SCB_MMFAR.

- **MSTKERR: Memory Manager Fault on Stacking for Exception Entry**

This is part of "MMFSR: Memory Management Fault Status Subregister".

0: No stacking fault.

1: Stacking for an exception entry has caused one or more access violations.

When this bit is 1, the SP is still adjusted but the values in the context area on the stack might be incorrect. The processor has not written a fault address to SCB_MMFAR.

- **MLSPERR: MemManage During Lazy State Preservation**

This is part of "MMFSR: Memory Management Fault Status Subregister".

0: No MemManage fault occurred during the floating-point lazy state preservation.

1: A MemManage fault occurred during the floating-point lazy state preservation.

- **MMARVALID: Memory Management Fault Address Register (SCB_MMFAR) Valid Flag**

This is part of "MMFSR: Memory Management Fault Status Subregister".

0: The value in SCB_MMFAR is not a valid fault address.

1: SCB_MMFAR holds a valid fault address.

If a memory management fault occurs and is escalated to a hard fault because of priority, the hard fault handler must set this bit to 0. This prevents problems on return to a stacked active memory management fault handler whose SCB_MMFAR value has been overwritten.

- **IBUSERR: Instruction Bus Error**

This is part of "BFSR: Bus Fault Status Subregister".

0: No instruction bus error.

1: Instruction bus error.

The processor detects the instruction bus error on prefetching an instruction, but it sets the IBUSERR flag to 1 only if it attempts to issue the faulting instruction.

When the processor sets this bit to 1, it does not write a fault address to the BFAR.

- **PRECISERR: Precise Data Bus Error**

This is part of "BFSR: Bus Fault Status Subregister".

0: No precise data bus error.

1: A data bus error has occurred, and the PC value stacked for the exception return points to the instruction that caused the fault.

When the processor sets this bit to 1, it writes the faulting address to the SCB_BFAR.

- **IMPRECISERR: Imprecise Data Bus Error**

This is part of "BFSR: Bus Fault Status Subregister".

0: No imprecise data bus error.

1: A data bus error has occurred, but the return address in the stack frame is not related to the instruction that caused the error.

When the processor sets this bit to 1, it does not write a fault address to the SCB_BFAR.

This is an asynchronous fault. Therefore, if it is detected when the priority of the current process is higher than the bus fault priority, the bus fault becomes pending and becomes active only when the processor returns from all higher priority processes. If a precise fault occurs before the processor enters the handler for the imprecise bus fault, the handler detects that both this bit and one of the precise fault status bits are set to 1.

- **UNSTKERR: Bus Fault on Unstacking for a Return From Exception**

This is part of "BFSR: Bus Fault Status Subregister".

0: No unstacking fault.

1: Unstack for an exception return has caused one or more bus faults.

This fault is chained to the handler. This means that when the processor sets this bit to 1, the original return stack is still present. The processor does not adjust the SP from the failing return, does not performed a new save, and does not write a fault address to the BFAR.

- **STKERR: Bus Fault on Stacking for Exception Entry**

This is part of “BFSR: Bus Fault Status Subregister”.

0: No stacking fault.

1: Stacking for an exception entry has caused one or more bus faults.

When the processor sets this bit to 1, the SP is still adjusted but the values in the context area on the stack might be incorrect. The processor does not write a fault address to the SCB_BFAR.

- **LSPERR: Bus Error During Lazy Floating-point State Preservation**

This is part of “BFSR: Bus Fault Status Subregister”.

0: No bus fault occurred during floating-point lazy state preservation

1: A bus fault occurred during floating-point lazy state preservation.

- **BFARVALID: Bus Fault Address Register (BFAR) Valid flag**

This is part of “BFSR: Bus Fault Status Subregister”.

0: The value in SCB_BFAR is not a valid fault address.

1: SCB_BFAR holds a valid fault address.

The processor sets this bit to 1 after a bus fault where the address is known. Other faults can set this bit to 0, such as a memory management fault occurring later.

If a bus fault occurs and is escalated to a hard fault because of priority, the hard fault handler must set this bit to 0. This prevents problems if returning to a stacked active bus fault handler whose SCB_BFAR value has been overwritten.

- **UNDEFINSTR: Undefined Instruction Usage Fault**

This is part of “UFSR: Usage Fault Status Subregister”.

0: No undefined instruction usage fault.

1: The processor has attempted to execute an undefined instruction.

When this bit is set to 1, the PC value stacked for the exception return points to the undefined instruction.

An undefined instruction is an instruction that the processor cannot decode.

- **INVSTATE: Invalid State Usage Fault**

This is part of “UFSR: Usage Fault Status Subregister”.

0: No invalid state usage fault.

1: The processor has attempted to execute an instruction that makes illegal use of the EPSR.

When this bit is set to 1, the PC value stacked for the exception return points to the instruction that attempted the illegal use of the EPSR.

This bit is not set to 1 if an undefined instruction uses the EPSR.

- **INVPC: Invalid PC Load Usage Fault**

This is part of "UFSR: Usage Fault Status Subregister". It is caused by an invalid PC load by EXC_RETURN:

0: No invalid PC load usage fault.

1: The processor has attempted an illegal load of EXC_RETURN to the PC, as a result of an invalid context, or an invalid EXC_RETURN value.

When this bit is set to 1, the PC value stacked for the exception return points to the instruction that tried to perform the illegal load of the PC.

- **NOCP: No Coprocessor Usage Fault**

This is part of "UFSR: Usage Fault Status Subregister". The processor does not support coprocessor instructions:

0: No usage fault caused by attempting to access a coprocessor.

1: The processor has attempted to access a coprocessor.

- **UNALIGNED: Unaligned Access Usage Fault**

This is part of "UFSR: Usage Fault Status Subregister".

0: No unaligned access fault, or unaligned access trapping not enabled.

1: The processor has made an unaligned memory access.

Enable trapping of unaligned accesses by setting the UNALIGN_TRP bit in the SCB_CCR to 1. See "Configuration and Control Register". Unaligned LDM, STM, LDRD, and STRD instructions always fault irrespective of the setting of UNALIGN_TRP.

- **DIVBYZERO: Divide by Zero Usage Fault**

This is part of "UFSR: Usage Fault Status Subregister".

0: No divide by zero fault, or divide by zero trapping not enabled.

1: The processor has executed an SDIV or UDIV instruction with a divisor of 0.

When the processor sets this bit to 1, the PC value stacked for the exception return points to the instruction that performed the divide by zero. Enable trapping of divide by zero by setting the DIV_0_TRP bit in the SCB_CCR to 1. See "Configuration and Control Register".

13.9.1.14 Configurable Fault Status Register (Byte Access)

Name: SCB_CFSR (BYTE)

Access: Read/Write

31	30	29	28	27	26	25	24
UFSR							

23	22	21	20	19	18	17	16
UFSR							

15	14	13	12	11	10	9	8
BFSR							

7	6	5	4	3	2	1	0
MMFSR							

- **MMFSR: Memory Management Fault Status Subregister**

The flags in the MMFSR subregister indicate the cause of memory access faults. See bitfield [7..0] description in Section 13.9.1.13.

- **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 13.9.1.13.

- **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 13.9.1.13.

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

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

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

13.9.1.15 Hard Fault Status Register

Name: SCB_HFSR

Access: Read/Write

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

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

- **DEBUGEVT: Reserved for Debug Use**

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

- **FORCED: Forced Hard Fault**

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

0: No forced hard fault.

1: Forced hard fault.

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

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

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

0: No bus fault on vector table read.

1: Bus fault on vector table read.

This error is always handled by the hard fault handler.

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

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

13.9.1.16 MemManage Fault Address Register

Name: SCB_MMFAR

Access: Read/Write

31	30	29	28	27	26	25	24
ADDRESS							

23	22	21	20	19	18	17	16
ADDRESS							

15	14	13	12	11	10	9	8
ADDRESS							

7	6	5	4	3	2	1	0
ADDRESS							

The SCB_MMFAR contains the address of the location that generated a memory management fault.

- **ADDRESS: Memory Management Fault Generation Location Address**

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

Notes:
1. When an unaligned access faults, the address is the actual address that faulted. Because a single read or write instruction can be split into multiple aligned accesses, the fault address can be any address in the range of the requested access size.
2. Flags in the MMFSR subregister indicate the cause of the fault, and whether the value in the SCB_MMFAR is valid. See “MMFSR: Memory Management Fault Status Subregister”.

13.9.1.17 Bus Fault Address Register

Name: SCB_BFAR

Access: Read/Write

31	30	29	28	27	26	25	24
ADDRESS							

23	22	21	20	19	18	17	16
ADDRESS							

15	14	13	12	11	10	9	8
ADDRESS							

7	6	5	4	3	2	1	0
ADDRESS							

The SCB_BFAR contains the address of the location that generated a bus fault.

- **ADDRESS: Bus Fault Generation Location Address**

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

Notes:
1. When an unaligned access faults, the address in the SCB_BFAR is the one requested by the instruction, even if it is not the address of the fault.
2. Flags in the BFSR indicate the cause of the fault, and whether the value in the SCB_BFAR is valid. See “BFSR: Bus Fault Status Subregister”.

13.9.1.18 Auxiliary Fault Status Register

Name: SCB_AFSR

Access: Read/Write

31	30	29	28	27	26	25	24
IMPDEF							

23	22	21	20	19	18	17	16
IMPDEF							

15	14	13	12	11	10	9	8
IMPDEF							

7	6	5	4	3	2	1	0
IMPDEF							

The SCB_AFSR contains additional system fault information. This register is read, write to clear. This means that bits in the register read normally, but writing a 1 to any bit clears that bit to 0.

- **IMPDEF: Implementation Defined**

The bits map to the **AUXFAULT** input signals.

Notes:
1. Each AFSR bit directly maps to an **AUXFAULT** input of the processor, and a single-cycle HIGH signal on the input sets the corresponding AFSR bit to one. It remains set to 1 until the user writes a one to the bit to clear it to zero.
2. When an AFSR bit is latched as one, an exception does not occur. Use an interrupt if an exception is required.

13.10 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.

13.10.1 System Timer (SysTick) User Interface

Table 13-35. System Timer (SYST) Register Mapping

Offset	Register	Name	Access	Reset
0xE000E010	SysTick Control and Status Register	SYST_CSR	Read/Write	0x00000000
0xE000E014	SysTick Reload Value Register	SYST_RVR	Read/Write	Unknown
0xE000E018	SysTick Current Value Register	SYST_CVR	Read/Write	Unknown
0xE000E01C	SysTick Calibration Value Register	SYST_CALIB	Read-only	0x00001770

13.10.1.1 SysTick Control and Status Register

Name: SYST_CSR

Access: Read/Write

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

The SysTick SYST_CSR enables the SysTick features.

- **COUNTFLAG: Count Flag**

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

- **CLKSOURCE: Clock Source**

Indicates the clock source:

0: External Clock.

1: Processor Clock.

- **TICKINT: SysTick Exception Request Enable**

Enables a SysTick exception request:

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

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

The software can use COUNTFLAG to determine if SysTick has ever counted to zero.

- **ENABLE: Counter Enable**

Enables the counter:

0: Counter disabled.

1: Counter enabled.

When ENABLE is set to 1, the counter loads the RELOAD value from the SYST_RVR and then counts down. On reaching 0, it sets the COUNTFLAG to 1 and optionally asserts the SysTick depending on the value of TICKINT. It then loads the RELOAD value again, and begins counting.

13.10.1.2 SysTick Reload Value Registers

Name: SYST_RVR

Access: Read/Write

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

23	22	21	20	19	18	17	16
RELOAD							

15	14	13	12	11	10	9	8
RELOAD							

7	6	5	4	3	2	1	0
RELOAD							

The SYST_RVR specifies the start value to load into the SYST_CVR.

- **RELOAD: SYST_CVR Load Value**

Value to load into the SYST_CVR when the counter is enabled and when it reaches 0.

The RELOAD value can be any value in the range 0x00000001–0x00FFFFFF. A start value of 0 is possible, but has no effect because the SysTick exception request and COUNTFLAG are activated when counting from 1 to 0.

The RELOAD value is calculated according to its use: For example, to generate a multi-shot timer with a period of N processor clock cycles, use a RELOAD value of N-1. If the SysTick interrupt is required every 100 clock pulses, set RELOAD to 99.

13.10.1.3 SysTick Current Value Register

Name: SYST_CVR

Access: Read/Write

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

23	22	21	20	19	18	17	16
CURRENT							

15	14	13	12	11	10	9	8
CURRENT							

7	6	5	4	3	2	1	0
CURRENT							

The SysTick SYST_CVR contains the current value of the SysTick counter.

- **CURRENT: SysTick Counter Current Value**

Reads return the current value of the SysTick counter.

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

13.10.1.4 SysTick Calibration Value Register

Name: SYST_CALIB

Access: Read/Write

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

The SysTick SYST_CSR indicates the SysTick calibration properties.

- **NOREF: No Reference Clock**

It indicates whether the device provides a reference clock to the processor:

0: Reference clock provided.

1: No reference clock provided.

If your device does not provide a reference clock, the SYST_CSR.CLKSOURCE bit reads-as-one and ignores writes.

- **SKEW: TENMS Value Verification**

It indicates whether the TENMS value is exact:

0: TENMS value is exact.

1: TENMS value is inexact, or not given.

An inexact TENMS value can affect the suitability of SysTick as a software real time clock.

- **TENMS: Ten Milliseconds**

The reload value for 10 ms (100 Hz) timing is subject to system clock skew errors. If the value reads as zero, the calibration value is not known.

The TENMS field default value is 0x00001770 (12000 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).

13.11 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 13-36 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 13-36. Memory Attributes Summary

Memory Type	Shareability	Other Attributes	Description
Strongly-ordered	–	–	All accesses to Strongly-ordered memory occur in program order. All Strongly-ordered regions are assumed to be shared.
Device	Shared	–	Memory-mapped peripherals that several processors share.
	Non-shared	–	Memory-mapped peripherals that only a single processor uses.
Normal	Shared	Non-cacheable Write-through Cacheable Write-back Cacheable	Normal memory that is shared between several processors.
	Non-shared	Non-cacheable Write-through Cacheable Write-back Cacheable	Normal memory that only a single processor uses.

13.11.1 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 13-37. TEX, C, B, and S Encoding

TEX	C	B	S	Memory Type	Shareability	Other Attributes
b000	0	0	x(1)	Strongly-ordered	Shareable	–
		1	x(1)	Device	Shareable	–
	1	0	0	Normal	Not shareable	Outer and inner write-through. No write allocate.
			1		Shareable	
		1	0	Normal	Not shareable	Outer and inner write-back. No write allocate.
			1		Shareable	
b001	0	0	0	Normal	Not shareable	Outer and inner noncacheable.
			1		Shareable	
		1	x(1)	Reserved encoding		–
	1	0	x(1)	Implementation defined attributes.		–
		1	0	Normal	Not shareable	Outer and inner write-back. Write and read allocate.
			1		Shareable	
b010	0	0	x(1)	Device	Not shareable	Nonshared Device.
		1	x(1)	Reserved encoding		–
	1	x(1)	x(1)	Reserved encoding		–
b1BB	A	A	0	Normal	Not shareable	Cached memory BB = outer policy, AA = inner policy.
			1		Shareable	

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

Table 13-38 shows the cache policy for memory attribute encodings with a TEX value is in the range 4–7.

Table 13-38. 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 13-39 shows the AP encodings that define the access permissions for privileged and unprivileged software.

Table 13-39. 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

13.11.1.1 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.

13.11.1.2 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.

13.11.1.3 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.

13.11.1.4 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
```

13.11.1.5 Subregions

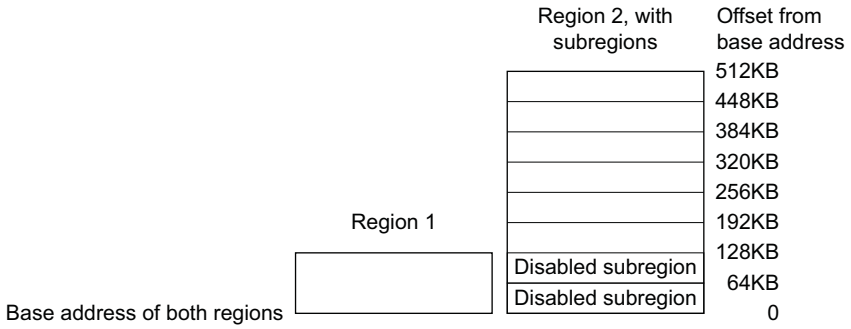
Regions of 256 bytes or more are divided into eight equal-sized subregions. Set the corresponding bit in the SRD field of the MPU_RASR field to disable a subregion. See "MPU Region Attribute and Size Register". The least significant bit of SRD controls the first subregion, and the most significant bit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

13.11.1.6 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 13-13 below:

Figure 13-13. SRD Use



13.11.1.7 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 13-40. 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.

13.11.2 Memory Protection Unit (MPU) User Interface

Table 13-41. 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

13.11.2.1 MPU Type Register

Name: MPU_TYPE

Access: Read/Write

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

23	22	21	20	19	18	17	16
IREGION							

15	14	13	12	11	10	9	8
DREGION							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	SEPARATE

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

- **IREGION: Instruction Region**

Indicates the number of supported MPU instruction regions.

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

- **DREGION: Data Region**

Indicates the number of supported MPU data regions:

0x08 = Eight MPU regions.

- **SEPARATE: Separate Instruction**

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

0: Unified.

13.11.2.2 MPU Control Register

Name: MPU_CTRL

Access: Read/Write

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

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

- **PRIVDEFENA: Privileged Default Memory Map Enable**

Enables privileged software access to the default memory map:

0: If the MPU is enabled, disables the use of the default memory map. Any memory access to a location not covered by any enabled region causes a fault.

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

When enabled, the background region acts as a region number -1. Any region that is defined and enabled has priority over this default map.

If the MPU is disabled, the processor ignores this bit.

- **HFNMIENA: Hard Fault and NMI Enable**

Enables the operation of MPU during hard fault, NMI, and FAULTMASK handlers.

When the MPU is enabled:

0: MPU is disabled during hard fault, NMI, and FAULTMASK handlers, regardless of the value of the ENABLE bit.

1: The MPU is enabled during hard fault, NMI, and FAULTMASK handlers.

When the MPU is disabled, if this bit is set to 1, the behavior is unpredictable.

- **ENABLE: MPU Enable**

Enables the MPU:

0: MPU disabled.

1: MPU enabled.

When ENABLE and PRIVDEFENA are both set to 1:

- For privileged accesses, the *default memory map* is as described in "Memory Model". Any access by privileged software that does not address an enabled memory region behaves as defined by the default memory map.
- Any access by unprivileged software that does not address an enabled memory region causes a memory management fault.

XN and Strongly-ordered rules always apply to the System Control Space regardless of the value of the ENABLE bit.

When the ENABLE bit is set to 1, at least one region of the memory map must be enabled for the system to function unless the PRIVDEFENA bit is set to 1. If the PRIVDEFENA bit is set to 1 and no regions are enabled, then only privileged software can operate.

When the ENABLE bit is set to 0, the system uses the default memory map. This has the same memory attributes as if the MPU is not implemented. The default memory map applies to accesses from both privileged and unprivileged software.

When the MPU is enabled, accesses to the System Control Space and vector table are always permitted. Other areas are accessible based on regions and whether PRIVDEFENA is set to 1.

Unless HFNMIENA is set to 1, the MPU is not enabled when the processor is executing the handler for an exception with priority –1 or –2. These priorities are only possible when handling a hard fault or NMI exception, or when FAULTMASK is enabled. Setting the HFNMIENA bit to 1 enables the MPU when operating with these two priorities.

13.11.2.3 MPU Region Number Register

Name: MPU_RNR

Access: Read/Write

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

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

- **REGION: MPU Region Referenced by the MPU_RBAR and MPU_RASRs**

Indicates the MPU region referenced by the MPU_RBAR and MPU_RASRs.

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

Normally, the required region number is written to this register before accessing the MPU_RBAR or MPU_RASR. However, the region number can be changed by writing to the MPU_RBAR with the VALID bit set to 1; see “MPU Region Base Address Register”. This write updates the value of the REGION field.

13.11.2.4 MPU Region Base Address Register

Name: MPU_RBAR

Access: Read/Write

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR			VALID	REGION			

The MPU_RBAR defines the base address of the MPU region selected by the MPU_RNR, and can update the value of the MPU_RNR.

Write MPU_RBAR with the VALID bit set to 1 to change the current region number and update the MPU_RNR.

- **ADDR: Region Base Address**

Software must ensure that the value written to the ADDR field aligns with the size of the selected region (SIZE field in the MPU_RASR).

If the region size is configured to 4 GB, in the MPU_RASR, there is no valid ADDR field. In this case, the region occupies the complete memory map, and the base address is 0x00000000.

The base address is aligned to the size of the region. For example, a 64 KB region must be aligned on a multiple of 64 KB, for example, at 0x00010000 or 0x00020000.

- **VALID: MPU Region Number Valid**

Write:

0: MPU_RNR not changed, and the processor updates the base address for the region specified in the MPU_RNR, and ignores the value of the REGION field.

1: The processor updates the value of the MPU_RNR to the value of the REGION field, and updates the base address for the region specified in the REGION field.

Always reads as zero.

- **REGION: MPU Region**

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

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

13.11.2.5 MPU Region Attribute and Size Register

Name: MPU_RASR

Access: Read/Write

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

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

15	14	13	12	11	10	9	8
SRD							

7	6	5	4	3	2	1	0
–	–	SIZE					ENABLE

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

MPU_RASR is accessible using word or halfword accesses:

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

- **XN: Instruction Access Disable**

0: Instruction fetches enabled.

1: Instruction fetches disabled.

- **AP: Access Permission**

See Table 13-39.

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

See Table 13-37.

- **S: Shareable**

See Table 13-37.

- **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”.

13.11.2.6 MPU Region Base Address Register Alias 1

Name: MPU_RBAR_A1

Access: Read/Write

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR			VALID	REGION			

The MPU_RBAR defines the base address of the MPU region selected by the MPU_RNR, and can update the value of the MPU_RNR.

Write MPU_RBAR with the VALID bit set to 1 to change the current region number and update the MPU_RNR.

- **ADDR: Region Base Address**

Software must ensure that the value written to the ADDR field aligns with the size of the selected region.

The value of N depends on the region size. The ADDR field is bits[31:N] of the MPU_RBAR. The region size, as specified by the SIZE field in the MPU_RASR, defines the value of N:

N = Log2(Region size in bytes),

If the region size is configured to 4 GB, in the MPU_RASR, there is no valid ADDR field. In this case, the region occupies the complete memory map, and the base address is 0x00000000.

The base address is aligned to the size of the region. For example, a 64 KB region must be aligned on a multiple of 64 KB, for example, at 0x00010000 or 0x00020000.

- **VALID: MPU Region Number Valid**

Write:

0: MPU_RNR not changed, and the processor updates the base address for the region specified in the MPU_RNR, and ignores the value of the REGION field.

1: The processor updates the value of the MPU_RNR to the value of the REGION field, and updates the base address for the region specified in the REGION field.

Always reads as zero.

- **REGION: MPU Region**

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

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

13.11.2.7 MPU Region Attribute and Size Register Alias 1

Name: MPU_RASR_A1

Access: Read/Write

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

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

15	14	13	12	11	10	9	8
SRD							

7	6	5	4	3	2	1	0
–	–	SIZE					ENABLE

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

MPU_RASR is accessible using word or halfword accesses:

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

- **XN: Instruction Access Disable**

0: Instruction fetches enabled.

1: Instruction fetches disabled.

- **AP: Access Permission**

See Table 13-39.

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

See Table 13-37.

- **S: Shareable**

See Table 13-37.

- **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”.

13.11.2.8 MPU Region Base Address Register Alias 2

Name: MPU_RBAR_A2

Access: Read/Write

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR			VALID	REGION			

The MPU_RBAR defines the base address of the MPU region selected by the MPU_RNR, and can update the value of the MPU_RNR.

Write MPU_RBAR with the VALID bit set to 1 to change the current region number and update the MPU_RNR.

- **ADDR: Region Base Address**

Software must ensure that the value written to the ADDR field aligns with the size of the selected region.

The value of N depends on the region size. The ADDR field is bits[31:N] of the MPU_RBAR. The region size, as specified by the SIZE field in the MPU_RASR, defines the value of N:

N = Log2(Region size in bytes),

If the region size is configured to 4 GB, in the MPU_RASR, there is no valid ADDR field. In this case, the region occupies the complete memory map, and the base address is 0x00000000.

The base address is aligned to the size of the region. For example, a 64 KB region must be aligned on a multiple of 64 KB, for example, at 0x00010000 or 0x00020000.

- **VALID: MPU Region Number Valid**

Write:

0: MPU_RNR not changed, and the processor updates the base address for the region specified in the MPU_RNR, and ignores the value of the REGION field.

1: The processor updates the value of the MPU_RNR to the value of the REGION field, and updates the base address for the region specified in the REGION field.

Always reads as zero.

- **REGION: MPU Region**

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

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

13.11.2.9 MPU Region Attribute and Size Register Alias 2

Name: MPU_RASR_A2

Access: Read/Write

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

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

15	14	13	12	11	10	9	8
SRD							

7	6	5	4	3	2	1	0
–	–	SIZE					ENABLE

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

MPU_RASR is accessible using word or halfword accesses:

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

- **XN: Instruction Access Disable**

0: Instruction fetches enabled.

1: Instruction fetches disabled.

- **AP: Access Permission**

See Table 13-39.

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

See Table 13-37.

- **S: Shareable**

See Table 13-37.

- **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”.

13.11.2.10 MPU Region Base Address Register Alias 3

Name: MPU_RBAR_A3

Access: Read/Write

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR			VALID	REGION			

The MPU_RBAR defines the base address of the MPU region selected by the MPU_RNR, and can update the value of the MPU_RNR.

Write MPU_RBAR with the VALID bit set to 1 to change the current region number and update the MPU_RNR.

- **ADDR: Region Base Address**

Software must ensure that the value written to the ADDR field aligns with the size of the selected region.

The value of N depends on the region size. The ADDR field is bits[31:N] of the MPU_RBAR. The region size, as specified by the SIZE field in the MPU_RASR, defines the value of N:

N = Log2(Region size in bytes),

If the region size is configured to 4 GB, in the MPU_RASR, there is no valid ADDR field. In this case, the region occupies the complete memory map, and the base address is 0x00000000.

The base address is aligned to the size of the region. For example, a 64 KB region must be aligned on a multiple of 64 KB, for example, at 0x00010000 or 0x00020000.

- **VALID: MPU Region Number Valid**

Write:

0: MPU_RNR not changed, and the processor updates the base address for the region specified in the MPU_RNR, and ignores the value of the REGION field.

1: The processor updates the value of the MPU_RNR to the value of the REGION field, and updates the base address for the region specified in the REGION field.

Always reads as zero.

- **REGION: MPU Region**

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

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

13.11.2.11 MPU Region Attribute and Size Register Alias 3

Name: MPU_RASR_A3

Access: Read/Write

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

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

15	14	13	12	11	10	9	8
SRD							

7	6	5	4	3	2	1	0
–	–	SIZE					ENABLE

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

MPU_RASR is accessible using word or halfword accesses:

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

- **XN: Instruction Access Disable**

0: Instruction fetches enabled.

1: Instruction fetches disabled.

- **AP: Access Permission**

See Table 13-39.

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

See Table 13-37.

- **S: Shareable**

See Table 13-37.

- **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”.

13.12 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.

13.12.1 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
```

13.12.2 Floating Point Unit (FPU) User Interface

Table 13-42. 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

13.12.2.1 Coprocessor Access Control Register

Name: CPACR

Access: Read/Write

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

23	22	21	20	19	18	17	16
CP11		CP10		–	–	–	–

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

The CPACR specifies the access privileges for coprocessors.

- **CP10: Access Privileges for Coprocessor 10**

The possible values of each field are:

0: Access denied. Any attempted access generates a NOCP UsageFault.

1: Privileged access only. An unprivileged access generates a NOCP fault.

2: Reserved. The result of any access is unpredictable.

3: Full access.

- **CP11: Access Privileges for Coprocessor 11**

The possible values of each field are:

0: Access denied. Any attempted access generates a NOCP UsageFault.

1: Privileged access only. An unprivileged access generates a NOCP fault.

2: Reserved. The result of any access is unpredictable.

3: Full access.

13.12.2.2 Floating-point Context Control Register

Name: FPCCR

Access: Read/Write

31	30	29	28	27	26	25	24
ASPEN	LSPEN	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	MONRDY
7	6	5	4	3	2	1	0
–	BFRDY	MMRDY	HFRDY	THREAD	–	USER	LSPACT

The FPCCR sets or returns FPU control data.

- **ASPEN: Automatic Hardware State Preservation And Restoration**

Enables CONTROL bit [2] setting on execution of a floating-point instruction. This results in an automatic hardware state preservation and restoration, for floating-point context, on exception entry and exit.

0: Disable CONTROL bit [2] setting on execution of a floating-point instruction.

1: Enable CONTROL bit [2] setting on execution of a floating-point instruction.

- **LSPEN: Automatic Lazy State Preservation**

0: Disable automatic lazy state preservation for floating-point context.

1: Enable automatic lazy state preservation for floating-point context.

- **MONRDY: Debug Monitor Ready**

0: DebugMonitor is disabled or the priority did not permit to set MON_PEND when the floating-point stack frame was allocated.

1: DebugMonitor is enabled and the priority permitted to set MON_PEND when the floating-point stack frame was allocated.

- **BFRDY: Bus Fault Ready**

0: BusFault is disabled or the priority did not permit to set the BusFault handler to the pending state when the floating-point stack frame was allocated.

1: BusFault is enabled and the priority permitted to set the BusFault handler to the pending state when the floating-point stack frame was allocated.

- **MMRDY: Memory Management Ready**

0: MemManage is disabled or the priority did not permit to set the MemManage handler to the pending state when the floating-point stack frame was allocated.

1: MemManage is enabled and the priority permitted to set the MemManage handler to the pending state when the floating-point stack frame was allocated.

- **HFRDY: Hard Fault Ready**

0: The priority did not permit to set the HardFault handler to the pending state when the floating-point stack frame was allocated.

1: The priority permitted to set the HardFault handler to the pending state when the floating-point stack frame was allocated.

- **THREAD: Thread Mode**

0: The mode was not the Thread Mode when the floating-point stack frame was allocated.

1: The mode was the Thread Mode when the floating-point stack frame was allocated.

- **USER: User Privilege Level**

0: The privilege level was not User when the floating-point stack frame was allocated.

1: The privilege level was User when the floating-point stack frame was allocated.

- **LSPACT: Lazy State Preservation Active**

0: The lazy state preservation is not active.

1: The lazy state preservation is active. The floating-point stack frame has been allocated but saving the state to it has been deferred.

13.12.2.3 Floating-point Context Address Register

Name: FPCAR

Access: Read/Write

31	30	29	28	27	26	25	24
ADDRESS							

23	22	21	20	19	18	17	16
ADDRESS							

15	14	13	12	11	10	9	8
ADDRESS							

7	6	5	4	3	2	1	0
ADDRESS					–	–	–

The FPCAR holds the location of the unpopulated floating-point register space allocated on an exception stack frame.

- **ADDRESS: Location of Unpopulated Floating-point Register Space Allocated on an Exception Stack Frame**

The location of the unpopulated floating-point register space allocated on an exception stack frame.

13.12.2.4 Floating-point Status Control Register

Name: FPSCR

Access: Read/Write

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

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

The FPSCR provides all necessary User level control of the floating-point system.

- **N: Negative Condition Code Flag**

Floating-point comparison operations update this flag.

- **Z: Zero Condition Code Flag**

Floating-point comparison operations update this flag.

- **C: Carry Condition Code Flag**

Floating-point comparison operations update this flag.

- **V: Overflow Condition Code Flag**

Floating-point comparison operations update this flag.

- **AHP: Alternative Half-precision Control**

0: IEEE half-precision format selected.

1: Alternative half-precision format selected.

- **DN: Default NaN Mode Control**

0: NaN operands propagate through to the output of a floating-point operation.

1: Any operation involving one or more NaNs returns the Default NaN.

- **FZ: Flush-to-zero Mode Control**

0: Flush-to-zero mode disabled. The behavior of the floating-point system is fully compliant with the IEEE 754 standard.

1: Flush-to-zero mode enabled.

- **RMode: Rounding Mode Control**

The encoding of this field is:

0b00: Round to Nearest (RN) mode

0b01: Round towards Plus Infinity (RP) mode.

0b10: Round towards Minus Infinity (RM) mode.

0b11: Round towards Zero (RZ) mode.

The specified rounding mode is used by almost all floating-point instructions.

- **IDC: Input Denormal Cumulative Exception**

IDC is a cumulative exception bit for floating-point exception; see also bits [4:0].

This bit is set to 1 to indicate that the corresponding exception has occurred since 0 was last written to it.

- **IXC: Inexact Cumulative Exception**

IXC is a cumulative exception bit for floating-point exception; see also bit [7].

This bit is set to 1 to indicate that the corresponding exception has occurred since 0 was last written to it.

- **UFC: Underflow Cumulative Exception**

UFC is a cumulative exception bit for floating-point exception; see also bit [7].

This bit is set to 1 to indicate that the corresponding exception has occurred since 0 was last written to it.

- **OFC: Overflow Cumulative Exception**

OFC is a cumulative exception bit for floating-point exception; see also bit [7].

This bit is set to 1 to indicate that the corresponding exception has occurred since 0 was last written to it.

- **DZC: Division by Zero Cumulative Exception**

DZC is a cumulative exception bit for floating-point exception; see also bit [7].

This bit is set to 1 to indicate that the corresponding exception has occurred since 0 was last written to it.

- **IOC: Invalid Operation Cumulative Exception**

IOC is a cumulative exception bit for floating-point exception; see also bit [7].

This bit is set to 1 to indicate that the corresponding exception has occurred since 0 was last written to it.

13.12.2.5 Floating-point Default Status Control Register

Name: FPDSCR

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	AHP	DN	FZ

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

The FPDSCR holds the default values for the floating-point status control data.

- **AHP: FPSCR.AHP Default Value**

Default value for FPSCR.AHP.

- **DN: FPSCR.DN Default Value**

Default value for FPSCR.DN.

- **FZ: FPSCR.FZ Default Value**

Default value for FPSCR.FZ.

- **RMode: FPSCR.RMode Default Value**

Default value for FPSCR.RMode.

13.13 Glossary

This glossary describes some of the terms used in technical documents from ARM.

Abort	A mechanism that indicates to a processor that the value associated with a memory access is invalid. An abort can be caused by the external or internal memory system as a result of attempting to access invalid instruction or data memory.
Aligned	A data item stored at an address that is divisible by the number of bytes that defines the data size is said to be aligned. Aligned words and halfwords have addresses that are divisible by four and two respectively. The terms word-aligned and halfword-aligned therefore stipulate addresses that are divisible by four and two respectively.
Banked register	A register that has multiple physical copies, where the state of the processor determines which copy is used. The Stack Pointer, SP (R13) is a banked register.
Base register	In instruction descriptions, a register specified by a load or store instruction that is used to hold the base value for the instruction's address calculation. Depending on the instruction and its addressing mode, an offset can be added to or subtracted from the base register value to form the address that is sent to memory. *See also* "Index register".
Big-endian (BE)	Byte ordering scheme in which bytes of decreasing significance in a data word are stored at increasing addresses in memory. *See also* "Byte-invariant", "Endianness", "Little-endian (LE)".
Big-endian memory	Memory in which: a byte or halfword at a word-aligned address is the most significant byte or halfword within the word at that address, a byte at a halfword-aligned address is the most significant byte within the halfword at that address. *See also* "Little-endian memory".
Breakpoint	A breakpoint is a mechanism provided by debuggers to identify an instruction at which program execution is to be halted. Breakpoints are inserted by the programmer to enable inspection of register contents, memory locations, variable values at fixed points in the program execution to test that the program is operating correctly. Breakpoints are removed after the program is successfully tested.
Byte-invariant	In a byte-invariant system, the address of each byte of memory remains unchanged when switching between little-endian and big-endian operation. When a data item larger than a byte is loaded from or stored to memory, the bytes making up that data item are arranged into the correct order depending on the endianness of the memory access. An ARM byte-invariant implementation also supports unaligned halfword and word memory accesses. It expects multi-word accesses to be word-aligned.
Cache	A block of on-chip or off-chip fast access memory locations, situated between the processor and main memory, used for storing and retrieving copies of often used instructions, data, or instructions and data. This is done to greatly increase the average speed of memory accesses and so improve processor performance.

Condition field	A four-bit field in an instruction that specifies a condition under which the instruction can execute.
Conditional execution	If the condition code flags indicate that the corresponding condition is true when the instruction starts executing, it executes normally. Otherwise, the instruction does nothing.
Context	The environment that each process operates in for a multitasking operating system. In ARM processors, this is limited to mean the physical address range that it can access in memory and the associated memory access permissions.
Coprocessor	A processor that supplements the main processor. Cortex-M4 does not support any coprocessors.
Debugger	A debugging system that includes a program, used to detect, locate, and correct software faults, together with custom hardware that supports software debugging.
Direct Memory Access (DMA)	An operation that accesses main memory directly, without the processor performing any accesses to the data concerned.
Doubleword	A 64-bit data item. The contents are taken as being an unsigned integer unless otherwise stated.
Doubleword-aligned	A data item having a memory address that is divisible by eight.
Endianness	Byte ordering. The scheme that determines the order that successive bytes of a data word are stored in memory. An aspect of the system's memory mapping. *See also* "Little-endian (LE)" and "Big-endian (BE)".
Exception	An event that interrupts program execution. When an exception occurs, the processor suspends the normal program flow and starts execution at the address indicated by the corresponding exception vector. The indicated address contains the first instruction of the handler for the exception. An exception can be an interrupt request, a fault, or a software-generated system exception. Faults include attempting an invalid memory access, attempting to execute an instruction in an invalid processor state, and attempting to execute an undefined instruction.
Exception service routine	*See* "Interrupt handler".
Exception vector	*See* "Interrupt vector".
Flat address mapping	A system of organizing memory in which each physical address in the memory space is the same as the corresponding virtual address.
Halfword	A 16-bit data item.
Illegal instruction	An instruction that is architecturally Undefined.

Implementation-defined	The behavior is not architecturally defined, but is defined and documented by individual implementations.
Implementation-specific	The behavior is not architecturally defined, and does not have to be documented by individual implementations. Used when there are a number of implementation options available and the option chosen does not affect software compatibility.
Index register	In some load and store instruction descriptions, the value of this register is used as an offset to be added to or subtracted from the base register value to form the address that is sent to memory. Some addressing modes optionally enable the index register value to be shifted prior to the addition or subtraction. *See also* "Base register".
Instruction cycle count	The number of cycles that an instruction occupies the Execute stage of the pipeline.
Interrupt handler	A program that control of the processor is passed to when an interrupt occurs.
Interrupt vector	One of a number of fixed addresses in low memory, or in high memory if high vectors are configured, that contains the first instruction of the corresponding interrupt handler.
Little-endian (LE)	Byte ordering scheme in which bytes of increasing significance in a data word are stored at increasing addresses in memory. *See also* "Big-endian (BE)", "Byte-invariant", "Endianness".
Little-endian memory	Memory in which: a byte or halfword at a word-aligned address is the least significant byte or halfword within the word at that address, a byte at a halfword-aligned address is the least significant byte within the halfword at that address. *See also* "Big-endian memory".
Load/store architecture	A processor architecture where data-processing operations only operate on register contents, not directly on memory contents.
Memory Protection Unit (MPU)	Hardware that controls access permissions to blocks of memory. An MPU does not perform any address translation.
Prefetching	In pipelined processors, the process of fetching instructions from memory to fill up the pipeline before the preceding instructions have finished executing. Prefetching an instruction does not mean that the instruction has to be executed.
Preserved	Preserved by writing the same value back that has been previously read from the same field on the same processor.

Read	Reads are defined as memory operations that have the semantics of a load. Reads include the Thumb instructions LDM, LDR, LDRSH, LDRH, LDRSB, LDRB, and POP.
Region	A partition of memory space.
Reserved	A field in a control register or instruction format is reserved if the field is to be defined by the implementation, or produces Unpredictable results if the contents of the field are not zero. These fields are reserved for use in future extensions of the architecture or are implementation-specific. All reserved bits not used by the implementation must be written as 0 and read as 0.
Thread-safe	In a multi-tasking environment, thread-safe functions use safeguard mechanisms when accessing shared resources, to ensure correct operation without the risk of shared access conflicts.
Thumb instruction	One or two halfwords that specify an operation for a processor to perform. Thumb instructions must be halfword-aligned.
Unaligned	A data item stored at an address that is not divisible by the number of bytes that defines the data size is said to be unaligned. For example, a word stored at an address that is not divisible by four.
Undefined	Indicates an instruction that generates an Undefined instruction exception.
Unpredictable	One cannot rely on the behavior. Unpredictable behavior must not represent security holes. Unpredictable behavior must not halt or hang the processor, or any parts of the system.
Warm reset	Also known as a core reset. Initializes the majority of the processor excluding the debug controller and debug logic. This type of reset is useful if debugging features of a processor.
WA	*See* “Write-allocate (WA)”.
WB	*See* “Write-back (WB)”.
Word	A 32-bit data item.
Write	Writes are defined as operations that have the semantics of a store. Writes include the Thumb instructions STM, STR, STRH, STRB, and PUSH.
Write-allocate (WA)	In a write-allocate cache, a cache miss on storing data causes a cache line to be allocated into the cache.

Write-back (WB)	In a write-back cache, data is only written to main memory when it is forced out of the cache on line replacement following a cache miss. Otherwise, writes by the processor only update the cache. This is also known as copyback.
Write buffer	A block of high-speed memory, arranged as a FIFO buffer, between the data cache and main memory, whose purpose is to optimize stores to main memory.
Write-through (WT)	In a write-through cache, data is written to main memory at the same time as the cache is updated.

14. Cortex-M Cache Controller (CMCC)

14.1 Description

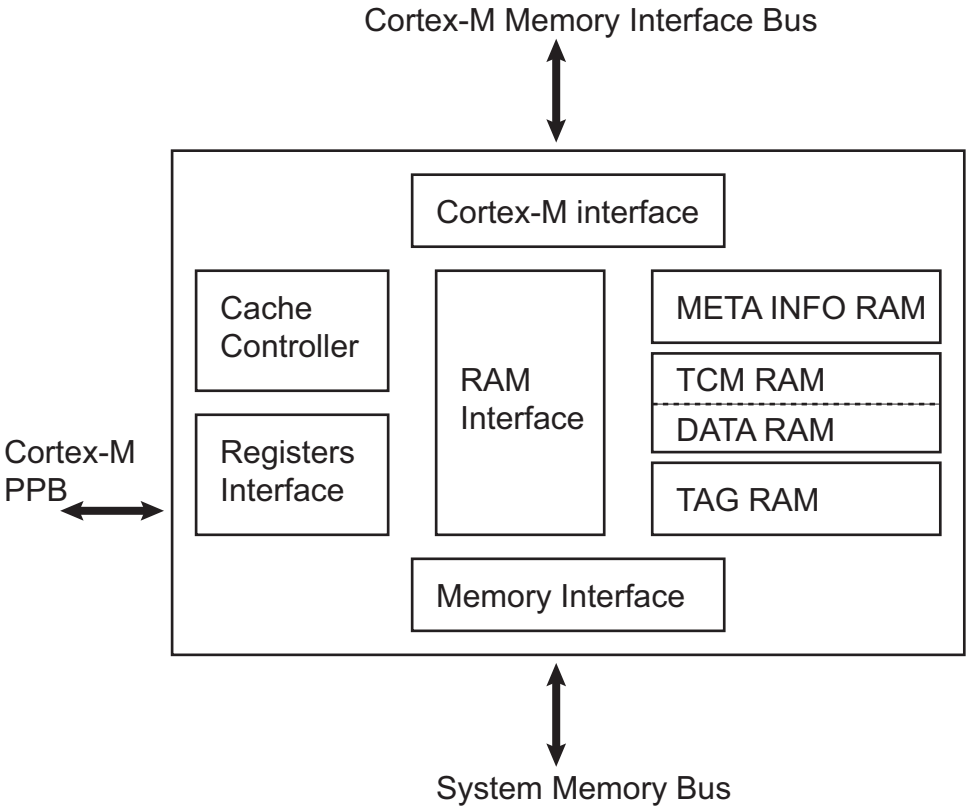
The Cortex-M Cache Controller (CMCC) is a 4-Way set associative unified cache controller. It integrates a controller, a tag directory, data memory, metadata memory and a configuration interface.

14.2 Embedded Characteristics

- Physically addressed and physically tagged
- CMCC memory size set to 8KB
- L1 data cache set up to 8 Kbytes
- L1 Tightly Coupled Memory RAM (TCM) up to 16KB
- L1 cache line size set to 16 Bytes
- L1 cache integrates 32 bus master interface
- Software-allocated RAM resource between cache and TCM
- Unified direct mapped cache architecture
- Unified 4-Way set associative cache architecture
- Write accesses forwarded, cache state not modified. Allocate on read.
- Round Robin victim selection policy
- Event Monitoring, with one programmable 32-bit counter
- Configuration registers accessible through Cortex-M Private Peripheral Bus (PPB)
- Cache interface includes cache maintenance operations registers

14.3 Block Diagram

Figure 14-1. Block Diagram



14.4 Functional Description

14.4.1 Cache Operation

On reset, the cache controller data entries are all invalidated and the cache is disabled. The cache is transparent to processor operations. The cache controller is activated with its configuration registers. The configuration interface is memory-mapped in the private peripheral bus.

Use the following sequence to enable the cache controller:

1. Verify that the cache controller is disabled by reading the value of the CSTS (Cache Controller Status) bit of the Status register (CMCC_SR).
2. Enable the cache controller by writing a one to the CEN (Cache Enable) bit of the Control register (CMCC_CTRL).

The cache controller integrates three memory areas, selectable through address decoding:

- A cacheable memory area—allows software execution with code located in slow memory (embedded Flash)
- A TCM area—provides fast and predictable code execution and data access
- A non-cacheable memory area—permits data and instruction access to system-level shared memory or peripheral

The total amount of RAM is shared between the cache controller and the TCM. If the application requires a large amount of TCM, the cache can be disabled and its memory reused. If the application requires only cache memory, the TCM is reallocated to the cache memory. The size of the RAM allocated to the cache is defined in the

PRGCSIZE field of the CMCC_CFG register. The difference between the cache size and the total CMCC RAM size is automatically allocated as TCM.

14.4.2 Cache Maintenance

If the contents seen by the cache have changed, the user must invalidate the cache entries. This can be done line-by-line or for all cache entries.

14.4.2.1 Cache Invalidate-by-Line Operation

When an invalidate-by-line command is issued, the cache controller resets the valid bit information of the decoded cache line. As the line is no longer valid, the replacement counter points to that line.

Use the following sequence to invalidate one line of cache:

1. Disable the cache controller by clearing the CEN bit of CMCC_CTRL.
2. Check the CSTS bit of CMCC_SR to verify that the cache is successfully disabled.
3. Perform an invalidate-by-line by configuring the bits INDEX and WAY in the Maintenance Register 1 (CMCC_MAINT1).
4. Enable the cache controller by writing a one the CEN bit of the CMCC_CTRL.

14.4.2.2 Cache Invalidate All Operation

To invalidate all cache entries, write a one to the INVALL bit of the Maintenance Register 0 (CMCC_MAINT0).

14.4.3 Cache Performance Monitoring

The Cortex-M cache controller includes a programmable 32-bit monitor counter. The monitor can be configured to count the number of clock cycles, the number of data hits or the number of instruction hits.

Use the following sequence to activate the counter:

1. Configure the monitor counter by writing to the MODE field of the Monitor Configuration register (CMCC_MCFG).
2. Enable the counter by writing a one to the MENABLE bit of the Monitor Enable register (CMCC_MEN).
3. If required, clear the counter by writing a one to the SWRST bit of the Monitor Control register (CMCC_MCTRL).
4. Check the value of the monitor counter by reading the EVENT_CNT field of the CMCC_MSR.

14.5 Cortex-M Cache Controller (CMCC) User Interface

Table 14-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Cache Controller Type Register	CMCC_TYPE	Read-only	0X000013D7
0x04	Cache Controller Configuration Register	CMCC_CFG	Read/Write	0x00000020
0x08	Cache Controller Control Register	CMCC_CTRL	Write-only	–
0x0C	Cache Controller Status Register	CMCC_SR	Read-only	0X00000000
0x10–0x1C	Reserved	–	–	–
0x20	Cache Controller Maintenance Register 0	CMCC_MAINT0	Write-only	–
0x24	Cache Controller Maintenance Register 1	CMCC_MAINT1	Write-only	–
0x28	Cache Controller Monitor Configuration Register	CMCC_MCFG	Read/Write	0x00000000
0x2C	Cache Controller Monitor Enable Register	CMCC_MEN	Read/Write	0x00000000
0x30	Cache Controller Monitor Control Register	CMCC_MCTRL	Write-only	–
0x34	Cache Controller Monitor Status Register	CMCC_MSR	Read-only	0x00000000
0x38–0xFC	Reserved	–	–	–

14.5.1 Cache Controller Type Register

Name: CMCC_TYPE

Address: 0x4003C000

Access: Read-only

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

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

15	14	13	12	11	10	9	8
–		CLSIZE			CSIZE		

7	6	5	4	3	2	1	0
LCKDOWN	WAYNUM		RRP	LRUP	RANDP	GCLK	AP

- **AP: Access Port Access Allowed**

0: Access Port Access is disabled.

1: Access Port Access is enabled.

- **GCLK: Dynamic Clock Gating Supported**

0: Cache controller does not support clock gating.

1: Cache controller uses dynamic clock gating.

- **RANDP: Random Selection Policy Supported**

0: Random victim selection is not supported.

1: Random victim selection is supported.

- **LRUP: Least Recently Used Policy Supported**

0: Least Recently Used Policy is not supported.

1: Least Recently Used Policy is supported.

- **RRP: Random Selection Policy Supported**

0: Random Selection Policy is not supported.

1: Random Selection Policy is supported.

- **WAYNUM: Number of Ways**

Value	Name	Description
0	DMAPPED	Direct Mapped Cache
1	ARCH2WAY	2-way set associative
2	ARCH4WAY	4-way set associative
3	ARCH8WAY	8-way set associative

- **LCKDOWN: Lockdown Supported**

0: Lockdown is not supported.

1: Lockdown is supported.

- **CSIZE: Data Cache Size**

Value	Name	Description
0	CSIZE_1KB	Data cache size is 1 Kbyte
1	CSIZE_2KB	Data cache size is 2 Kbytes
2	CSIZE_4KB	Data cache size is 4 Kbytes
3	CSIZE_8KB	Data cache size is 8 Kbytes

- **CLSIZE: Cache LIne Size**

Value	Name	Description
0	CLSIZE_1KB	Cache line size is 4 bytes
1	CLSIZE_2KB	Cache line size is 8 bytes
2	CLSIZE_4KB	Cache line size is 16 bytes
3	CLSIZE_8KB	Cache line size is 32 bytes

14.5.2 Cache Controller Configuration Register

Name: CMCC_CFG

Address: 0x4003C004

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
–	PRGCSIZE			–	DCDIS	ICDIS	GCLKDIS

- **GCLKDIS: Disable Clock Gating**

0: Clock gating is activated.

1: Clock gating is disabled.

- **ICDIS: Instruction Caching Disable**

0: Instruction caching enabled.

1: Instruction caching disabled.

- **DCDIS: Data Caching Disable**

0: Data caching enabled.

1: Data caching disabled.

- **PRGCSIZE: Programmable Cache Size**

Value	Name	Description
0	-	Reserved
1	PRGCSIZE_2KB	Programmable cache size is 2 Kbytes
2	PRGCSIZE_4KB	Programmable cache size is 4 Kbytes (default value)
3	PRGCSIZE_8KB	Programmable cache size is 8 Kbytes

14.5.3 Cache Controller Control Register

Name: CMCC_CTRL

Address: 0x4003C008

Access: Write-only

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	CEN

- **CEN: Cache Controller Enable**

0: The cache controller is disabled.

1: The cache controller is enabled.

14.5.4 Cache Controller Status Register

Name: CMCC_SR

Address: 0x4003C00C

Access: Read-only

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

- **CSTS: Cache Controller Status**

0: The cache controller is disabled.

1: The cache controller is enabled.

14.5.5 Cache Controller Maintenance Register 0

Name: CMCC_MAINT0

Address: 0x4003C020

Access: Write-only

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

- **INVALL: Cache Controller Invalidate All**

0: No effect.

1: All cache entries are invalidated.

14.5.6 Cache Controller Maintenance Register 1

Name: CMCC_MAINT1

Address: 0x4003C024

Access: Write-only

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

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

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

7	6	5	4	3	2	1	0
INDEX				–	–	–	–

- **INDEX: Invalidate Index**

This field indicates the cache line that is being invalidated.

The size of the INDEX field depends on the cache size:

For example:

– for 2 Kbytes: 5 bits

– for 4 Kbytes: 6 bits

– for 8 Kbytes: 7 bits

- **WAY: Invalidate Way**

Value	Name	Description
0	WAY0	Way 0 is selection for index invalidation
1	WAY1	Way 1 is selection for index invalidation
2	WAY2	Way 2 is selection for index invalidation
3	WAY3	Way 3 is selection for index invalidation

14.5.7 Cache Controller Monitor Configuration Register

Name: CMCC_MCFG

Address: 0x4003C028

Access: Read/Write

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

- **MODE: Cache Controller Monitor Counter Mode**

Value	Name	Description
0	CYCLE_COUNT	Cycle counter
1	IHIT_COUNT	Instruction hit counter
2	DHIT_COUNT	Data hit counter

14.5.8 Cache Controller Monitor Enable Register

Name: CMCC_MEN

Address: 0x4003C02C

Access: Read/Write

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	MENABLE

- **MENABLE: Cache Controller Monitor Enable**

0: The monitor counter is disabled.

1: The monitor counter is enabled.

14.5.9 Cache Controller Monitor Control Register

Name: CMCC_MCTRL

Address: 0x4003C030

Access: Write-only

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

- **SWRST: Monitor**

0: No effect.

1: Resets the event counter register.

14.5.10 Cache Controller Monitor Status Register

Name: CMCC_MSR

Address: 0x4003C034

Access: Read-only

31	30	29	28	27	26	25	24
EVENT_CNT							

23	22	21	20	19	18	17	16
EVENT_CNT							

15	14	13	12	11	10	9	8
EVENT_CNT							

7	6	5	4	3	2	1	0
EVENT_CNT							

- **EVENT_CNT: Monitor Event Counter**

15. Bus Matrix (MATRIX)

15.1 Description

The Bus Matrix (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.

15.2 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
- Automatic clock-off mode for power reduction
- Register Write Protection

15.3 Master/Slave Management

15.3.1 Matrix Masters

The Bus Matrix manages five 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 15-1. List of Bus Matrix Masters

Master No.	Name
0	Processor Instruction/Data Bus
1	Processor System Bus
2	Peripheral DMA Controller (PDC)
3	CRC Calculation Unit (CRCCU)
4	USB Host DMA

15.3.2 Matrix Slaves

The Bus Matrix manages five slaves. Each slave has its own arbiter, providing a different arbitration per slave

Table 15-2. List of Bus Matrix Slaves

Slave No.	Name
0	Internal SRAM
1	Internal ROM
2	Internal Flash
3	Peripheral Bridge
4	USB Host Register

15.3.3 Master to Slave Access

Table 15-3 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 15-3. Master to Slave Access

		Masters				
		0	1	2	3	4
	Slaves	Processor I/D Bus	Processor System Bus	PDC	CRCCU	USB Host DMA
0	Internal SRAM	–	X	X	X	X
1	Internal ROM	X	–	X	–	–
2	Internal Flash	X	–	X	X	–
3	Peripheral Bridge	–	X	X	–	–
4	USB Host Register	–	X	–	–	–

15.4 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.

15.5 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

15.5.1 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.

15.5.2 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.

15.5.3 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 Section 15.9 “Bus Matrix (MATRIX) User Interface”.

15.6 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:

- Round-robin arbitration (default)
- Fixed priority arbitration

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 15.6.1 “Arbitration Rules”.

15.6.1 Arbitration Rules

Each arbiter has the ability to arbitrate between requests from two or more masters. To avoid burst breaking and to provide the maximum throughput for slave interfaces, arbitration should take place during the following cycles:

- Idle cycles: When a slave is not connected to any master or is connected to a master which is not currently accessing it.
- Single cycles: When a slave is currently doing a single access.
- 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 15.6.1.1 “Undefined Length Burst Arbitration”.
- 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 15.6.1.2 “Slot Cycle Limit Arbitration”.

15.6.1.1 Undefined Length Burst Arbitration

In order to prevent long AHB burst lengths that can lock the access to the slave for an excessive period of time, the user can trigger the re-arbitration before the end of the incremental bursts. The re-arbitration period can be selected from the following Undefined Length Burst Type (ULBT) possibilities:

- Unlimited: no predetermined end of burst is generated. This value enables 1 Kbyte burst lengths.
- 4-beat bursts: predetermined end of burst is generated at the end of each 4-beat boundary during INCR transfer.
- 8-beat bursts: predetermined end of burst is generated at the end of each 8-beat boundary during INCR transfer.
- 16-beat bursts: predetermined end of burst is generated at the end of each 16-beat boundary during INCR transfer.

This selection is made through the ULBT field of the Master Configuration registers (MATRIX_MCFG).

15.6.1.2 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.

15.6.1.3 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.

15.6.1.4 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 requests from two or more masters are active at the same time, the master with the highest priority number is serviced first. If requests from two or more master 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).

15.7 System Configuration

The System I/O Configuration register (CCFG_SYSIO) configures I/O lines in system I/O mode (such as JTAG, ERASE, USB I/Os, 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.

The USB Management register (CCFG_USBMR) configures the USB transceiver and selects the USB mode (Host or Device).

The I2S Clock Source Selection register (CCFG_I2SCLKSEL) configures the I2S peripheral to provide a clock source independent of the processor clock.

The Dynamic Clock Gating register (CCFG_DYNCKG) optimizes the power consumption for specific applications. When enabled, the system bus circuitry is only driven by the clock when necessary (transfer in progress, access to peripheral, etc.).

15.8 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 "Bus Matrix Write Protection Mode Register" (MATRIX_WPMR).

If a write access to a write-protected register is detected, the WPVS flag in the "Bus Matrix Write Protection Status Register" (MATRIX_WPSR) is set and the field WPVSRC indicates the register in which the write access has been attempted.

The WPVS flag is reset by writing the MATRIX_WPMR with the appropriate access key WPKEY.

The following registers can be write-protected:

- "Bus Matrix Master Configuration Registers"
- "Bus Matrix Slave Configuration Registers"
- "Bus Matrix Priority Registers A For Slaves"
- "System I/O Configuration Register"

15.9 Bus Matrix (MATRIX) User Interface

Table 15-4. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	Master Configuration Register 0	MATRIX_MCFG0	Read/Write	0x00000000
0x0004	Master Configuration Register 1	MATRIX_MCFG1	Read/Write	0x00000000
0x0008	Master Configuration Register 2	MATRIX_MCFG2	Read/Write	0x00000000
0x000C	Master Configuration Register 3	MATRIX_MCFG3	Read/Write	0x00000000
0x0010	Master Configuration Register 4	MATRIX_MCFG4	Read/Write	0x00000000
0x0014–0x003C	Reserved	–	–	–
0x0040	Slave Configuration Register 0	MATRIX_SCFG0	Read/Write	0x00010010
0x0044	Slave Configuration Register 1	MATRIX_SCFG1	Read/Write	0x00050010
0x0048	Slave Configuration Register 2	MATRIX_SCFG2	Read/Write	0x00000010
0x004C	Slave Configuration Register 3	MATRIX_SCFG3	Read/Write	0x00000010
0x0050	Slave Configuration Register 4	MATRIX_SCFG4	Read/Write	0x00000010
0x0054–0x007C	Reserved	–	–	–
0x0080	Priority Register A for Slave 0	MATRIX_PRAS0	Read/Write	0x00000000
0x0084	Reserved	–	–	–
0x0088	Priority Register A for Slave 1	MATRIX_PRAS1	Read/Write	0x00000000
0x008C	Reserved	–	–	–
0x0090	Priority Register A for Slave 2	MATRIX_PRAS2	Read/Write	0x00000000
0x0094	Reserved	–	–	–
0x0098	Priority Register A for Slave 3	MATRIX_PRAS3	Read/Write	0x00000000
0x009C	Reserved	–	–	–
0x00A0	Priority Register A for Slave 4	MATRIX_PRAS4	Read/Write	0x00000000
0x00A4–0x010C	Reserved	–	–	–
0x0110	Reserved	–	–	0x22222224[(1)]
0x0114	System I/O Configuration Register	CCFG_SYSIO	Read/Write	0x0000_0C00
0x0118	Dynamic Clock Gating Register	CCFG_DYNCKG	Read/Write	0x00000007
0x011C	I2S Clock Source Selection Register	CCFG_I2SCLKSEL	Read/Write	0x00000000
0x0120	USB Management Register	CCFG_USBMR	Read/Write	0x00000000
0x0124–0x01E0	Reserved	–	–	–
0x1E4	Write Protection Mode Register	MATRIX_WPMR	Read/Write	0x0
0x1E8	Write Protection Status Register	MATRIX_WPSR	Read-only	0x0
0x01EC–0x01FC	Reserved	–	–	–

Note: 1. This default reset value must not be modified.

15.9.1 Bus Matrix Master Configuration Registers

Name: MATRIX_MCFGx [x = 0..4]

Address: 0x400E0200

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the Bus Matrix Write Protection Mode Register.

- **ULBT: Undefined Length Burst Type**

Value	Name	Description
0	UNLIMITED	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	4_BEAT	The undefined length burst is split into a 4-beat bursts allowing rearbitration at each 4-beat burst end.
3	8_BEAT	The undefined length burst is split into 8-beat bursts allowing rearbitration at each 8-beat burst end.
4	16_BEAT	The undefined length burst is split into 16-beat bursts allowing rearbitration at each 16-beat burst end.

15.9.2 Bus Matrix Slave Configuration Registers

Name: MATRIX_SCFGx [x = 0..4]

Address: 0x400E0240

Access: Read/Write

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
SLOT_CYCLE							

This register can only be written if the WPEN bit is cleared in the Bus Matrix Write Protection Mode Register.

- **SLOT_CYCLE: Maximum Number of Allowed Cycles for a Burst**

When SLOT_CYCLE AHB clock cycles have elapsed since the last arbitration, a new arbitration takes place to let another master access this slave. If another master is requesting the slave bus, then the current master burst is broken.

If SLOT_CYCLE = 0, the slot cycle limit feature is disabled and bursts always complete unless broken according to the ULBT.

This limit has been placed in order to enforce arbitration so as to meet potential latency constraints of masters waiting for slave access.

This limit must not be too small. Unreasonably small values break every burst and the Bus Matrix arbitrates without performing any data transfer. The default maximum value is usually an optimal conservative choice.

In most cases, this feature is not needed and should be disabled for power saving.

See Section 15.6.1.2 “Slot Cycle Limit Arbitration” for details.

- **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.

15.9.3 Bus Matrix Priority Registers A For Slaves

Name: MATRIX_PRAS0..MATRIX_PRAS4

Address: 0x400E0280 [0], 0x400E0288 [1], 0x400E0290 [2], 0x400E0298 [3], 0x400E02A0 [4]

Access: Read/Write

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

23	22	21	20	19	18	17	16
–	–	–	–	–	–	M4PR	

15	14	13	12	11	10	9	8
–	–	M3PR		–	–	M2PR	

7	6	5	4	3	2	1	0
–	–	M1PR		–	–	M0PR	

This register can only be written if the WPEN bit is cleared in the Bus Matrix Write Protection Mode Register.

- **MxPR: Master x Priority**

Fixed priority of master x to access the selected slave. The higher the number, the higher the priority.

15.9.4 System I/O Configuration Register

Name: CCFG_SYSIO

Address: 0x400E0314

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
–	–	–	SYSIO12	SYSIO11	SYSIO10	–	–
7	6	5	4	3	2	1	0
SYSIO7	SYSIO6	SYSIO5	SYSIO4	–	–	–	–

This register can only be written if the WPEN bit is cleared in the Bus Matrix Write Protection Mode Register.

- **SYSIO4: PB4 or TDI Assignment**

0: TDI function selected.

1: PB4 function selected.

- **SYSIO5: PB5 or TDO/TRACESWO Assignment**

0: TDO/TRACESWO function selected.

1: PB5 function selected.

- **SYSIO6: PB6 or TMS/SWDIO Assignment**

0: TMS/SWDIO function selected.

1: PB6 function selected.

- **SYSIO7: PB7 or TCK/SWCLK Assignment**

0: TCK/SWCLK function selected.

1: PB7 function selected.

- **SYSIO10: PA21 or DM Assignment**

0: DM function selected.

1: PA21 function selected.

- **SYSIO11: PA22 or DP Assignment**

0: DP function selected.

1: PA22 function selected.

- **SYSIO12: PB12 or ERASE Assignment**

0: ERASE function selected.

1: PB12 function selected.

15.9.5 Dynamic Clock Gating Register

Name: CCFG_DYNCKG

Address: 0x400E0318

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
–	–	–	–	–	EFCCKG	BRIDCKG	MATCKG

Note: Clearing all the bits provides the optimal energy reduction for the system bus circuitry.

- **MATCKG: MATRIX Dynamic Clock Gating**

0: MATRIX Dynamic Clock Gating Enabled. The MATRIX circuitry is driven by the clock only when a transfer to a peripheral is being performed. The energy consumption is optimized.

1: MATRIX Dynamic Clock Gating Disabled. The MATRIX circuitry is always driven by the clock in Active mode.

- **BRIDCKG: Bridge Dynamic Clock Gating Enable**

0: Bridge Dynamic Clock Gating Enabled. The Peripheral Bridge circuitry is driven by the clock only when a transfer to/from any peripheral located on the APB bus is being performed. The energy consumption is optimized.

1: Bridge Dynamic Clock Gating Disabled. The Peripheral Bridge circuitry is always driven by the clock in Active mode.

- **EFCCKG: EFC Dynamic Clock Gating Enable**

0: EFC Dynamic Clock Gating Enabled. The Embedded Flash Controller circuitry is driven by the clock only when an access to the Flash memory is being performed. The energy consumption is optimized.

1: EFC Dynamic Clock Gating Disabled. The Embedded Flash Controller is always driven by the clock in Active mode.

15.9.6 I2S Clock Source Selection Register

Name: CCFG_I2SCLKSEL

Address: 0x400E031C

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
–	–	–	–	–	–	CLKSEL1	CLKSEL0

- **CLKSEL0: I2S0 Clock Source**

0: Peripheral Clock selected (same frequency as processor clock)

1: PMC PCK4 selected (independent of processor clock)

- **CLKSEL1: I2S1 Clock Source**

0: Peripheral Clock selected (same frequency as processor clock)

1: PMC PCK4 selected (independent of processor clock)

15.9.7 USB Management Register

Name: CCFG_USBMR

Address: 0x400E0320

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
–			–	–	USBHTSC	USBHTSSC	USBMODE

- **USBMODE: USB Mode Selection**

The USB transceiver is shared by the USB Host and the USB device. The selection is done through the USBMODE bit.

0: USB Host selected

1: USB Device selected

- **USBHTSSC: USB Transceiver Suspend Software Control**

0: USB Host Transceiver active

1: USB Host Transceiver in Suspend mode

- **USBHTSC: USB Host Transceiver Suspend Control**

0: USB Host transceiver is controlled by the USB Host (OHCI)

1: USB Host transceiver is controlled by the USBHTSSC bit

15.9.8 Bus Matrix Write Protection Mode Register

Name: MATRIX_WPMR

Address: 0x400E03E4

Access: Read/Write

31	30	29	28	27	26	25	24
WPKEY							

23	22	21	20	19	18	17	16
WPKEY							

15	14	13	12	11	10	9	8
WPKEY							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WPEN

- **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 15.8 “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.

15.9.9 Bus Matrix Write Protection Status Register

Name: MATRIX_WPSR

Address: 0x400E03E8

Access: Read-only

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

23	22	21	20	19	18	17	16
WPVSRC							

15	14	13	12	11	10	9	8
WPVSRC							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WPVS

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last write of the MATRIX_WPMR.

1: A write protection violation has occurred since the last write of the MATRIX_WPMR. 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.

16. Parallel Input/Output Controller (PIO)

16.1 Description

The Parallel Input/Output Controller (PIO) manages up to 32 fully programmable input/output lines. Each I/O line may be dedicated as a general-purpose I/O or be assigned to a function of an embedded peripheral. This ensures effective optimization of the pins of the product.

Each I/O line is associated with a bit number in all of the 32-bit registers of the 32-bit wide user interface.

Each I/O line of the PIO Controller features:

- An input change interrupt enabling level change detection on any I/O line.
- Additional Interrupt modes enabling rising edge, falling edge, low-level or high-level detection on any I/O line.
- A glitch filter providing rejection of glitches lower than one-half of peripheral clock cycle.
- A debouncing filter providing rejection of unwanted pulses from key or push button operations.
- Multi-drive capability similar to an open drain I/O line.
- Control of the pull-up and pull-down of the I/O line.
- Input visibility and output control.

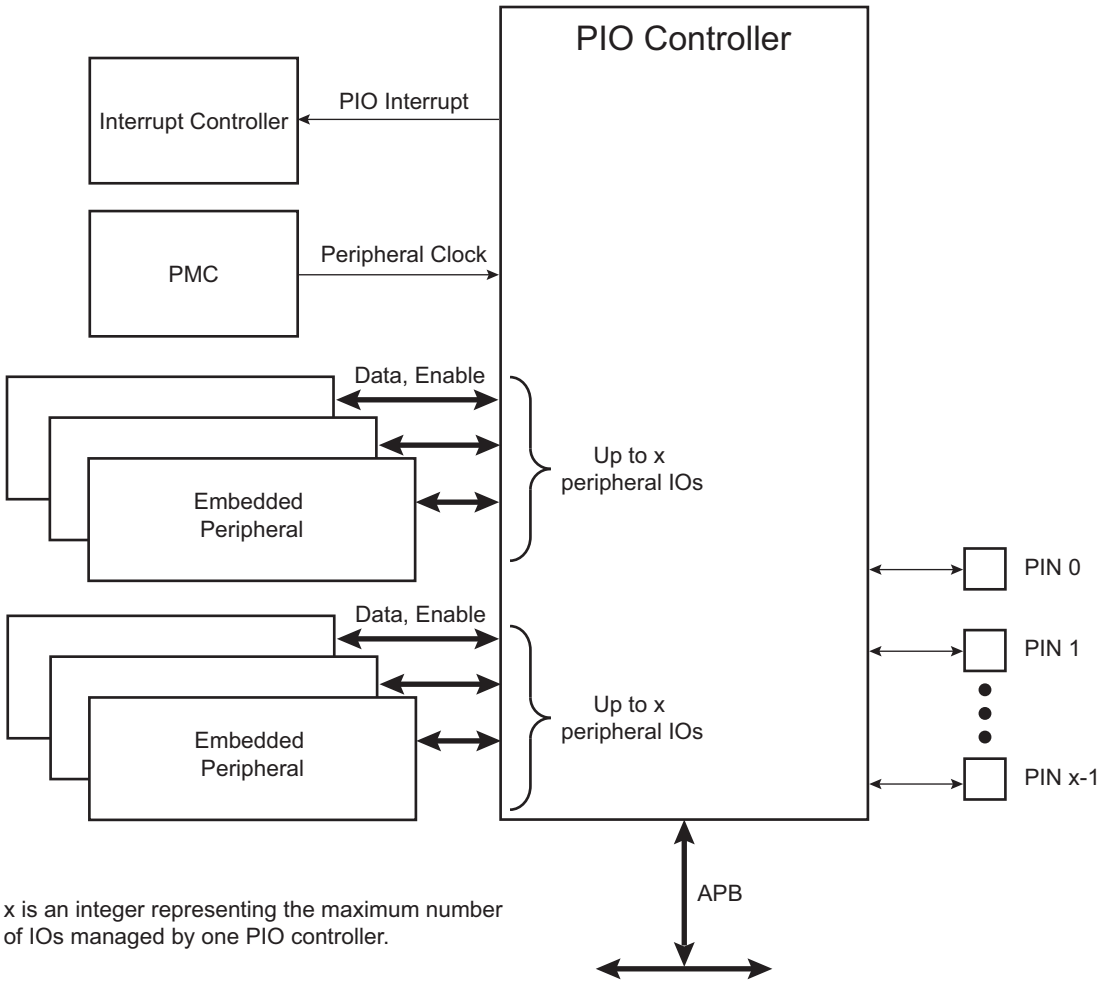
The PIO Controller also features a synchronous output providing up to 32 bits of data output in a single write operation.

16.2 Embedded Characteristics

- Up to 32 Programmable I/O Lines
- Fully Programmable through Set/Clear Registers
- Multiplexing of Four Peripheral Functions per I/O Line
- For each I/O Line (Whether Assigned to a Peripheral or Used as General Purpose I/O)
  - Input Change Interrupt
  - Programmable Glitch Filter
  - Programmable Debouncing Filter
  - Multi-drive Option Enables Driving in Open Drain
  - Programmable Pull-Up on Each I/O Line
  - Pin Data Status Register, Supplies Visibility of the Level on the Pin at Any Time
  - Additional Interrupt Modes on a Programmable Event: Rising Edge, Falling Edge, Low-Level or High-Level
- Synchronous Output, Provides Set and Clear of Several I/O Lines in a Single Write
- Register Write Protection
- Programmable Schmitt Trigger Inputs
- Programmable I/O Drive

16.3 Block Diagram

Figure 16-1. Block Diagram



16.4 Product Dependencies

16.4.1 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.

16.4.2 External Interrupt Lines

When the WKUPx input pins must be used as external interrupt lines, the PIO Controller must be configured to disable the peripheral control on these IOs, and the corresponding IO lines must be set to Input mode.

16.4.3 Power Management

The Power Management Controller controls the peripheral clock in order to save power. Writing any of the registers of the user interface does not require the peripheral clock to be enabled. This means that the configuration of the I/O lines does not require the peripheral clock to be enabled.

However, when the clock is disabled, not all of the features of the PIO Controller are available, including glitch filtering. Note that the input change interrupt, the interrupt modes on a programmable event and the read of the pin level require the clock to be validated.

After a hardware reset, the peripheral clock is disabled by default.

The user must configure the Power Management Controller before any access to the input line information.

16.4.4 Interrupt Sources

For interrupt handling, the PIO Controllers are considered as user peripherals. This means that the PIO Controller interrupt lines are connected among the interrupt sources. Refer to the PIO Controller peripheral identifier in the Peripheral Identifiers table to identify the interrupt sources dedicated to the PIO Controllers. Using the PIO Controller requires the Interrupt Controller to be programmed first.

The PIO Controller interrupt can be generated only if the peripheral clock is enabled.

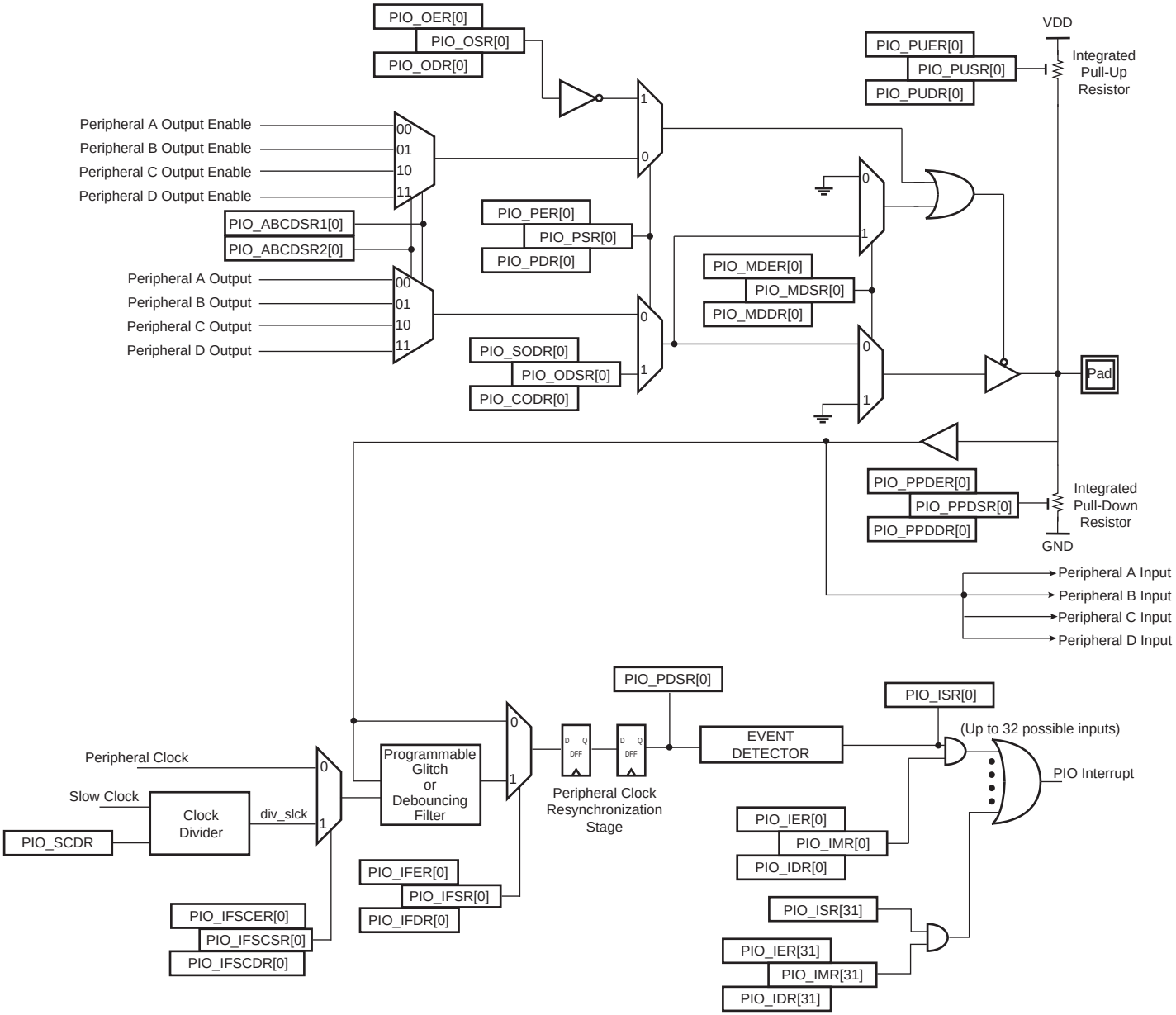
Table 16-1. Peripheral IDs

Instance	ID
PIOA	11
PIOB	12

16.5 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 16-2. In this description each signal shown represents one of up to 32 possible indexes.

Figure 16-2. I/O Line Control Logic



16.5.1 Pull-up and Pull-down Resistor Control

Each I/O line is designed with an embedded pull-up resistor and an embedded pull-down resistor. The pull-up resistor can be enabled or disabled by writing to the Pull-up Enable Register (PIO_PUER) or Pull-up Disable Register (PIO_PUDR), respectively. Writing to these registers results in setting or clearing the corresponding bit in the Pull-up Status Register (PIO_PUSR). Reading a one in PIO_PUSR means the pull-up is disabled and reading a zero means the pull-up is enabled. The pull-down resistor can be enabled or disabled by writing the Pull-down Enable Register (PIO_PPDER) or the Pull-down Disable Register (PIO_PPDDR), respectively. Writing in these

registers results in setting or clearing the corresponding bit in the Pull-down Status Register (PIO_PPDSR). Reading a one in PIO_PPDSR means the pull-up is disabled and reading a zero means the pull-down is enabled.

Enabling the pull-down resistor while the pull-up resistor is still enabled is not possible. In this case, the write of PIO_PPDER for the relevant I/O line is discarded. Likewise, enabling the pull-up resistor while the pull-down resistor is still enabled is not possible. In this case, the write of PIO_PUER for the relevant I/O line is discarded.

Control of the pull-up resistor is possible regardless of the configuration of the I/O line.

After reset, depending on the I/O, pull-up or pull-down can be set.

16.5.2 I/O Line or Peripheral Function Selection

When a pin is multiplexed with one or two peripheral functions, the selection is controlled with the Enable Register (PIO_PER) and the Disable Register (PIO_PDR). The Status Register (PIO_PSR) is the result of the set and clear registers and indicates whether the pin is controlled by the corresponding peripheral or by the PIO Controller. A value of zero indicates that the pin is controlled by the corresponding on-chip peripheral selected in the Peripheral ABCD Select registers (PIO_ABCDSR1 and PIO_ABCDSR2). A value of one indicates the pin is controlled by the PIO Controller.

If a pin is used as a general-purpose I/O line (not multiplexed with an on-chip peripheral), PIO_PER and PIO_PDR have no effect and PIO_PSR returns a one for the corresponding bit.

After reset, the I/O lines are controlled by the PIO Controller, i.e., PIO_PSR resets at one. However, in some events, it is important that PIO lines are controlled by the peripheral (as in the case of memory chip select lines that must be driven inactive after reset, or for address lines that must be driven low for booting out of an external memory). Thus, the reset value of PIO_PSR is defined at the product level and depends on the multiplexing of the device.

16.5.3 Peripheral A or B or C or D Selection

The PIO Controller provides multiplexing of up to four peripheral functions on a single pin. The selection is performed by writing PIO_ABCDSR1 and PIO_ABCDSR2.

For each pin:

- The corresponding bit at level zero in PIO_ABCDSR1 and the corresponding bit at level zero in PIO_ABCDSR2 means peripheral A is selected.
- The corresponding bit at level one in PIO_ABCDSR1 and the corresponding bit at level zero in PIO_ABCDSR2 means peripheral B is selected.
- The corresponding bit at level zero in PIO_ABCDSR1 and the corresponding bit at level one in PIO_ABCDSR2 means peripheral C is selected.
- The corresponding bit at level one in PIO_ABCDSR1 and the corresponding bit at level one in PIO_ABCDSR2 means peripheral D is selected.

Note that multiplexing of peripheral lines A, B, C and D only affects the output line. The peripheral input lines are always connected to the pin input (see Figure 16-2).

Writing in PIO_ABCDSR1 and PIO_ABCDSR2 manages the multiplexing regardless of the configuration of the pin. However, assignment of a pin to a peripheral function requires a write in PIO_ABCDSR1 and PIO_ABCDSR2 in addition to a write in PIO_PDR.

After reset, PIO_ABCDSR1 and PIO_ABCDSR2 are zero, thus indicating that all the PIO lines are configured on peripheral A. However, peripheral A generally does not drive the pin as the PIO Controller resets in I/O line mode.

If the software selects a peripheral A, B, C or D which does not exist for a pin, no alternate functions are enabled for this pin and the selection is taken into account. The PIO Controller does not carry out checks to prevent selection of a peripheral which does not exist.

16.5.4 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.

16.5.5 Synchronous Data Output

Clearing one or more PIO line(s) and setting another one or more PIO line(s) synchronously cannot be done by using PIO_SODR and PIO_CODR. It requires two successive write operations into two different registers. To overcome this, the PIO Controller offers a direct control of PIO outputs by single write access to PIO_ODSR. Only bits unmasked by the Output Write Status Register (PIO_OWSR) are written. The mask bits in PIO_OWSR are set by writing to the Output Write Enable Register (PIO_OWER) and cleared by writing to the Output Write Disable Register (PIO_OWDR).

After reset, the synchronous data output is disabled on all the I/O lines as PIO_OWSR resets at 0x0.

16.5.6 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.

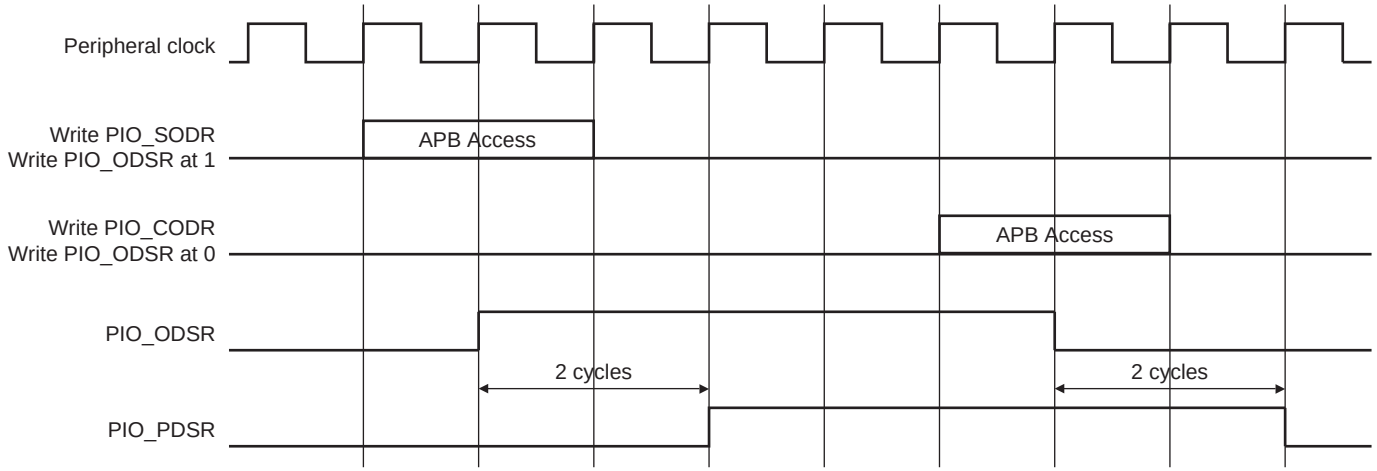
The multi-drive feature is controlled by the Multi-driver Enable Register (PIO_MDER) and the Multi-driver Disable Register (PIO_MDDR). The multi-drive can be selected whether the I/O line is controlled by the PIO Controller or assigned to a peripheral function. The Multi-driver Status Register (PIO_MDSR) 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.

16.5.7 Output Line Timings

Figure 16-3 shows how the outputs are driven either by writing PIO_SODR or PIO_CODR, or by directly writing PIO_ODSR. This last case is valid only if the corresponding bit in PIO_OWSR is set. Figure 16-3 also shows when the feedback in the Pin Data Status Register (PIO_PDSR) is available.

Figure 16-3. Output Line Timings



16.5.8 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.

16.5.9 Input Glitch and Debouncing Filters

Optional input glitch and debouncing filters are independently programmable on each I/O line.

The glitch filter can filter a glitch with a duration of less than 1/2 peripheral clock and the debouncing filter can filter a pulse of less than 1/2 period of a programmable divided slow clock.

The selection between glitch filtering or debounce filtering is done by writing in the PIO Input Filter Slow Clock Disable Register (PIO_IFSCDR) and the PIO Input Filter Slow Clock Enable Register (PIO_IFSCER). Writing PIO_IFSCDR and PIO_IFSCER, respectively, sets and clears bits in the Input Filter Slow Clock Status Register (PIO_IFSCSR).

The current selection status can be checked by reading the PIO_IFSCSR.

- If PIO_IFSCSR[i] = 0: The glitch filter can filter a glitch with a duration of less than 1/2 master clock period.
- If PIO_IFSCSR[i] = 1: The debouncing filter can filter a pulse with a duration of less than 1/2 programmable divided slow clock period.

For the debouncing filter, the period of the divided slow clock is defined by writing in the DIV field of the Slow Clock Divider Debouncing Register (PIO_SCDR):

$t_{div\_slck} = ((DIV + 1) \times 2) \times t_{slck}$

When the glitch or debouncing filter is enabled, a glitch or pulse with a duration of less than 1/2 selected clock cycle (selected clock represents peripheral clock or divided slow clock depending on PIO_IFSCDR and PIO_IFSCER programming) is automatically rejected, while a pulse with a duration of one selected clock (peripheral clock or divided slow clock) cycle or more is accepted. For pulse durations between 1/2 selected clock cycle and one selected clock cycle, the pulse may or may not be taken into account, depending on the precise timing of its occurrence. Thus for a pulse to be visible, it must exceed one selected clock cycle, whereas for a glitch to be reliably filtered out, its duration must not exceed 1/2 selected clock cycle.

The filters also introduce some latencies, illustrated in Figure 16-4 and Figure 16-5.

The glitch filters are controlled by the Input Filter Enable Register (PIO_IFER), the Input Filter Disable Register (PIO_IFDR) and the Input Filter Status Register (PIO_IFSR). Writing PIO_IFER and PIO_IFDR respectively sets and clears bits in PIO_IFSR. This last register enables the glitch filter on the I/O lines.

When the glitch and/or debouncing filter is enabled, it does not modify the behavior of the inputs on the peripherals. It acts only on the value read in PIO_PDSR and on the input change interrupt detection. The glitch and debouncing filters require that the peripheral clock is enabled.

Figure 16-4. Input Glitch Filter Timing

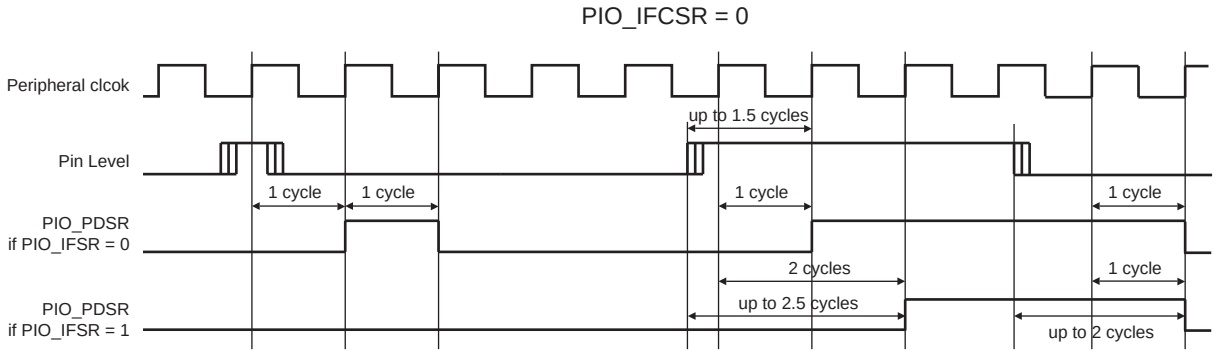
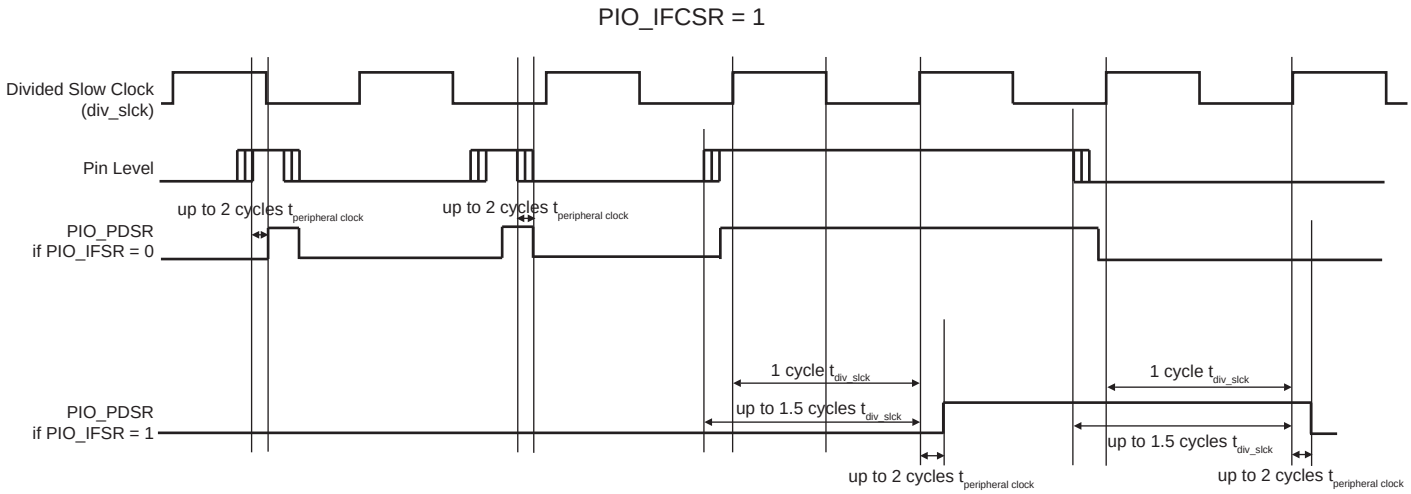


Figure 16-5. Input Debouncing Filter Timing



16.5.10 Input Edge/Level Interrupt

The PIO Controller can be programmed to generate an interrupt when it detects an edge or a level on an I/O line. The Input Edge/Level interrupt is controlled by writing the Interrupt Enable Register (PIO_IER) and the Interrupt Disable Register (PIO_IDR), which enable and disable the input change interrupt respectively by setting and clearing the corresponding bit in the Interrupt Mask Register (PIO_IMR). As input change detection is possible only by comparing two successive samplings of the input of the I/O line, the peripheral clock must be enabled. The Input Change interrupt is available regardless of the configuration of the I/O line, i.e., configured as an input only, controlled by the PIO Controller or assigned to a peripheral function.

By default, the interrupt can be generated at any time an edge is detected on the input.

Some additional interrupt modes can be enabled/disabled by writing in the Additional Interrupt Modes Enable Register (PIO_AIMER) and Additional Interrupt Modes Disable Register (PIO_AIMDR). The current state of this selection can be read through the Additional Interrupt Modes Mask Register (PIO_AIMMR).

These additional modes are:

- Rising edge detection
- Falling edge detection
- Low-level detection
- High-level detection

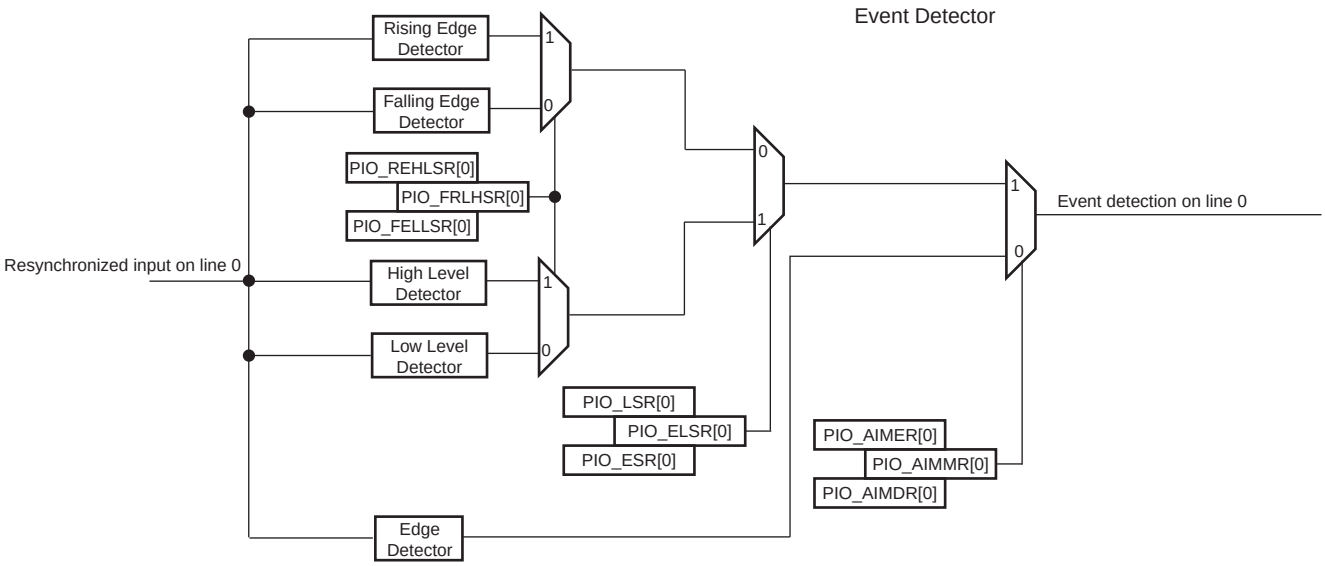
In order to select an additional interrupt mode:

- The type of event detection (edge or level) must be selected by writing in the Edge Select Register (PIO_ESR) and Level Select Register (PIO_LSR) which select, respectively, the edge and level detection. The current status of this selection is accessible through the Edge/Level Status Register (PIO_ELSR).
- The polarity of the event detection (rising/falling edge or high/low-level) must be selected by writing in the Falling Edge/Low-Level Select Register (PIO_FELLSR) and Rising Edge/High-Level Select Register (PIO_REHLSR) which allow to select falling or rising edge (if edge is selected in PIO_ELSR) edge or high- or low-level detection (if level is selected in PIO_ELSR). The current status of this selection is accessible through the Fall/Rise - Low/High Status Register (PIO_FRLHSR).

When an input edge or level is detected on an I/O line, the corresponding bit in the Interrupt Status Register (PIO_ISR) is set. If the corresponding bit in PIO_IMR is set, the PIO Controller interrupt line is asserted.The interrupt signals of the 32 channels are ORed-wired together to generate a single interrupt signal to the interrupt controller.

When the software reads PIO_ISR, all the interrupts are automatically cleared. This signifies that all the interrupts that are pending when PIO_ISR is read must be handled. When an Interrupt is enabled on a “level”, the interrupt is generated as long as the interrupt source is not cleared, even if some read accesses in PIO_ISR are performed.

Figure 16-6. Event Detector on Input Lines (Figure Represents Line 0)



Example of interrupt generation on following lines:

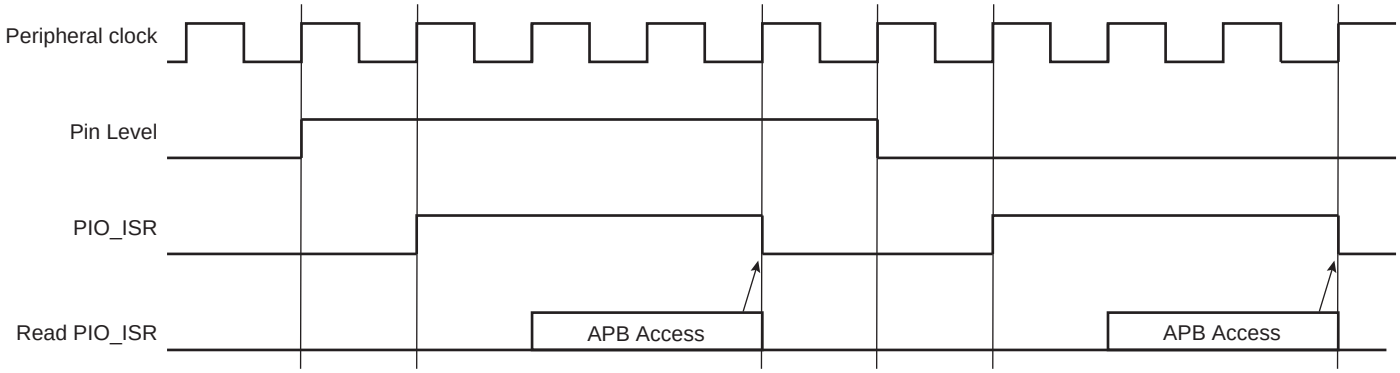
- Rising edge on PIO line 0
- Falling edge on PIO line 1
- Rising edge on PIO line 2
- Low-level on PIO line 3
- High-level on PIO line 4
- High-level on PIO line 5
- Falling edge on PIO line 6
- Rising edge on PIO line 7
- Any edge on the other lines

Table 16-2 provides the required configuration for this example.

Table 16-2. Configuration for Example Interrupt Generation

Configuration	Description
Interrupt Mode	All the interrupt sources are enabled by writing 32'hFFFF_FFFF in PIO_IER. Then the additional interrupt mode is enabled for lines 0 to 7 by writing 32'h0000_00FF in PIO_AIMER.
Edge or Level Detection	Lines 3, 4 and 5 are configured in level detection by writing 32'h0000_0038 in PIO_LSR. The other lines are configured in edge detection by default, if they have not been previously configured. Otherwise, lines 0, 1, 2, 6 and 7 must be configured in edge detection by writing 32'h0000_00C7 in PIO_ESR.
Falling/Rising Edge or Low/High-Level Detection	Lines 0, 2, 4, 5 and 7 are configured in rising edge or high-level detection by writing 32'h0000_00B5 in PIO_REHLSR. The other lines are configured in falling edge or low-level detection by default if they have not been previously configured. Otherwise, lines 1, 3 and 6 must be configured in falling edge/low-level detection by writing 32'h0000_004A in PIO_FELLSR.

Figure 16-7. Input Change Interrupt Timings When No Additional Interrupt Modes



16.5.11 Programmable I/O Drive

It is possible to configure the I/O drive for pads -. Refer to the section “Electrical Characteristics”.

16.5.12 Programmable Schmitt Trigger

It is possible to configure each input for the Schmitt trigger. By default the Schmitt trigger is active. Disabling the Schmitt trigger is requested when using the QTouch® Library.

16.5.13 I/O Lines Programming Example

The programming example shown in Table 16-3 is used to obtain the following configuration:

- 4-bit output port on I/O lines 0 to 3 (should be written in a single write operation), open-drain, with pull-up resistor
- Four output signals on I/O lines 4 to 7 (to drive LEDs for example), driven high and low, no pull-up resistor, no pull-down resistor
- Four input signals on I/O lines 8 to 11 (to read push-button states for example), with pull-up resistors, glitch filters and input change interrupts
- Four input signals on I/O line 12 to 15 to read an external device status (polled, thus no input change interrupt), no pull-up resistor, no glitch filter
- I/O lines 16 to 19 assigned to peripheral A functions with pull-up resistor
- I/O lines 20 to 23 assigned to peripheral B functions with pull-down resistor
- I/O lines 24 to 27 assigned to peripheral C with input change interrupt, no pull-up resistor and no pull-down resistor
- I/O lines 28 to 31 assigned to peripheral D, no pull-up resistor and no pull-down resistor

Table 16-3. Programming Example

Register	Value to be Written
PIO_PER	0x0000_FFFF
PIO_PDR	0xFFFF_0000
PIO_OER	0x0000_00FF
PIO_ODR	0xFFFF_FF00
PIO_IFER	0x0000_0F00
PIO_IFDR	0xFFFF_F0FF
PIO_SODR	0x0000_0000
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

16.5.14 Register Write Protection

To prevent any single software error from corrupting PIO behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the PIO Write Protection Mode Register (PIO_WPMR).

If a write access to a write-protected register is detected, the WPVS flag in the PIO Write Protection Status Register (PIO_WPSR) is set and the field WPVSRC indicates the register in which the write access has been attempted.

The WPVS bit is automatically cleared after reading the PIO_WPSR.

The following registers can be write-protected:

- PIO Enable Register
- PIO Disable Register
- PIO Output Enable Register
- PIO Output Disable Register
- PIO Input Filter Enable Register
- PIO Input Filter Disable Register
- PIO Multi-driver Enable Register
- PIO Multi-driver Disable Register
- PIO Pull-Up Disable Register
- PIO Pull-Up Enable Register
- PIO Peripheral ABCD Select Register 1
- PIO Peripheral ABCD Select Register 2
- PIO Output Write Enable Register
- PIO Output Write Disable Register
- PIO Pad Pull-Down Disable Register
- PIO Pad Pull-Down Enable Register

16.6 Parallel Input/Output Controller (PIO) User Interface

Each I/O line controlled by the PIO Controller is associated with a bit in each of the PIO Controller User Interface registers. Each register is 32-bit wide. If a parallel I/O line is not defined, writing to the corresponding bits has no effect. Undefined bits read zero. If the I/O line is not multiplexed with any peripheral, the I/O line is controlled by the PIO Controller and PIO_PSR returns one systematically.

Table 16-4. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	PIO Enable Register	PIO_PER	Write-only	–
0x0004	PIO Disable Register	PIO_PDR	Write-only	–
0x0008	PIO Status Register	PIO_PSR	Read-only	(1)
0x000C	Reserved	–	–	–
0x0010	Output Enable Register	PIO_OER	Write-only	–
0x0014	Output Disable Register	PIO_ODR	Write-only	–
0x0018	Output Status Register	PIO_OSR	Read-only	0x00000000
0x001C	Reserved	–	–	–
0x0020	Glitch Input Filter Enable Register	PIO_IFER	Write-only	–
0x0024	Glitch Input Filter Disable Register	PIO_IFDR	Write-only	–
0x0028	Glitch Input Filter Status Register	PIO_IFSR	Read-only	0x00000000
0x002C	Reserved	–	–	–
0x0030	Set Output Data Register	PIO_SODR	Write-only	–
0x0034	Clear Output Data Register	PIO_CODR	Write-only	
0x0038	Output Data Status Register	PIO_ODSR	Read-only or(2) Read/Write	–
0x003C	Pin Data Status Register	PIO_PDSR	Read-only	(3)
0x0040	Interrupt Enable Register	PIO_IER	Write-only	–
0x0044	Interrupt Disable Register	PIO_IDR	Write-only	–
0x0048	Interrupt Mask Register	PIO_IMR	Read-only	0x00000000
0x004C	Interrupt Status Register(4)	PIO_ISR	Read-only	0x00000000
0x0050	Multi-driver Enable Register	PIO_MDER	Write-only	–
0x0054	Multi-driver Disable Register	PIO_MDDR	Write-only	–
0x0058	Multi-driver Status Register	PIO_MDSR	Read-only	0x00000000
0x005C	Reserved	–	–	–
0x0060	Pull-up Disable Register	PIO_PUDR	Write-only	–
0x0064	Pull-up Enable Register	PIO_PUER	Write-only	–
0x0068	Pad Pull-up Status Register	PIO_PUSR	Read-only	(1)
0x006C	Reserved	–	–	–
0x0070	Peripheral ABCD Select Register 1	PIO_ABCDSR1	Read/Write	0x00000000
0x0074	Peripheral ABCD Select Register 2	PIO_ABCDSR2	Read/Write	0x00000000
0x0078–0x007C	Reserved	–	–	–

Table 16-4. Register Mapping (Continued)

Offset	Register	Name	Access	Reset
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	0x00000000
0x00E8	Write Protection Status Register	PIO_WPSR	Read-only	0x00000000
0x00EC–0x00FC	Reserved	–	–	–
0x0100	Schmitt Trigger Register	PIO_SCHMITT	Read/Write	0x00000000
0x0104–0x010C	Reserved	–	–	–
0x0110	I/O Drive Register	PIO_DRIVER	Read/Write	0x00000000
0x0114–0x011C	Reserved	–	–	–
0x0120–0x014C	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.

16.6.1 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 the PIO Write Protection Mode Register.

- **P0–P31: PIO Enable**

0: No effect.

1: Enables the PIO to control the corresponding pin (disables peripheral control of the pin).

16.6.2 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 the PIO Write Protection Mode Register.

- **P0–P31: PIO Disable**

0: No effect.

1: Disables the PIO from controlling the corresponding pin (enables peripheral control of the pin).

16.6.3 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).

16.6.4 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 the PIO Write Protection Mode Register.

- **P0–P31: Output Enable**

0: No effect.

1: Enables the output on the I/O line.

16.6.5 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 the PIO Write Protection Mode Register.

- **P0–P31: Output Disable**

0: No effect.

1: Disables the output on the I/O line.

16.6.6 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.

16.6.7 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 the PIO Write Protection Mode Register.

- **P0–P31: Input Filter Enable**

0: No effect.

1: Enables the input glitch filter on the I/O line.

16.6.8 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 the PIO Write Protection Mode Register.

- **P0–P31: Input Filter Disable**

0: No effect.

1: Disables the input glitch filter on the I/O line.

16.6.9 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 Filter Status**

0: The input glitch filter is disabled on the I/O line.

1: The input glitch filter is enabled on the I/O line.

16.6.10 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.

16.6.11 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.

16.6.12 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.

16.6.13 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.

16.6.14 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.

16.6.15 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.

16.6.16 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.

16.6.17 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.

16.6.18 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 the PIO Write Protection Mode Register.

- **P0-P31: Multi-drive Enable**

0: No effect.

1: Enables multi-drive on the I/O line.

16.6.19 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 the PIO Write Protection Mode Register.

- **P0–P31: Multi-drive Disable**

0: No effect.

1: Disables multi-drive on the I/O line.

16.6.20 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.

16.6.21 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 the PIO Write Protection Mode Register.

- **P0–P31: Pull-Up Disable**

0: No effect.

1: Disables the pull-up resistor on the I/O line.

16.6.22 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 the PIO Write Protection Mode Register.

- **P0–P31: Pull-Up Enable**

0: No effect.

1: Enables the pull-up resistor on the I/O line.

16.6.23 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.

16.6.24 PIO Peripheral ABCD Select Register 1

Name: PIO_ABCDSR1

Address: 0x400E0E70 (PIOA), 0x400E1070 (PIOB)

Access: Read/Write

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **P0–P31: Peripheral Select**

If the same bit is set to 0 in PIO_ABCDSR2:

0: Assigns the I/O line to the Peripheral A function.

1: Assigns the I/O line to the Peripheral B function.

If the same bit is set to 1 in PIO_ABCDSR2:

0: Assigns the I/O line to the Peripheral C function.

1: Assigns the I/O line to the Peripheral D function.

16.6.25 PIO Peripheral ABCD Select Register 2

Name: PIO_ABCDSR2

Address: 0x400E0E74 (PIOA), 0x400E1074 (PIOB)

Access: Read/Write

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **P0–P31: Peripheral Select**

If the same bit is set to 0 in PIO_ABCDSR1:

0: Assigns the I/O line to the Peripheral A function.

1: Assigns the I/O line to the Peripheral C function.

If the same bit is set to 1 in PIO_ABCDSR1:

0: Assigns the I/O line to the Peripheral B function.

1: Assigns the I/O line to the Peripheral D function.

16.6.26 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: Peripheral Clock Glitch Filtering Select**

0: No effect.

1: The glitch filter is able to filter glitches with a duration < $t_{peripheral\ clock}/2$.

16.6.27 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: Slow Clock Debouncing Filtering Select**

0: No effect.

1: The debouncing filter is able to filter pulses with a duration < $t_{div\_slck}/2$.

16.6.28 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 < $t_{peripheral\ clock}/2$.

1: The debouncing filter is able to filter pulses with a duration < $t_{div\_slck}/2$.

16.6.29 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**

$t_{div\_slck} = ((DIV + 1) \times 2) \times t_{slck}$

16.6.30 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 the PIO Write Protection Mode Register.

- **P0–P31: Pull-Down Disable**

0: No effect.

1: Disables the pull-down resistor on the I/O line.

16.6.31 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 the PIO Write Protection Mode Register.

- **P0–P31: Pull-Down Enable**

0: No effect.

1: Enables the pull-down resistor on the I/O line.

16.6.32 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

- **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.

16.6.33 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 the PIO Write Protection Mode Register.

- **P0–P31: Output Write Enable**

0: No effect.

1: Enables writing PIO_ODSR for the I/O line.

16.6.34 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 the PIO Write Protection Mode Register.

- **P0–P31: Output Write Disable**

0: No effect.

1: Disables writing PIO_ODSR for the I/O line.

16.6.35 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.

16.6.36 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.

16.6.37 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).

16.6.38 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: IO Line Index**

Selects the IO event type triggering an interrupt.

0: The interrupt source is a both-edge detection event.

1: The interrupt source is described by the registers PIO_ELSR and PIO_FRLHSR.

16.6.39 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.

16.6.40 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.

16.6.41 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.

16.6.42 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.

16.6.43 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.

16.6.44 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).

16.6.45 PIO Write Protection Mode Register

Name: PIO_WPMR

Address: 0x400E0EE4 (PIOA), 0x400E10E4 (PIOB)

Access: Read/Write

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

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x50494F (“PIO” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x50494F (“PIO” in ASCII).

See Section 16.5.14 “Register Write Protection” for the list of registers that can be protected.

- **WPKEY: Write Protection Key**

Value	Name	Description
0x50494F	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

16.6.46 PIO Write Protection Status Register

Name: PIO_WPSR

Address: 0x400E0EE8 (PIOA), 0x400E10E8 (PIOB)

Access: Read-only

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

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of the PIO_WPSR.

1: A write protection violation has occurred since the last read of the PIO_WPSR. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protection Violation Source**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

16.6.47 PIO Schmitt Trigger Register

Name: PIO_SCHMITT

Address: 0x400E0F00 (PIOA), 0x400E1100 (PIOB)

Access: Read/Write

31	30	29	28	27	26	25	24
SCHMITT31	SCHMITT30	SCHMITT29	SCHMITT28	SCHMITT27	SCHMITT26	SCHMITT25	SCHMITT24
23	22	21	20	19	18	17	16
SCHMITT23	SCHMITT22	SCHMITT21	SCHMITT20	SCHMITT19	SCHMITT18	SCHMITT17	SCHMITT16
15	14	13	12	11	10	9	8
SCHMITT15	SCHMITT14	SCHMITT13	SCHMITT12	SCHMITT11	SCHMITT10	SCHMITT9	SCHMITT8
7	6	5	4	3	2	1	0
SCHMITT7	SCHMITT6	SCHMITT5	SCHMITT4	SCHMITT3	SCHMITT2	SCHMITT1	SCHMITT0

- **SCHMITTx [x=0..31]: Schmitt Trigger Control**

0: Schmitt trigger is enabled.

1: Schmitt trigger is disabled.

16.6.48 PIO I/O Drive Register

Name: PIO_DRIVER

Address: 0x400E0F10 (PIOA), 0x400E1110 (PIOB)

Access: Read/Write

31	30	29	28	27	26	25	24
LINE31	LINE30	LINE29	LINE28	LINE27	LINE26	LINE25	LINE24
23	22	21	20	19	18	17	16
LINE23	LINE22	LINE21	LINE20	LINE19	LINE18	LINE17	LINE16
15	14	13	12	11	10	9	8
LINE15	LINE14	LINE13	LINE12	LINE11	LINE10	LINE9	LINE8
7	6	5	4	3	2	1	0
LINE7	LINE6	LINE5	LINE4	LINE3	LINE2	LINE1	LINE0

- **LINEx [x=0..31]: Drive of PIO Line x**

Value	Name	Description
0	LOW_DRIVE	Lowest drive
1	HIGH_DRIVE	Highest drive

17. Clock Generator

17.1 Description

The Clock Generator user interface is embedded within the Power Management Controller and is described in Section 18.20 ”Power Management Controller (PMC) User Interface”. However, the Clock Generator registers are named CKGR_.

17.2 Embedded Characteristics

The Clock Generator is made up of:

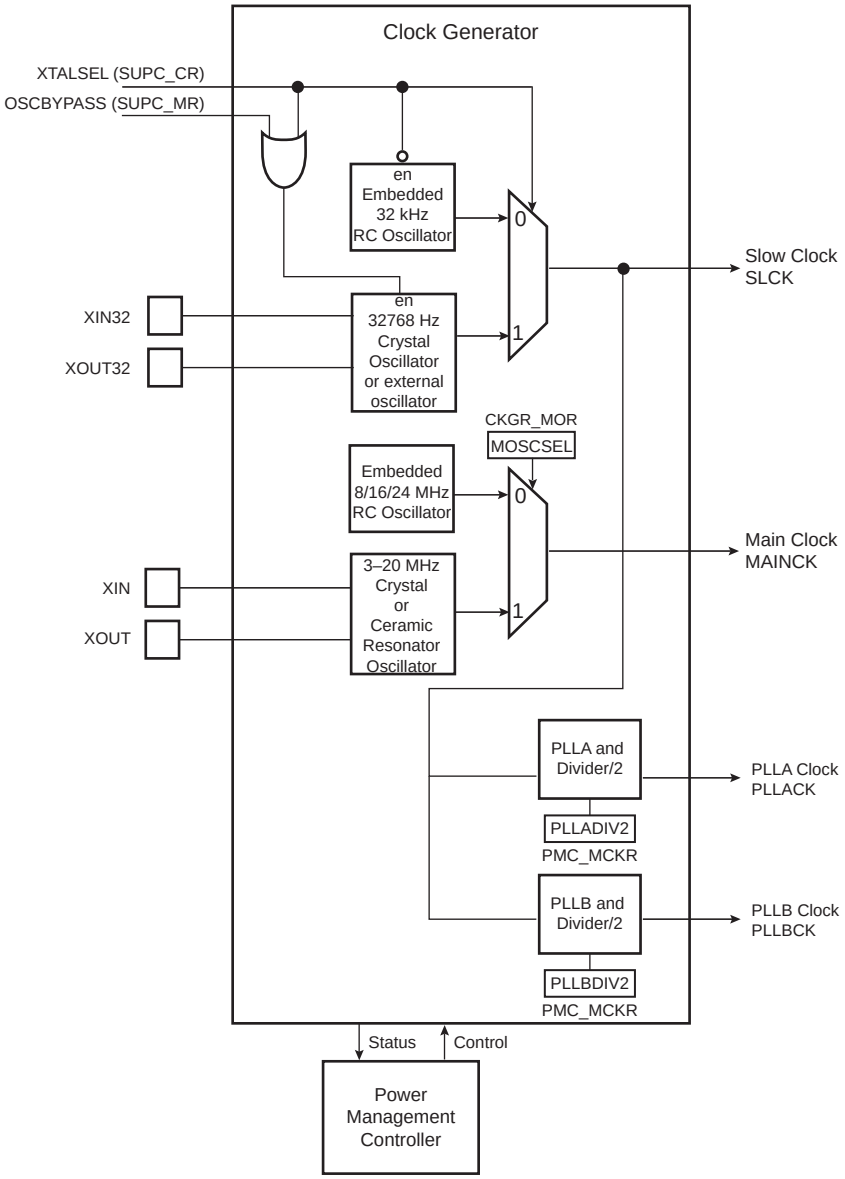
- A low-power 32768 Hz crystal oscillator with Bypass mode
- A low-power embedded 32 kHz (typical) RC oscillator
- A 3 to 20 MHz crystal or ceramic resonator-based oscillator, which can be bypassed.
- A factory-trimmed embedded RC oscillator. Three output frequencies can be selected: 8/16/24 MHz. By default 8 MHz is selected.
- Two programmable PLLs, (PLLA input from 20 to 300 KHz, output clock range 48 to 120 MHz and PLLB input from 20 to 100KHz, output clock range 24 to 48MHz), capable of providing the clock MCK to the processor and to the peripherals.

It provides the following clocks:

- SLCK, the slow clock, which is the only permanent clock within the system.
- MAINCK is the output of the main clock oscillator selection: either the crystal or ceramic resonator-based oscillator or 8/16/24 MHz RC oscillator.
- PLLACK is the output of the 48 to 120 MHz programmable PLL (PLLA).
- PLLBCK is the output of the 24 to 48MHz programmable PLL (PLLB).

17.3 Block Diagram

Figure 17-1. Clock Generator Block Diagram



17.4 Slow Clock

The Supply Controller embeds a slow clock generator that is supplied with the VDDIO power supply. As soon as VDDIO is supplied, both the 32768 Hz crystal oscillator and the embedded 32 kHz (typical) RC oscillator are powered up, but only the RC oscillator is enabled. This allows the slow clock to be valid in a short time (about 100 µs).

The slow clock is generated either by the 32768 Hz crystal oscillator or by the embedded 32 kHz (typical) RC oscillator.

The selection of the slow clock source is made via the XTALSEL bit in the Supply Controller Control Register (SUPC_CR).

The OSCSEL bit of the Supply Controller Status Register (SUPC_SR) and the OSCSELS bit of the PMC Status Register (PMC_SR) report which oscillator is selected as the slow clock source. PMC_SR.OSCSELS informs when the switch sequence initiated by a new value written in SUPC_CR.XTALSEL is done.

17.4.1 Embedded 32 kHz (typical) RC Oscillator

By default, the embedded 32 kHz (typical) RC oscillator is enabled and selected. The user has to take into account the possible drifts of this oscillator. More details are given in the section “DC Characteristics”.

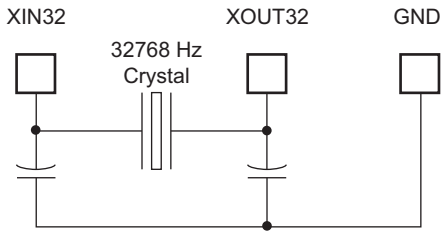
This oscillator is disabled by clearing the SUPC_CR.XTALSEL.

17.4.2 32768 Hz Crystal Oscillator

The Clock Generator integrates a low-power 32768 Hz crystal oscillator. To use this oscillator, the XIN32 and XOUT32 pins must be connected to a 32768 Hz crystal. Two external capacitors must be wired as shown in Figure 17-2. More details are given in the section “DC Characteristics”.

Note that the user is not obliged to use the 32768 Hz crystal oscillator and can use the 32 kHz (typical) RC oscillator instead.

Figure 17-2. Typical 32768 Hz Crystal Oscillator Connection



The 32768 Hz crystal oscillator provides a more accurate frequency than the 32 kHz (typical) RC oscillator.

To select the 32768 Hz crystal oscillator as the source of the slow clock, the bit SUPC_CR.XTALSEL must be set. This results in a sequence which first configures the PIO lines multiplexed with XIN32 and XOUT32 to be driven by the slow clock oscillator, then enables the 32768 Hz crystal oscillator and then disables the 32 kHz (typical) RC oscillator to save power. The switch of the slow clock source is glitch-free.

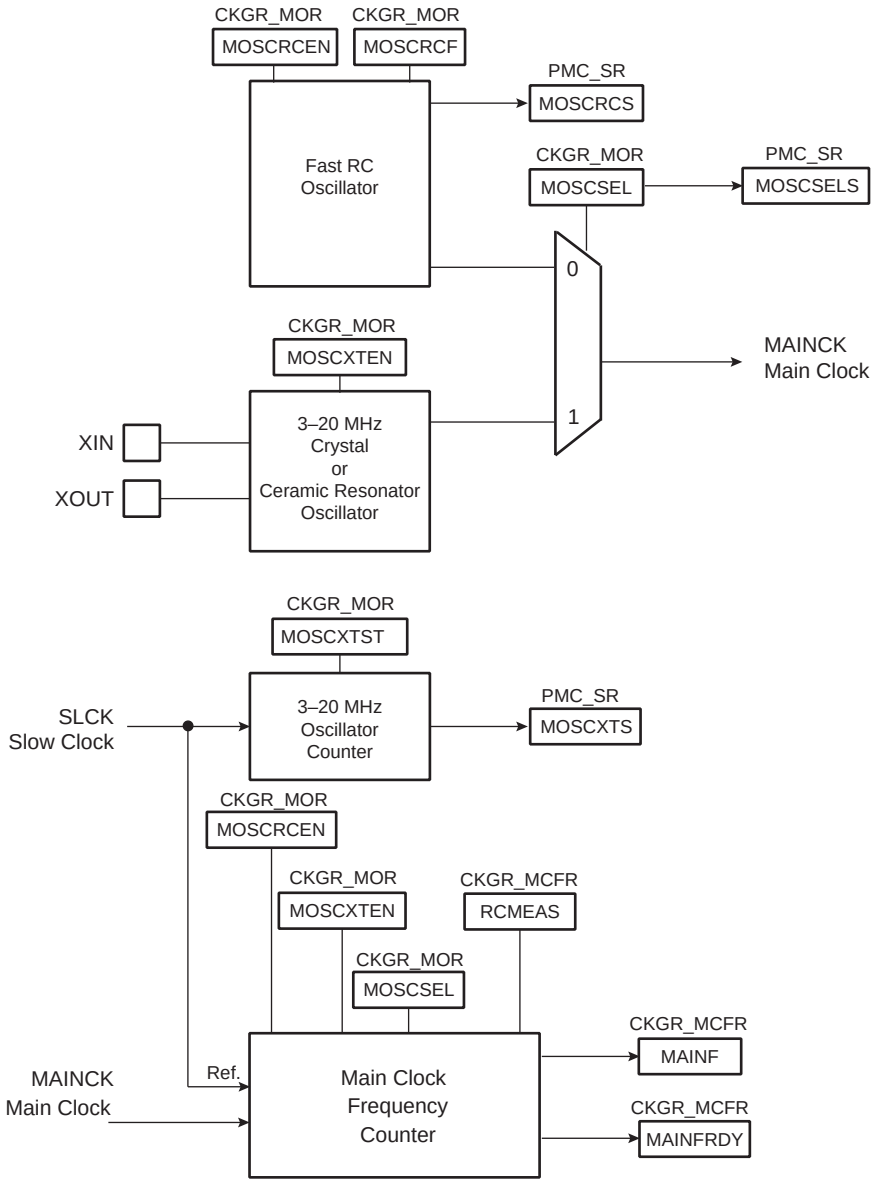
Reverting to the 32 kHz (typical) RC oscillator is only possible by shutting down the VDDIO power supply. If the user does not need the 32768 Hz crystal oscillator, the XIN32 and XOUT32 pins can be left unconnected since by default the XIN32 and XOUT32 system I/O pins are in PIO input mode with pull-up after reset.

The user can also set the 32768 Hz crystal oscillator in Bypass mode instead of connecting a crystal. In this case, the user must provide the external clock signal on XIN32. The input characteristics of the XIN32 pin are given in the section “Electrical Characteristics”. To enter Bypass mode, the OSCBYPASS bit of the Supply Controller Mode Register (SUPC_MR) must be set prior to setting SUPC_CR.XTALSEL.

17.5 Main Clock

Figure 17-3 shows the main clock block diagram.

Figure 17-3. Main Clock Block Diagram



The main clock has two sources:

- A 8/16/24 MHz RC oscillator with a fast start-up time and that is selected by default to start the system
- A 3 to 20 MHz crystal or ceramic resonator-based oscillator which can be bypassed

17.5.1 Embedded 8/16/24 MHz RC Oscillator

After reset, the 8/16/24 MHz RC oscillator is enabled with the 8 MHz frequency selected. This oscillator is selected as the source of MAINCK. MAINCK is the default clock selected to start the system.

The 8/16/24 MHz RC oscillator frequencies are calibrated in production except for the lowest frequency which is not calibrated.

Refer to the section “DC Characteristics”.

The software can disable or enable the 8/16/24 MHz RC oscillator with the MOSCRCEN bit in the Clock Generator Main Oscillator Register (CKGR_MOR).

The output frequency of the RC oscillator can be selected among 8/16/24 MHz. The selection is done via the CKGR_MOR.MOSCRCF field. When changing the frequency selection, the MOSCRCS bit in the Power Management Controller Status Register (PMC_SR) is automatically cleared and MAINCK is stopped until the oscillator is stabilized. Once the oscillator is stabilized, MAINCK restarts and PMC_SR.MOSCRCS is set.

When disabling the main clock by clearing the CKGR_MOR.MOSCRCEN bit, the PMC_SR.MOSCRCS bit is automatically cleared, indicating the main clock is off.

Setting the MOSCRCS bit in the Power Management Controller Interrupt Enable Register (PMC_IER) can trigger an interrupt to the processor.

When main clock (MAINCK) is not used to drive the processor and frequency monitor (SLCK or PLLACK or PLLBCK is used instead), it is recommended to disable the 8/16/24 MHz RC oscillator and 3 to 20 MHz crystal oscillator.

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 memory plane for code. These values cannot be modified by the user and cannot be erased by a Flash erase command or by the ERASE pin. Values written by the user application in PMC_OCR are reset after each power up or peripheral reset.

17.5.2 8/16/24 MHz RC Oscillator Clock Frequency Adjustment

It is possible for the user to adjust the 8/16/24 MHz RC oscillator frequency through PMC_OCR. By default, SEL8/16/24 bits are cleared, so the RC oscillator will be driven with Flash calibration bits which are programmed during chip production.

The user can adjust the trimming of the 8/16/24 MHz RC oscillator through this register. This can be used to compensate derating factors such as temperature and voltage, thus providing greater accuracy.

In order to calibrate the RC oscillator lower frequency, SEL8 bit must be set to 1 and a frequency value must be configured in the field CAL8. Likewise, SEL16/24 bit must be set to 1 and a trim value must be configured in the field CAL16/24 in order to adjust the other frequencies of the RC oscillator.

It is possible to adjust the RC oscillator frequency while operating from this clock. For example, when running on lowest frequency it is possible to change the CAL8 value if PMC_OCR.SEL8 bit is set.

At any time, it is possible to restart a measurement of the frequency of the selected clock via the RCMEAS bit in Main Clock Frequency Register (CKGR_MCFR). Thus, when CKGR_MCFR.MAINFRDY reads 1, another read access on CKGR_MCFR provides an image of the frequency on CKGR_MCFR.MAINF 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.

17.5.3 3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator

After reset, the 3 to 20 MHz crystal or ceramic resonator-based oscillator is disabled and is not selected as the source of MAINCK.

As the source of MAINCK, the 3 to 20 MHz crystal or ceramic resonator-based oscillator provides a very precise frequency. The software enables or disables this oscillator in order to reduce power consumption via CKGR_MOR.MOSCXTEN.

When disabling this oscillator by clearing the CKGR_MOR.MOSCXTEN, PMC_SR.MOSCXTS is automatically cleared, indicating the 3 to 20 MHz crystal oscillator is off.

When enabling this oscillator, the user must initiate the start-up time counter. The start-up time depends on the characteristics of the external device connected to this oscillator.

When CKGR_MOR.MOSCXTEN and CKGR_MOR.MOSCXTST are written to enable this oscillator, the XIN and XOUT pins are automatically switched into Oscillator mode. PMC_SR.MOSCXTS is cleared and the counter starts counting down on the slow clock divided by 8 from the CKGR_MOR.MOSCXTST value. Since the CKGR_MOR.MOSCXTST value is coded with 8 bits, the maximum start-up time is about 62 ms.

When the start-up time counter reaches 0, PMC_SR.MOSCXTS is set, indicating that the 3 to 20 MHz crystal oscillator is stabilized. Setting the MOSCXTS bit in the Interrupt Mask Register (PMC_IMR) can trigger an interrupt to the processor.

17.5.4 Main Clock Source Selection

The user can select the source of the main clock from either the 8/16/24 MHz RC oscillator, the 3 to 20 MHz crystal oscillator or the ceramic resonator-based oscillator.

The advantage of the 8/16/24 MHz RC oscillator is its fast start-up time. By default, this oscillator is selected to start the system and when entering Wait mode.

The advantage of the 3 to 20 MHz crystal oscillator or ceramic resonator-based oscillator is its precise frequency.

The selection of the oscillator is made by writing CKGR_MOR.MOSCSEL. The switch of the main clock source is glitch-free, so there is no need to run out of SLCK, PLLACK in order to change the selection. PMC_SR.MOSCSELS indicates when the switch sequence is done.

Setting PMC_IMR.MOSCSELS triggers an interrupt to the processor.

Enabling the 8/16/24 MHz RC oscillator (MOSCRCEN = 1) and changing its frequency (MOSCCRF) at the same time is not allowed.

This oscillator must be enabled first and its frequency changed in a second step.

17.5.5 Bypassing the 3 to 20 MHz Crystal Oscillator

Prior to bypassing the 3 to 20 MHz crystal oscillator, the external clock frequency provided on the XIN pin must be stable and within the values specified in the XIN Clock characteristics in the section “Electrical Characteristics”.

The sequence is as follows:

1. Ensure that an external clock is connected on XIN.
2. Enable the bypass by writing a 1 to CKGR_MOR.MOSCXTBY.
3. Disable the 3 to 20 MHz crystal oscillator by writing a 0 to bit CKGR_MOR.MOSCXTEN.

17.5.6 Main Clock Frequency Counter

The frequency counter is managed by CKGR_MCFR.

During the measurement period, the frequency counter increments at the main clock speed.

A measurement is started in the following cases:

- When the RCMEAS bit of CKGR_MCFR is written to 1.
- When the 8/16/24 MHz RC oscillator is selected as the source of main clock and when this oscillator becomes stable (i.e., when the MOSCRCS bit is set)
- When the 3 to 20 MHz crystal or ceramic resonator-based oscillator is selected as the source of main clock and when this oscillator becomes stable (i.e., when the MOSCXTS bit is set)
- When the main clock source selection is modified

The measurement period ends at the 16th falling edge of slow clock, the MAINFRDY bit in CKGR_MCFR is set and the counter stops counting. Its value can be read in the MAINF field of CKGR_MCFR and gives the number of clock cycles during 16 periods of slow clock, so that the frequency of the 8/16/24 MHz RC oscillator or 3 to 20 MHz crystal or ceramic resonator-based oscillator can be determined.

17.5.7 Switching Main Clock between the RC Oscillator and the Crystal Oscillator

When switching the source of the main clock between the RC oscillator and the crystal oscillator, both oscillators must be enabled. After completion of the switch, the unused oscillator can be disabled.

If switching to the crystal oscillator, a check must be carried out to ensure that the oscillator is present and that its frequency is valid. Follow the sequence below:

1. Select the slow clock as MCK by configuring bit CSS = 0 in the Master Clock Register (PMC_MCKR)).
2. Wait for PMC_SR.MCKRDY flag in PMC_SR to rise.
3. Enable the crystal oscillator by setting CKGR_MOR.MOSCXTEN. Configure the CKGR_MOR. MOSCXTST field with the crystal oscillator start-up time as defined in the section “Electrical Characteristics”.
4. Wait for PMC_SR.MOSCXTS flag to rise, indicating the end of a start-up period of the crystal oscillator.
5. Select the crystal oscillator as the source of the main clock by setting CKGR_MOR.MOSCSEL.
6. Read CKGR_MOR.MOSCSEL until its value equals 1.
7. Check the status of PMC_SR.MOSCSELS flag:
   - If MOSCSELS = 1: There is a crystal oscillator connected.

   a. Initiate a new frequency measurement by setting CKGR_MCFR.RCMEAS.
   b. Read CKGR_MCFR.MAINFRDY until its value equals 1.
   c. Read CKGR_MCFR.MAINF and compute the value of the crystal frequency.
   d. If the MAINF value is valid, the main clock can be switched to the crystal oscillator.
   - If MOSCSELS = 0:

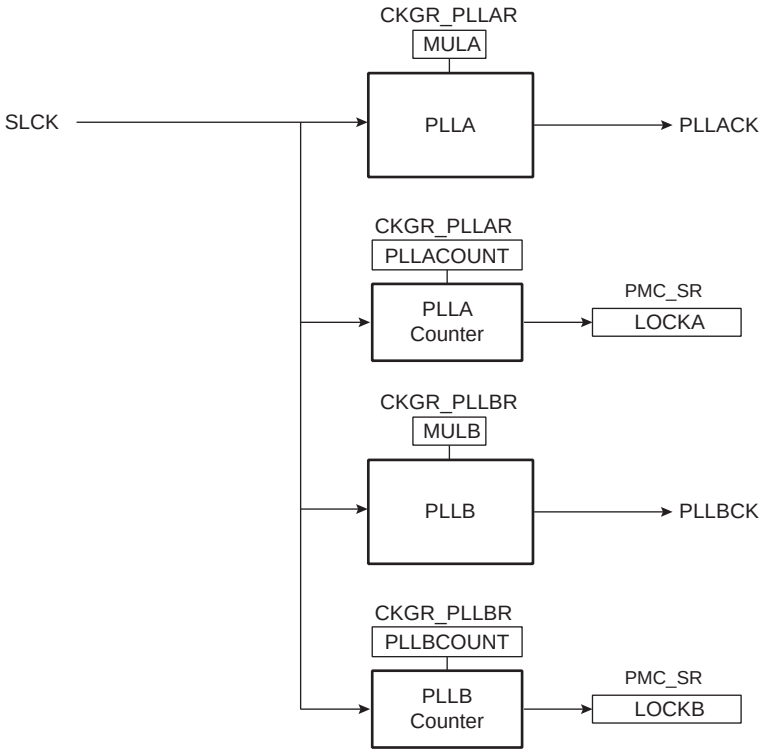
   a. There is no crystal oscillator connected or the crystal oscillator is out of specification.
   b. Select the RC oscillator as the source of the main clock by clearing CKGR_MOR.MOSCSEL.

17.6 Divider and PLL Block

The device features two PLL blocks that permit a wide range of frequencies to be selected on either the master clock, the processor clock or the programmable clock outputs. A 48 MHz clock signal is provided to the embedded USB device port regardless of the frequency of the main clock.

Figure 17-4 shows the block diagram of the divider and PLL blocks.

Figure 17-4. PLL Block Diagram



17.6.1 Phase Lock Loop Programming

The PLLs (PLLA, PLLB) allow multiplication of the SLCK clock source. The PLL clock signal has a frequency that depends on the respective source signal frequency and MUL (MULA) and PLLEN (PLLAEN, PLLBEN). The factor applied to the source signal frequency is MUL + 1. When MUL is written to 0 or PLLEN = 0, the PLL is disabled and its power consumption is saved. Note that there is a delay of two SLCK clock cycles between the disable command and the real disable of the PLL. Re-enabling the PLL can be performed by writing a value higher than 0 in the MUL field and PLLA(B)EN higher than 0.

Whenever the PLL is re-enabled or one of its parameters is changed, the LOCK (LOCKA, LOCKB) bit in PMC_SR is automatically cleared. The values written in the PLLCOUNT field (PLLACOUNT, PLLBCOUNT) in CKGR_PLLR (CKGR_PLLAR, CKGR_PLLBR) are loaded in the PLL counter. The PLL counter then decrements at the speed of the slow clock until it reaches 0. At this time, 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, PLLBDIV2) 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 and PLLB_MMAX fields in PMC_PMMR.

It is prohibited to change the frequency of the 8/16/24 MHz RC oscillator or to change the source of the main clock in CKGR_MOR while the master clock source is the PLL.

The user must:

1. Switch on the 8/16/24 MHz RC oscillator by writing a 1 to the CSS field of PMC_MCKR.
2. Change the frequency (MOSCRCF) or oscillator selection (MOSCSEL) in CKGR_MOR.
3. Wait for MOSCRCS (if frequency changes) or MOSCSELS (if oscillator selection changes) in PMC_SR.
4. Disable and then enable the PLL.
5. Wait for the LOCK flag in PMC_SR.
6. Switch back to the PLL by writing the appropriate value to the CSS field of PMC_MCKR.

18. Power Management Controller (PMC)

18.1 Description

The Power Management Controller (PMC) optimizes power consumption by controlling all system and user peripheral clocks. The PMC enables/disables the clock inputs to many of the peripherals and the Cortex-M4 processor.

The Supply Controller selects either the embedded 32 kHz RC oscillator or the 32768 Hz crystal oscillator. The unused oscillator is disabled automatically so that power consumption is optimized.

By default, at startup, the chip runs out of the master clock using the 8/16/24 MHz RC oscillator running at 8 MHz.

The user can trim the 16 and 24 MHz RC oscillator frequencies by software.

18.2 Embedded Characteristics

The PMC provides the following clocks:

- MCK, the Master Clock, programmable from a few hundred Hz to the maximum operating frequency of the device. It is available to the modules running permanently, such as the Enhanced Embedded Flash Controller.
- Processor Clock (HCLK) , automatically switched off when entering the processor in Sleep Mode
- Free-running processor Clock (FCLK)
- The Cortex-M4 SysTick external clock
- Peripheral Clocks, provided to the embedded peripherals (USART, SPI, TWI, TC, etc.) and independently controllable. Some of the peripherals can be configured to be driven by MCK divided by 2, 4, 8.
- Programmable Clock Outputs (PCKx), selected from the clock generator outputs to drive the device PCK pins
- Clock sources independent of MCK and HCLK, provided by internal PCKx for FLEXCOM (USART/SPI/TWI), Timer, ADCC and PDMIC

The PMC also provides the following features on clocks:

- A 3 to 20 MHz crystal oscillator clock failure detector
- A 32768 Hz crystal oscillator frequency monitor
- A frequency counter on main clock
- An on-the-fly adjustable 8/16/24 MHz RC oscillator frequency
- Asynchronous partial wakeup (SleepWalking) for FLEXCOM0-7, ADC

18.3 Block Diagram

Figure 18-1. General Clock Block Diagram

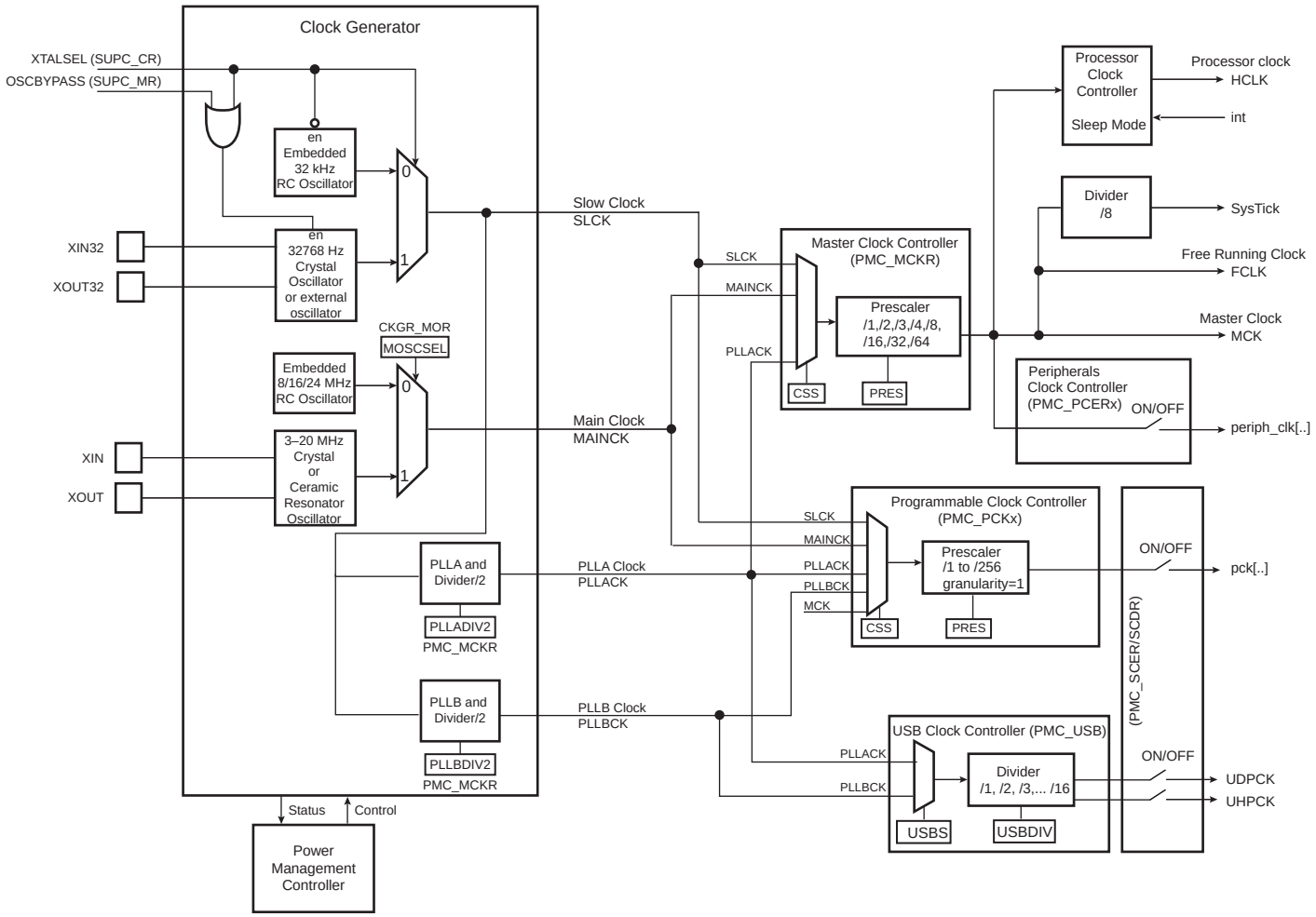
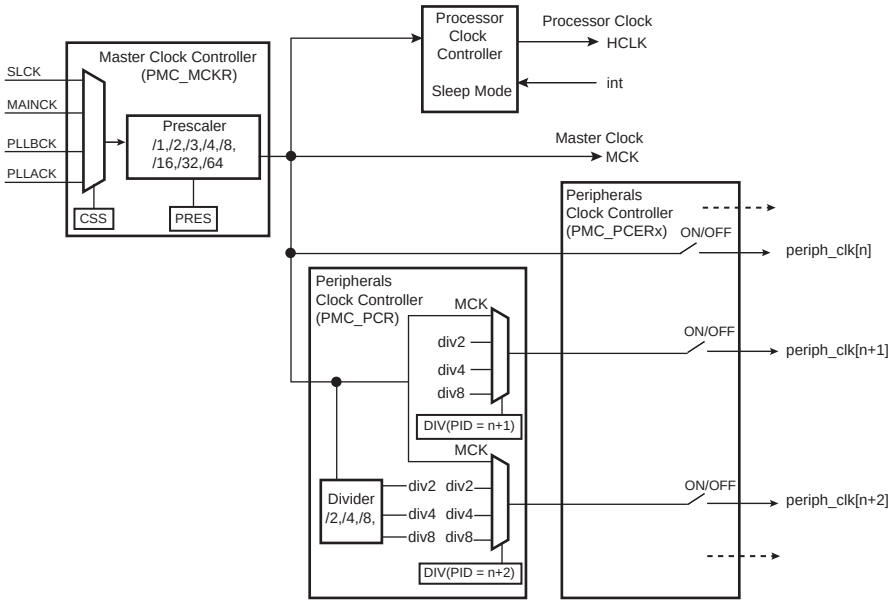


Figure 18-2. Peripheral Clock Divider Block Diagram



18.4 Master Clock Controller

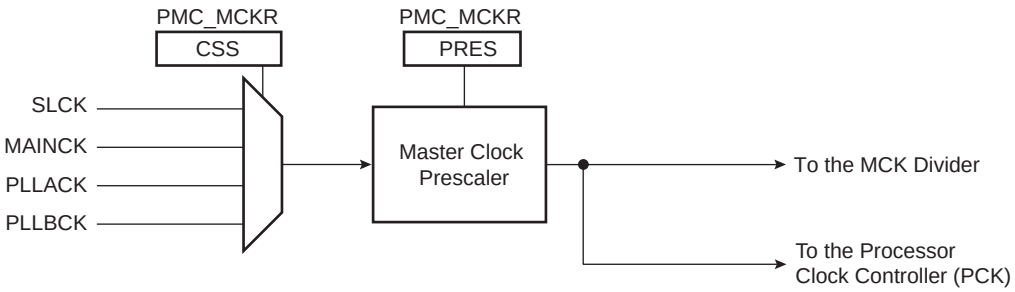
The Master Clock Controller provides selection and division of the master clock (MCK). MCK is the source clock of the peripheral clocks. The master clock is selected from one of the clocks provided by the Clock Generator.

Selecting the slow clock provides a slow clock signal to the whole device. Selecting the main clock saves power consumption of the PLL. The Master Clock Controller is made up of a clock selector and a prescaler.

The master clock selection is made by writing the CSS field (Clock Source Selection) in PMC_MCKR. The prescaler supports the division by a power of 2 of the selected clock between 1 and 64, and the division by 3. The PRES field in PMC_MCKR programs the prescaler.

Each time PMC_MCKR is written to define a new master clock, the MCKRDY bit is cleared in PMC_SR. It reads 0 until the master clock is established. Then, the MCKRDY bit is set and can trigger an interrupt to the processor. This feature is useful when switching from a high-speed clock to a lower one to inform the software when the change is actually done.

Figure 18-3. Master Clock Controller



18.5 Processor Clock Controller

The PMC features a Processor Clock Controller (HCLK) that implements the processor Sleep mode. These processor clock can be disabled by executing the WFI (WaitForInterrupt) or the WFE (WaitForEvent) processor instruction while the LPM bit is at 0 in the PMC Fast Startup Mode Register (PMC_FSMR).

The Processor Clock Controller HCLK is enabled after a reset and is automatically re-enabled by any enabled interrupt. The processor Sleep mode is entered by disabling the processor clock, which is automatically re-enabled by any enabled fast or normal interrupt, or by the reset of the product.

When processor Sleep mode is entered, the current instruction is finished before the clock is stopped, but this does not prevent data transfers from other masters of the system bus.

18.6 SysTick Clock

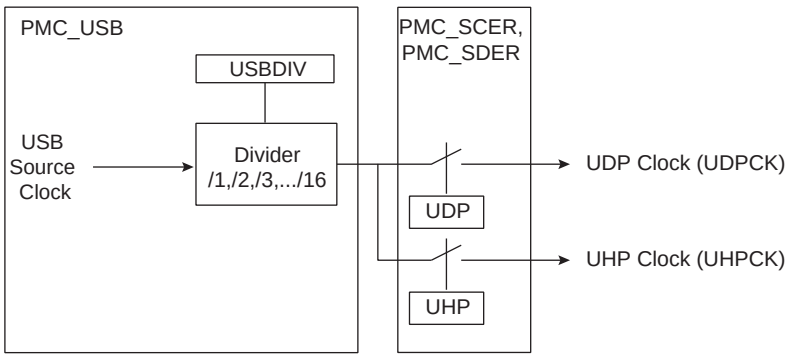
The SysTick calibration value is fixed to 12500 which allows the generation of a time base of 1 ms with SysTick clock to the maximum frequency on MCK divided by 8.

18.7 USB Clock Controller

The user can select the PLLA or the PLLB output as the USB source clock by writing the USBS bit in PMC_USB.If using the USB, the user must program the PLL to generate an appropriate frequency depending on the USBDIV bit in the USB Clock Register (PMC_USB).

When the PLL output is stable, i.e., the LOCK bit is set, the USB device and host FS clocks can be enabled by setting the UDP, UHP, bits in the System Clock Enable Register (PMC_SCER). To save power on this peripheral when it is not used, the user can set the UDP, UHP, bits in the System Clock Disable Register (PMC_SCDR). The UDP, UHP, bits in the System Clock Status Register (PMC_SCSR) gives the activity of this clock. The USB device and host ports require both the 48 MHz signal and the peripheral clock. The USB peripheral clock may be controlled by means of the Master Clock Controller.

Figure 18-4. USB Clock Controller



18.8 Peripheral Clock Controller

The PMC controls the clocks of each embedded peripheral by means of the Peripheral Clock Controller. The user can individually enable and disable the clock on the peripherals.

The user can also enable and disable these clocks by writing Peripheral Clock Enable 0 (PMC_PCER0), Peripheral Clock Disable 0 (PMC_PCDR0), Peripheral Clock Enable 1 (PMC_PCER1) and Peripheral Clock Disable 1 (PMC_PCDR1) registers. The status of the peripheral clock activity can be read in the Peripheral Clock Status Register (PMC_PCSR0) and Peripheral Clock Status Register (PMC_PCSR1).

When a peripheral clock is disabled, the clock is immediately stopped. The peripheral clocks are automatically disabled after a reset.

To stop a peripheral, it is recommended that the system software wait until the peripheral has executed its last programmed operation before disabling the clock. This is to avoid data corruption or erroneous behavior of the system.

The bit number within the Peripheral Clock Control registers (PMC_PCER0–1, PMC_PCDR0–1, and PMC_PCSR0–1) is the Peripheral Identifier defined at the product level. The bit number corresponds to the interrupt source number assigned to the peripheral.

In order to reduce power consumption, the clock of FLEXCOM0-7, PDMIC, TC0-5, ADC peripherals can be MCK divided by a division factor of 1, 2, 4, 8.

The divisor is defined in PMC_PCR. To apply a division factor, PID, CMD and DIV must be written in a single operation. The target peripheral clock is defined by the PID field. The divisor value is defined by DIV and the bit CMD must be set. To read the current division factor associated with a peripheral clock, two separate operations must be performed:

1. Write a one to the bit CMD and configure PID for the target peripheral clock. DIV is not significant for this operation.
2. Read the PMC_PCR. The value of DIV is the divisor applied on the peripheral clock defined by PID.

DIV must not be changed while peripheral is in use or when the peripheral clock is enabled. To change the clock division factor (DIV) of a peripheral, its clock must first be disabled by writing either EN to 0 for the corresponding PID (DIV must be kept the same if this method is used), or writing to the PMC_PCDR. Then, the second write must be performed in the PMC_PCR with the new value of DIV and the third write must be performed to enable the peripheral clock (by using either the PMC_PCR or PMC_PCER).

18.9 Asynchronous Partial Wakeup (SleepWalking)

18.9.1 Description

The asynchronous partial wakeup (SleepWalking) wakes up a peripheral in a fully asynchronous way when activity is detected on the communication line. Moreover, under some user configurable conditions, the asynchronous partial wakeup can trigger an exit of the system from Wait mode (full system wakeup).

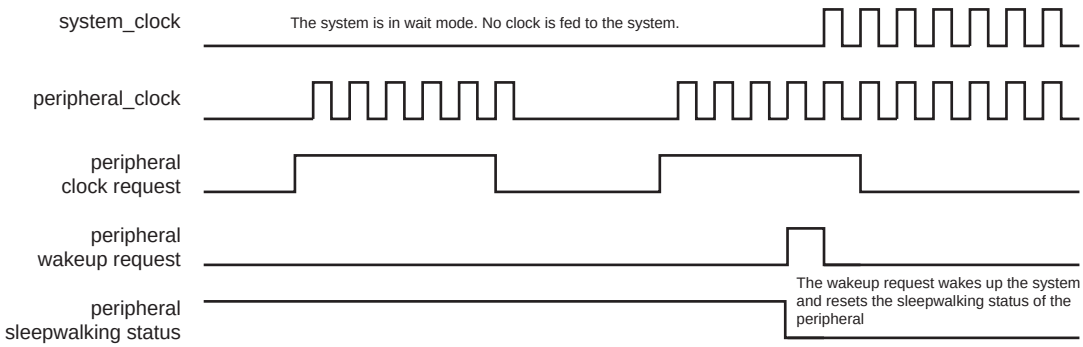
The asynchronous partial wakeup function automatically manages the peripheral clock. It improves the overall power consumption of the system by clocking peripherals only when needed.

Only the following peripherals can be configured with asynchronous partial wakeup: FLEXCOM0-7, ADC.

The peripheral selected for asynchronous partial wakeup must be first configured so that its clock is enabled by setting the appropriate PIDx bit in PMC_PCERx.

When the system is in Wait mode, all clocks of the system (except SLCK) are stopped. When an asynchronous clock request from a peripheral occurs, the PMC partially wakes up the system to feed the clock only to this peripheral. The rest of the system is not fed with the clock, thus optimizing power consumption. Finally, depending on user-configurable conditions, the peripheral either wakes up the whole system if these conditions are met or stops the peripheral clock until the next clock request. If a wakeup request occurs, the Asynchronous Partial Wakeup mode is automatically disabled until the user instructs the PMC to enable asynchronous partial wakeup. This is done by setting PIDx in the PMC SleepWalking Enable Register (PMC_SLPWK_ER).

Figure 18-5. SleepWalking During Wait Mode

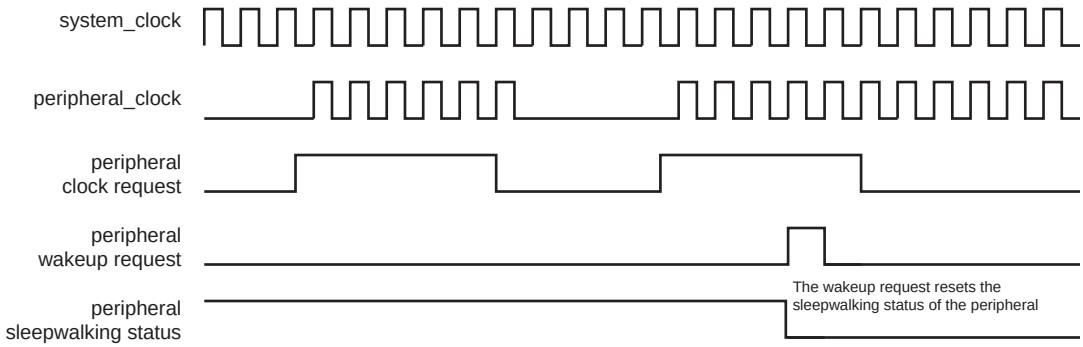


When the system is in Active mode, peripherals enabled for asynchronous partial wakeup have their respective clocks stopped until the peripherals request a clock. When a peripheral requests the clock, the PMC provides the clock without CPU intervention.

The triggering of the peripheral clock request depends on conditions which can be configured for each peripheral. If these conditions are met, the peripheral asserts a request to the PMC. The PMC disables the Asynchronous Partial Wakeup mode of the peripheral and provides the clock to the peripheral until the user instructs the PMC to re-enable partial wakeup on the peripheral. This is done by setting PIDx in the PMC_SLPWK_ER.

If the conditions are not met, the peripheral clears the clock request and PMC stops the peripheral clock until the clock request is re-asserted by the peripheral.

Figure 18-6. SleepWalking During Active Mode



18.9.2 Configuration Procedure

Before configuring the asynchronous partial wakeup (SleepWalking) function of a peripheral, check that the PIDx bit in the PMC Peripheral Clock Status Register (PMC_PCSR) is set. This ensures that the peripheral clock is enabled.

To enable the asynchronous partial wakeup (SleepWalking) function of a peripheral, follow the steps below:

1. Check that the corresponding PIDx bit in the PMC SleepWalking Activity Status Register (PMC_SLPWK_ASR) is cleared. This ensures that the peripheral has no activity in progress.
2. Enable the asynchronous partial wakeup function of the peripheral by writing a one to the corresponding PIDx bit in the PMC_SLPWK_ER.
3. Check that the corresponding PIDx bit in PMC_SLPWK_ASR is cleared. This ensures that no activity has started during the enable phase.
4. In the PMC_SLPWK_ASR, if the corresponding PIDx bit is set, the asynchronous partial wakeup function must be immediately disabled by writing a one to the PIDx bit in the PMC SleepWalking Disable Register (PMC_SLPWK_DR). Wait for the end of peripheral activity before reinitializing the procedure.
   If the corresponding PIDx bit is cleared, the peripheral clock is disabled and the system can now be placed in Wait mode.

Before entering Wait mode, check that all the PIDx bits in PMC_SLPWK_ASR are cleared.
This ensures that none of the peripherals has any activity in progress.

Note: When asynchronous partial wakeup (SleepWalking) of a peripheral is enabled and the core is running (system not in Wait mode), the peripheral must not be accessed before a wakeup of the peripheral is performed.

18.10 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).

18.11 Programmable Clock Output Controller

The PMC controls three signals to be output on external pins, PCKx. Each signal can be independently programmed via the Programmable Clock Registers (PMC_PCKx).

PCKx can be independently selected between the slow clock (SLCK), the main clock (MAINCK), the PLLA clock (PLLACK), the PLLB clock (PLLBCK),and the master clock (MCK) by writing the CSS field in PMC_PCKx. Each output signal can also be divided by a power of 2 between 1 and 64 by writing the PRES (Prescaler) field in PMC_PCKx.

Each output signal can be enabled and disabled by writing a 1 to the corresponding PCKx bit of PMC_SCER and PMC_SCDR, respectively. Status of the active programmable output clocks are given in the PCKx bits of PMC_SCSR.

The PCKRDYx status flag in PMC_SR indicates that the programmable clock is actually what has been programmed in the programmable clock registers.

As the Programmable Clock Controller does not manage with glitch prevention when switching clocks, it is strongly recommended to disable the programmable clock before any configuration change and to re-enable it after the change is actually performed.

18.12 Core and Bus Independent Clocks for Peripherals

The USART/UART/SPI/TWI/TIMER/PDMIC/ADCC can operate while the core, bus and peripheral clock frequencies are modified, thus providing communications at a rate which is independent for the core/bus/peripheral clock. This mode of operation is possible by using the internally generated independent clock sources PCK3 to PCK7.

PCK3 to PCK7 internal clocks can be independently selected between the slow clock (SLCK), the main clock (MAINCK), any available PLL clock, and the master clock (MCK) by writing the CSS field in the Programmable Clock Registers (PMC_PCK3 to PMC_PCK7). The independent clock sources can be also divided by writing the field PRES.

Each internal clock signal (PCKx) can be enabled and disabled by writing a one to the corresponding PCKx bit of PMC_SCER and PMC_SCDR, respectively. The status of the internal clocks are given in the PCKx bits of PMC_SCSR.

The PCKRDYx status flag in PMC_SR indicates that the programmable internal clock has been programmed in the programmable clock registers.

The independent clock source must also be selected in each peripheral (USART/UART/SPI/TWI/TIMER/PDMIC/ADCC) to operate communications, timings, etc without influence of the frequency of the core/bus/peripherals (except frequency limitations listed in each peripheral).

Table 18-1. Clock Assignment

Clock Name	Peripheral
PCK3	TC
PCK4	PDMIC, I2SC0/1
PCK5	ADCC
PCK6	FLEXCOM0/1/2/3 (USART/SPI/TWI)
PCK7	FLEXCOM4/5/6/7 (USART/SPI/TWI)

18.13 Fast Startup

At exit from Wait mode, the device allows the processor to restart in less than 10 microseconds only if the C-code function that manages the Wait mode entry and exit is linked to and executed from on-chip SRAM.

The fast startup time cannot be achieved if the first instruction after an exit is located in the embedded Flash.

If fast startup is not required, or if the first instruction after a Wait mode exit is located in embedded Flash, see Section 18.14 ”Startup from Embedded Flash”.

Prior to instructing the device to enter Wait mode:

1. Select the 8/16/24 MHz RC oscillator as the master clock source (the CSS field in PMC_MCKR must be written to 1).
2. Disable the PLL if enabled.
3. Wait for two SLCK clock cycles.
4. Clear the internal wakeup sources.
5. Verify that none of the enabled external wakeup 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 the wakeup input pins WKUPx (up to 16 inputs, see Peripheral Signal Multiplexing on I/O Lines section for exact number) or upon an active alarm from the RTC, RTT and USB Controller. The polarity of each wake-up input is programmable by writing the PMC Fast Startup Polarity Register (PMC_FSPR).

WARNING: The duration of the WKUPx pins active level must be greater than four main clock cycles.

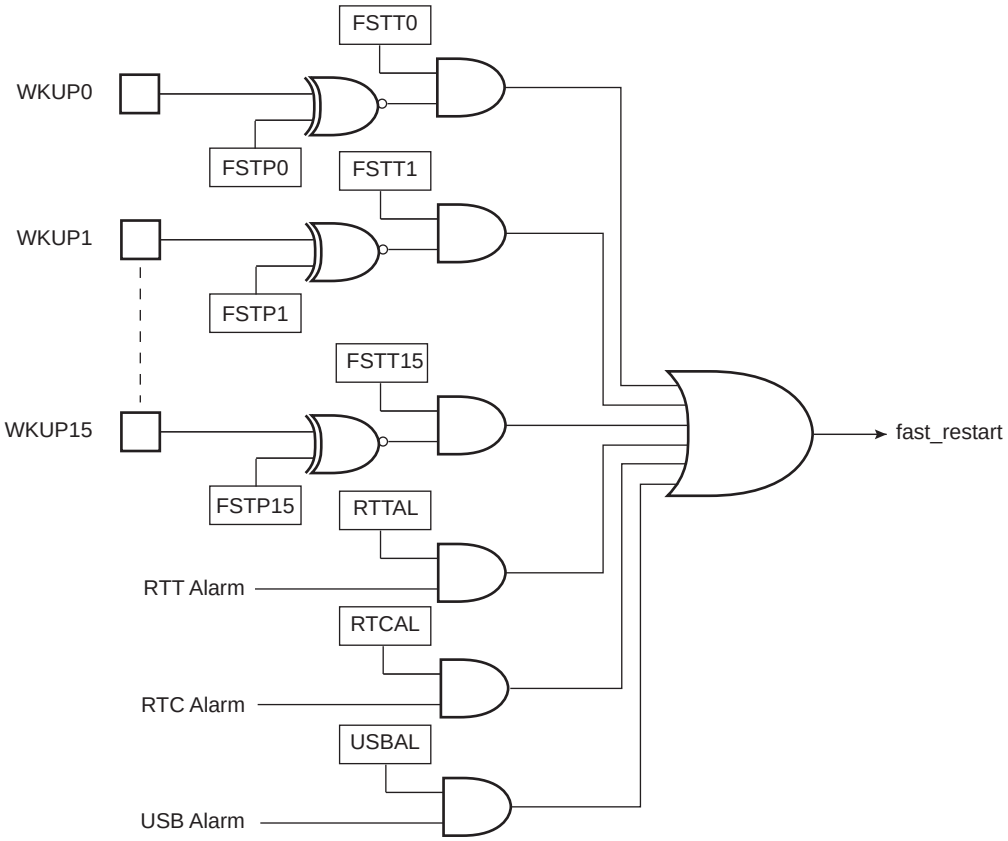
The fast startup circuitry, as shown in Figure 18-7, is fully asynchronous and provides a fast startup signal to the PMC. As soon as the fast startup signal is asserted, the embedded 8/16/24 MHz RC oscillator restarts automatically.

When entering Wait mode, the embedded Flash can be placed in one of the Low-power modes (Deep-power-down or Standby modes) depending on the configuration of the FLPM field in the PMC_FSMR. The FLPM field can be programmed at anytime and its value will be applied to the next Wait mode period.

The power consumption reduction is optimal when configuring 1 (Deep-power-down mode) in field FLPM. If 0 is programmed (Standby mode), the power consumption is slightly higher than in Deep-power-down mode.

When programming 2 in field FLPM, the Wait mode Flash power consumption is equivalent to that of the Active mode when there is no read access on the Flash.

Figure 18-7. Fast Startup Circuitry



Each wakeup input pin and alarm can be enabled to generate a fast startup event by setting the corresponding bit in PMC_FSMR.

The user interface does not provide any status for fast startup, but the user can easily recover this information by reading the PIO Controller and the status registers of the RTC, RTT and USB Controller.

18.14 Startup from Embedded Flash

The inherent start-up time of the embedded Flash cannot provide a fast startup of the system.

If system fast start-up time is not required, the first instruction after a Wait mode exit can be located in the embedded Flash. Under these conditions, prior to entering Wait mode, the Flash controller must be programmed to perform access in 0 wait-state. Refer to the section “Enhanced Embedded Flash Controller (EEFC)”.

If the 8/16/24 MHz 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 18.13 "Fast Startup"). If this 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 18.13 "Fast Startup").

18.15 Main Clock Failure Detector

The clock failure detector monitors the 3 to 20 MHz crystal oscillator or ceramic resonator-based oscillator to identify a failure of this oscillator when selected as main clock.

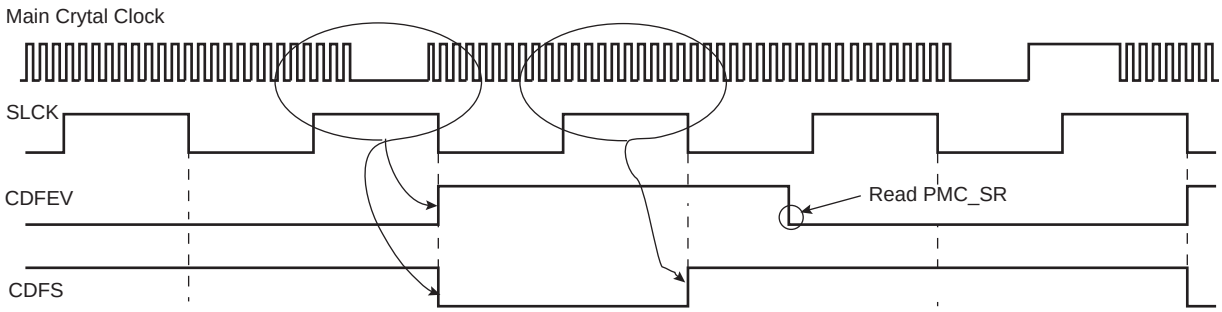
The clock failure detector can be enabled or disabled by bit CFDEN in CKGR_MOR. After a VDDCORE reset, the detector is disabled. However, if the oscillator is disabled (MOSCXTEN = 0), the detector is also disabled.

A failure is detected by means of a counter incrementing on the main clock and detection logic is triggered by the 32 kHz (typical) RC oscillator which is automatically enabled when CFDEN=1.

The counter is cleared when the 32 kHz (typical) RC oscillator clock signal is low and enabled when the signal is high. Thus, the failure detection time is one RC oscillator period. If, during the high level period of the 32 kHz (typical) RC oscillator clock signal, less than eight 3 to 20 MHz crystal oscillator clock periods have been counted, then a failure is reported.

If a failure of the main clock is detected, bit CFDEV in PMC_SR indicates a failure event and generates an interrupt if the corresponding interrupt source is enabled. The interrupt remains active until a read occurs in PMC_SR. The user can know the status of the clock failure detection at any time by reading the CFDS bit in PMC_SR.

Figure 18-8. Clock Failure Detection (Example)



Note: ratio of clock periods is for illustration purposes only

If the 3 to 20 MHz crystal oscillator or ceramic resonator-based oscillator is selected as the source clock of MAINCK (MOSCSEL in CKGR_MOR = 1), and if MCK source is PLLACK or PLLBCK (CSS = 2), a clock failure detection automatically forces MAINCK to be the source clock for MCK. Then, regardless of the PMC configuration, a clock failure detection automatically forces the 8/16/24 MHz RC oscillator to be the source clock for MAINCK. If this oscillator is disabled when a clock failure detection occurs, it is automatically re-enabled by the clock failure detection mechanism.

It takes two 32 kHz (typical) RC oscillator clock cycles to detect and switch from the 3 to 20 MHz crystal oscillator, to the 8/16/24 MHz RC oscillator if the source master clock (MCK) is main clock (MAINCK), or three 32 kHz (typical) RC oscillator clock cycles if the source of MCK is PLLACK or PLLBCK.

The 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).

18.16 32768 Hz Crystal Oscillator Frequency Monitor

The frequency of the 32768 Hz crystal oscillator can be monitored by means of logic driven by the 8/16/24 MHz RC oscillator known as a reliable clock source. This function is enabled by configuring the XT32KFME bit of CKGR_MOR. The SEL4/SEL8/SEL12 bits of PMC_OCR must be cleared.

An error flag (XT32KERR in PMC_SR) is asserted when the 32768 Hz crystal oscillator frequency is out of the ±60% nominal frequency value (i.e., 32768 Hz). The error flag can be cleared only if the slow clock frequency monitoring is disabled.

The monitored clock frequency is declared invalid if at least four consecutive clock period measurement results are over the nominal period ±60%.

Due to the possible frequency variation of the embedded 8/16/24 MHz RC oscillator acting as reference clock for the monitor logic, any 32768 Hz crystal oscillator frequency deviation over ±60% of the nominal frequency is systematically reported as an error by the XT32KERR bit in PMC_SR. Between -1% and -60% and +1% and +60%, the error is not systematically reported.

Thus only a crystal running at 32768 Hz frequency ensures that the error flag will not be asserted. The permitted drift of the crystal is 10000 ppm (1%), which allows any standard crystal to be used.

If the 8/16/24 MHz RC frequency needs to be changed while the slow clock frequency monitor is operating, the monitoring must be stopped prior to changing the 8/16/24 MHz RC frequency. It can then be re-enabled as soon as MOSCRCS is set in PMC_SR.

The error flag can be defined as an interrupt source of the PMC by setting the XT32KERR bit of PMC_IER.

18.17 Programming Sequence

1. If the 3 to 20 MHz crystal oscillator is not required, the PLL and divider can be directly configured (Step 6.) else this oscillator must be started (Step 2.).
2. Enable the 3 to 20 MHz crystal oscillator by setting the MOSCXTEN field in CKGR_MOR:

   The user can define a start-up time. This is done by writing a value in the MOSCXTST field in CKGR_MOR. Once this register has been correctly configured, the user must wait for MOSCXTS field in PMC_SR to be set. This is done either by polling MOSCXTS in PMC_SR, or by waiting for the interrupt line to be raised if the associated interrupt source (MOSCXTS) has been enabled in PMC_IER.
3. Switch the MAINCK to the 3 to 20 MHz crystal oscillator by setting MOSCSEL in CKGR_MOR.
4. Wait for the MOSCSELS to be set in PMC_SR to ensure the switch is complete.
5. Check the main clock frequency:

   This main clock frequency can be measured via CKGR_MCFR.

   Read CKGR_MCFR until the MAINFRDY field is set, after which the user can read the MAINF field in CKGR_MCFR by performing an additional read. This provides the number of main clock cycles that have been counted during a period of 16 slow clock cycles.

   If MAINF = 0, switch the MAINCK to the 8/16/24 MHz RC Oscillator by clearing MOSCSEL in CKGR_MOR. If MAINF ≠ 0, proceed to Step 6.
6. Set PLLx and Divider (if not required, proceed to Step 7.):

   In the names PLLx, DIVx, MULx, LOCKx, PLLxCOUNT, and CKGR_PLLxR, ‘x’ represents Aor B.

   All parameters needed to configure PLLx and the divider are located in CKGR_PLLxR.

   The DIVx field is used to control the divider itself. This parameter can be programmed between 0 and 127. Divider output is divider input divided by DIVx parameter. By default, DIVx field is cleared which means that the divider and PLLx are turned off.

The MULx field is the PLLx multiplier factor. This parameter can be programmed between 0 and 7500. If MULx is cleared, PLLx will be turned off, otherwise the PLLx output frequency is PLLx input frequency multiplied by (MULx + 1).

The PLLxCOUNT field specifies the number of slow clock cycles before the LOCKx bit is set in the PMC_SR after CKGR_PLLxR has been written.

Once CKGR_PLLxR has been written, the user must wait for the LOCKx bit to be set in the PMC_SR. This can be done either by polling LOCKx in PMC_SR or by waiting for the interrupt line to be raised if the associated interrupt source (LOCKx) has been enabled in PMC_IER. All fields in CKGR_PLLxR can be programmed in a single write operation. If at some stage one of the following parameters, MULx or DIVx is modified, the LOCKx bit goes low to indicate that PLLx is not yet ready. When PLLx is locked, LOCKx is set again. The user must wait for the LOCKx bit to be set before using the PLLx output clock.

7. Select the master clock and processor clock

The master clock and the processor clock are configurable via PMC_MCKR.

The CSS field is used to select the clock source of the master clock and processor clock dividers. By default, the selected clock source is the main clock.

The PRES field is used to define the processor clock and master clock prescaler. The user can choose between different values (1, 2, 3, 4, 8, 16, 32, 64). Prescaler output is the selected clock source frequency divided by the PRES value.

Once the PMC_MCKR has been written, the user must wait for the MCKRDY bit to be set in the PMC_SR. This can be done either by polling MCKRDY in PMC_SR or by waiting for the interrupt line to be raised if the associated interrupt source (MCKRDY) has been enabled in PMC_IER. PMC_MCKR must not be programmed in a single write operation. The programming sequence for PMC_MCKR is as follows:

- If a new value for CSS field corresponds to PLL clock,
  - Program the PRES field in PMC_MCKR.
  - Wait for the MCKRDY bit to be set in PMC_SR.
  - Program the CSS field in PMC_MCKR.
  - Wait for the MCKRDY bit to be set in PMC_SR.
- If a new value for CSS field corresponds to main clock or slow clock,
  - Program the CSS field in PMC_MCKR.
  - Wait for the MCKRDY bit to be set in the PMC_SR.
  - Program the PRES field in PMC_MCKR.
  - Wait for the MCKRDY bit to be set in PMC_SR.

If at some stage, parameters CSS or PRES are modified, the MCKRDY bit goes low to indicate that the master clock and the processor clock are not yet ready. The user must wait for MCKRDY bit to be set again before using the master and processor clocks.

Note: IF PLLx clock was selected as the master clock and the user decides to modify it by writing in CKGR_PLLxR, the MCKRDY flag will go low while PLLx is unlocked. Once PLLx is locked again, LOCKx goes high and MCKRDY is set. While PLLx is unlocked, the master clock selection is automatically changed to slow clock for PLLA and main clock for PLLB. For further information, see Section 18.18.2 ”Clock Switching Waveforms”.

Code Example:

```
write_register(PMC_MCKR,0x00000001)
wait (MCKRDY=1)
write_register(PMC_MCKR,0x00000011)
wait (MCKRDY=1)
```

The master clock is main clock divided by 2.

8. Select the programmable clocks

   Programmable clocks are controlled via registers, PMC_SCER, PMC_SCDR and PMC_SCSR.

   Programmable clocks can be enabled and/or disabled via PMC_SCER and PMC_SCDR. Three programmable clocks can be used. PMC_SCSR indicates which programmable clock is enabled. By default all programmable clocks are disabled.

   PMC_PCKx registers are used to configure programmable clocks.

   The CSS field is used to select the programmable clock divider source. Several clock options are available: main clock, slow clock, master clock, PLLACK, PLLBCK. The slow clock is the default clock source.

   The PRES field is used to control the programmable clock prescaler. It is possible to choose between different values (1, 2, 4, 8, 16, 32, 64). Programmable clock output is prescaler input divided by PRES parameter. By default, the PRES value is cleared which means that PCKx is equal to slow clock.

   Once PMC_PCKx register has been configured, the corresponding programmable clock must be enabled and the user is constrained to wait for the PCKRDYx bit to be set in the PMC_SR. This can be done either by polling PCKRDYx in PMC_SR or by waiting for the interrupt line to be raised if the associated interrupt source (PCKRDYx) has been enabled in PMC_IER. All parameters in PMC_PCKx can be programmed in a single write operation.

   If the CSS and PRES parameters are to be modified, the corresponding programmable clock must be disabled first. The parameters can then be modified. Once this has been done, the user must re-enable the programmable clock and wait for the PCKRDYx bit to be set.

9. Enable the peripheral clocks

   Once all of the previous steps have been completed, the peripheral clocks can be enabled and/or disabled via registers PMC_PCER0, PMC_PCER, PMC_PCDR0 and PMC_PCDR.

18.18 Clock Switching Details

18.18.1 Master Clock Switching Timings

Table 18-2 and Table 18-3 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 18-2. Clock Switching Timings (Worst Case)

From / To	Main Clock	SLCK	PLL Clock
Main Clock	–	4 x SLCK + 2.5 x Main Clock	3 x PLL Clock + 4 x SLCK + 1 x Main Clock
SLCK	0.5 x Main Clock + 4.5 x SLCK	–	3 x PLL Clock + 5 x SLCK
PLL Clock	0.5 x Main Clock + 4 x SLCK + PLLCOUNT x SLCK + 2.5 x PLLx Clock	2.5 x PLL Clock + 5 x SLCK + PLLCOUNT x SLCK	2.5 x PLL Clock + 4 x SLCK + PLLCOUNT x SLCK

Notes: 1. PLL designates either the PLLA or the PLLB Clock.
2. PLLCOUNT designates either PLLACOUNT or PLLBCOUNT.

Table 18-3. Clock Switching Timings between Two PLLs (Worst Case)

From / To	PLLA Clock	PLLB Clock
PLLA Clock	2.5 x PLLA Clock + 4 x SLCK + PLLACOUNT x SLCK	3 x PLLA Clock + 4 x SLCK + 1.5 x PLLA Clock
PLLB Clock	3 x PLLB Clock + 4 x SLCK + 1.5 x PLLB Clock	2.5 x PLLB Clock + 4 x SLCK+ PLLBCOUNT x SLCK

18.18.2 Clock Switching Waveforms

Figure 18-9. Switch Master Clock from Slow Clock to PLLx Clock

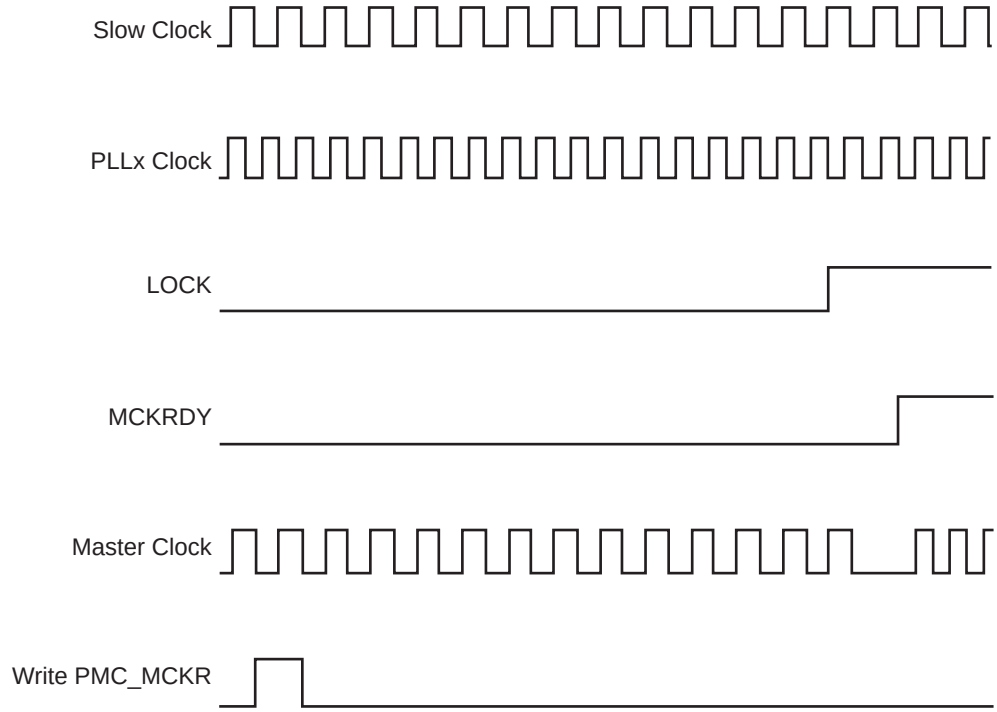


Figure 18-10. Switch Master Clock from Main Clock to Slow Clock

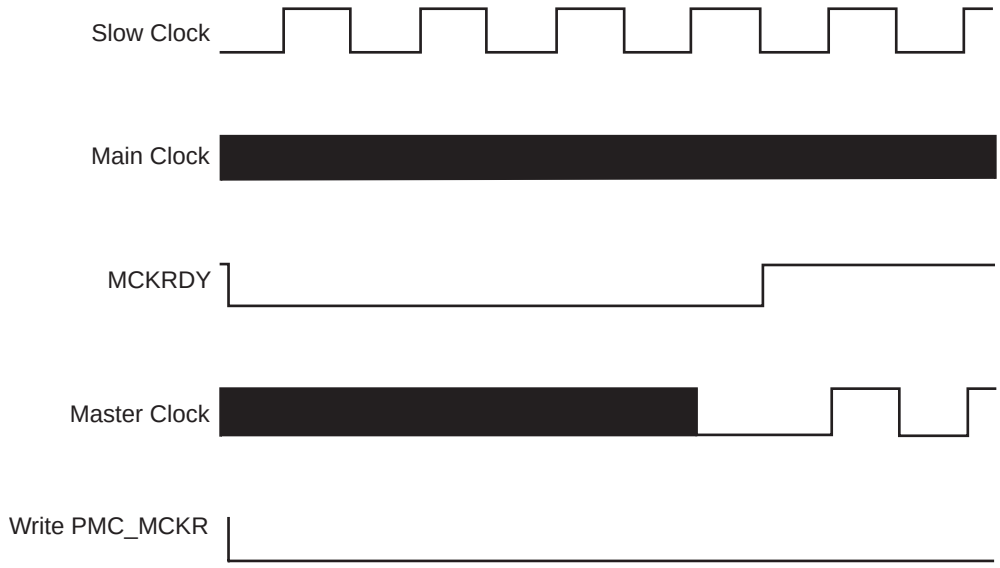


Figure 18-11. Change PLLx Programming

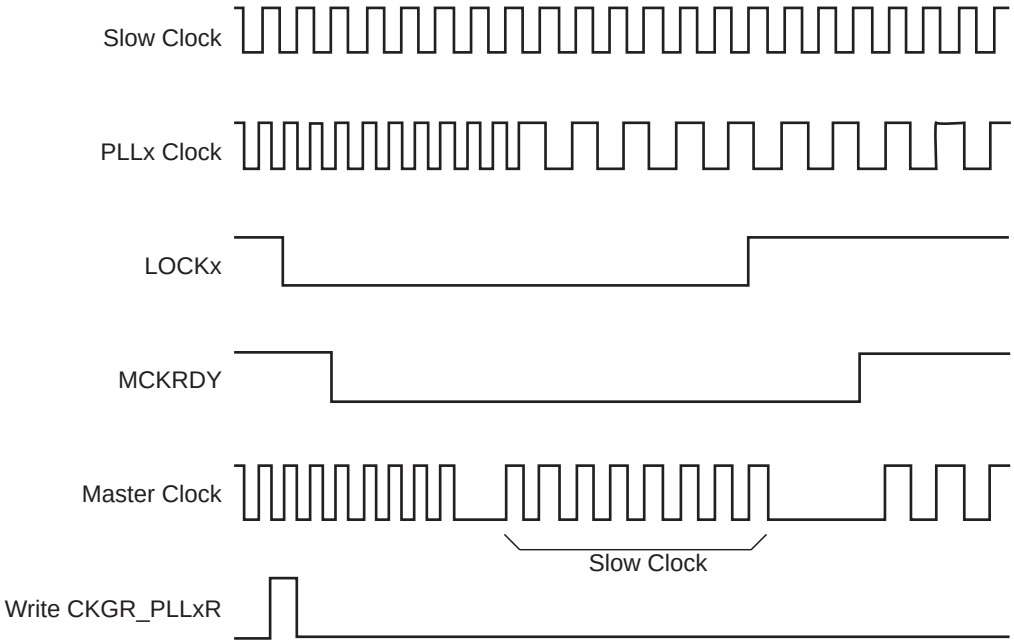
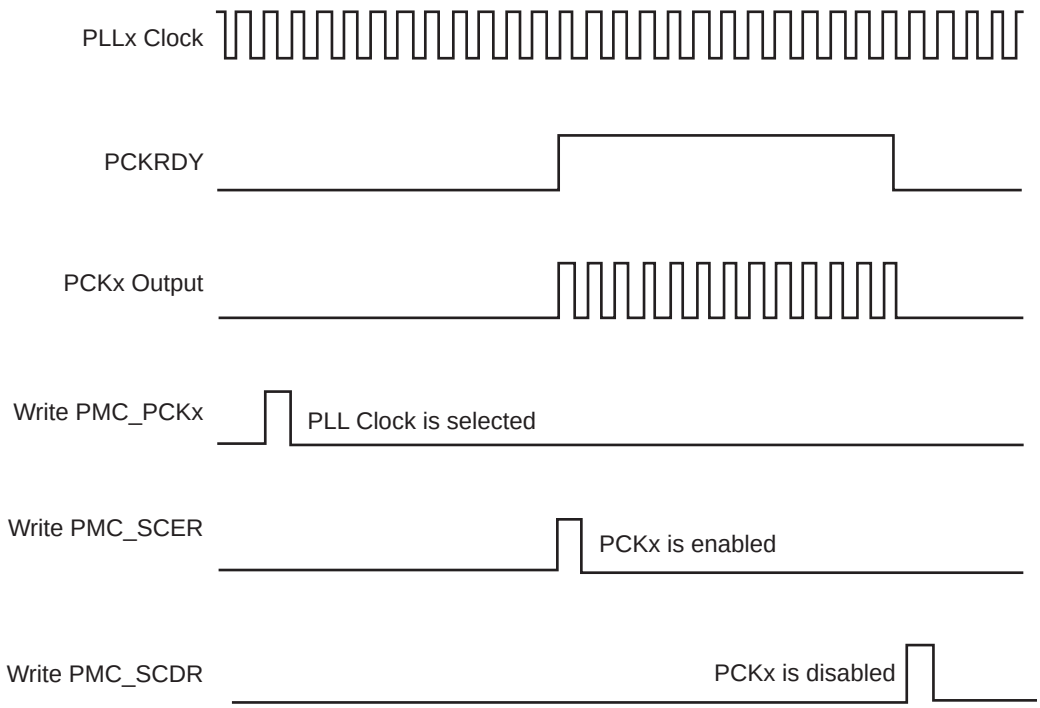


Figure 18-12. Programmable Clock Output Programming



18.19 Register Write Protection

To prevent any single software error from corrupting PMC behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the "PMC Write Protection Mode Register" (PMC_WPMR).

If a write access to a write-protected register is detected, the WPVS flag in the "PMC Write Protection Status Register" (PMC_WPSR) is set and the field WPVSRC indicates the register in which the write access has been attempted.

The WPVS bit is automatically cleared after reading the PMC_WPSR.

The following registers can be write-protected:

- "PMC System Clock Enable Register"
- "PMC System Clock Disable Register"
- "PMC Peripheral Clock Enable Register 0"
- "PMC Peripheral Clock Disable Register 0"
- "PMC Clock Generator Main Oscillator Register"
- "PMC Clock Generator Main Clock Frequency Register"
- "PMC Clock Generator PLLA Register"
- "PMC Clock Generator PLLB Register"
- "PMC Master Clock Register"
- "PMC USB Clock Register"
- "PMC Programmable Clock Register"
- "PMC Fast Startup Mode Register"
- "PMC Fast Startup Polarity Register"
- "PMC Peripheral Clock Enable Register 1"
- "PMC Peripheral Clock Disable Register 1"
- "PMC Oscillator Calibration Register"
- "PMC SleepWalking Enable Register 0"
- "PMC SleepWalking Disable Register 0"
- "PLL Maximum Multiplier Value Register"

18.20 Power Management Controller (PMC) User Interface

Table 18-4. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	System Clock Enable Register	PMC_SCER	Write-only	–
0x0004	System Clock Disable Register	PMC_SCDR	Write-only	–
0x0008	System Clock Status Register	PMC_SCSR	Read-only	0x0000_0001
0x000C	Reserved	–	–	–
0x0010	Peripheral Clock Enable Register 0	PMC_PCER0	Write-only	–
0x0014	Peripheral Clock Disable Register 0	PMC_PCDR0	Write-only	–
0x0018	Peripheral Clock Status Register 0	PMC_PCSR0	Read-only	0x0000_0000
0x0020	Main Oscillator Register	CKGR_MOR	Read/Write	0x0000_0008
0x0024	Main Clock Frequency Register	CKGR_MCFR	Read/Write	0x0000_0000
0x0028	PLLA Register	CKGR_PLLAR	Read/Write	0x0000_3F00
0x002C	PLLB Register	CKGR_PLLBR	Read/Write	0x0000_3F00
0x0030	Master Clock Register	PMC_MCKR	Read/Write	0x0000_0001
0x0034	Reserved	–	–	–
0x0038	USB Clock Register	PMC_USB	Read/Write	0x0000_0000
0x003C	Reserved	–	–	–
0x0040	Programmable Clock 0 Register	PMC_PCK0	Read/Write	0x0000_0000
0x0044	Programmable Clock 1 Register	PMC_PCK1	Read/Write	0x0000_0000
0x0048	Programmable Clock 2 Register	PMC_PCK2	Read/Write	0x0000_0000
0x004C	Programmable Clock 3 Register	PMC_PCK3	Read/Write	0x0000_0000
0x0050	Programmable Clock 4 Register	PMC_PCK4	Read/Write	0x0000_0000
0x0054	Programmable Clock 5 Register	PMC_PCK5	Read/Write	0x0000_0000
0x0058	Programmable Clock 6 Register	PMC_PCK6	Read/Write	0x0000_0000
0x005C	Programmable Clock 7 Register	PMC_PCK7	Read/Write	0x0000_0000
0x0060	Interrupt Enable Register	PMC_IER	Write-only	–
0x0064	Interrupt Disable Register	PMC_IDR	Write-only	–
0x0068	Status Register	PMC_SR	Read-only	0x0003_0008
0x006C	Interrupt Mask Register	PMC_IMR	Read-only	0x0000_0000
0x0070	Fast Startup Mode Register	PMC_FSMR	Read/Write	0x0000_0000
0x0074	Fast Startup Polarity Register	PMC_FSPR	Read/Write	0x0000_0000
0x0078	Fault Output Clear Register	PMC_FOCR	Write-only	–
0x007C–0x00E0	Reserved	–	–	–
0x00E4	Write Protection Mode Register	PMC_WPMR	Read/Write	0x0000_0000
0x00E8	Write Protection Status Register	PMC_WPSR	Read-only	0x0000_0000
0x00EC–0x00FC	Reserved	–	–	–
0x0100	Peripheral Clock Enable Register 1	PMC_PCER1	Write-only	–

Table 18-4. Register Mapping (Continued)

Offset	Register	Name	Access	Reset
0x0104	Peripheral Clock Disable Register 1	PMC_PCDR1	Write-only	–
0x0108	Peripheral Clock Status Register 1	PMC_PCSR1	Read-only	0x0000_0000
0x010C	Peripheral Control Register	PMC_PCR	Read/Write	0x0000_0000
0x0110	Oscillator Calibration Register	PMC_OCR	Read/Write	0x0040_4040
0x0114	SleepWalking Enable Register 0	PMC_SLPWK_ER0	Write-only	–
0x0118	SleepWalking Disable Register 0	PMC_SLPWK_DR0	Write-only	–
0x011C	SleepWalking Status Register 0	PMC_SLPWK_SR0	Read-only	0x0000_0000
0x0120	SleepWalking Activity Status Register 0	PMC_SLPWK_ASR0	Read-Only	–
0x0130	PLL Maximum Multiplier Value Register	PMC_PMMR	Read/Write	0x07FF_07FF

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

18.20.1 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
PCK7	PCK6	PCK5	PCK4	PCK3	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
UDP	UHP		–	–	–	–	–

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **UHP: USB Host Port Clock Enable**

0: No effect.

1: Enables the 48 MHz clock (UHPCK) of the USB Host Port.

- **UDP: USB Device Port Clock Enable**

0: No effect.

1: Enables the 48 MHz clock (UDPCK) of the USB Device Port.

- **PCKx: Programmable Clock x Output Enable**

0: No effect.

1: Enables the corresponding Programmable Clock output.

18.20.2 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
PCK7	PCK6	PCK5	PCK4	PCK3	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
UDP	UHP		–	–	–	–	–

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **UHP: USB Host Port Clock Disable**

0: No effect.

1: Disables the 48 MHz clock (UHPCK) of the USB Host Port.

- **PCKx: Programmable Clock x Output Disable**

0: No effect.

1: Disables the corresponding Programmable Clock output.

18.20.3 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
PCK7	PCK6	PCK5	PCK4	PCK3	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
UDP	UHP		–	–	–	–	–

- **UHP: USB Host Port Clock Status**

0: The 48 MHz clock (UHPCK) of the USB Device Port is disabled.

1: The 48 MHz clock (UHPCK) of the USB Host Port is enabled.

- **PCKx: Programmable Clock x Output Status**

0: The corresponding Programmable Clock output is disabled.

1: The corresponding Programmable Clock output is enabled.

18.20.4 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 the “PMC Write Protection Mode Register” .

- **PIDx: Peripheral Clock x Enable**

0: No effect.

1: Enables the corresponding peripheral clock.

Note: PIDx refers to identifiers defined in the section “Peripheral Identifiers”. Other peripherals can be enabled in PMC_PCER1 (Section 18.20.23 ”PMC Peripheral Clock Enable Register 1”).

Note: Programming the control bits of the Peripheral ID that are not implemented has no effect on the behavior of the PMC.

18.20.5 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 the “PMC Write Protection Mode Register” .

- **PIDx: Peripheral Clock x Disable**

0: No effect.

1: Disables the corresponding peripheral clock.

Note: PIDx refers to identifiers defined in the section “Peripheral Identifiers”. Other peripherals can be disabled in PMC_PCDR1 (Section 18.20.24 ”PMC Peripheral Clock Disable Register 1”).

18.20.6 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: PIDx refers to identifiers defined in the section “Peripheral Identifiers”. Other peripherals status can be read in PMC_PCSR1 (Section 18.20.25 ”PMC Peripheral Clock Status Register 1”).

18.20.7 PMC Clock Generator Main Oscillator Register

Name: CKGR_MOR

Address: 0x400E0420

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	XT32KFME	CFDEN	MOSCSEL

23	22	21	20	19	18	17	16
KEY							

15	14	13	12	11	10	9	8
MOSCXTST							

7	6	5	4	3	2	1	0
–	MOSCRCF			MOSCRCEN	WAITMODE	MOSCXTBY	MOSCXTEN

This register can only be written if the WPEN bit is cleared in the "PMC Write Protection Mode Register" .

- **MOSCXTEN: 3 to 20 MHz Crystal Oscillator Enable**

A crystal must be connected between XIN and XOUT.

0: The 3 to 20 MHz crystal oscillator is disabled.

1: The 3 to 20 MHz crystal oscillator is enabled. MOSCXTBY must be cleared.

When MOSCXTEN is set, the MOSCXTS flag is set once the crystal oscillator start-up time is achieved.

- **MOSCXTBY: 3 to 20 MHz Crystal Oscillator Bypass**

0: No effect.

1: The 3 to 20 MHz crystal oscillator is bypassed. MOSCXTEN must be cleared. An external clock must be connected on XIN.

When MOSCXTBY is set, the MOSCXTS flag in PMC_SR is automatically set.

Clearing MOSCXTEN and MOSCXTBY bits resets the MOSCXTS flag.

Note: When the crystal oscillator bypass is disabled (MOSCXTBY = 0), the MOSCXTS flag must be read at 0 in PMC_SR before enabling the crystal oscillator (MOSCXTEN = 1).

- **WAITMODE: Wait Mode Command (Write-only)**

0: No effect.

1: Puts the device in Wait mode.

- **MOSCRCEN: 8/16/24 MHz RC Oscillator Enable**

0: The 8/16/24 MHz RC oscillator is disabled.

1: The 8/16/24 MHz RC oscillator is enabled.

When MOSCRCEN is set, the MOSCRCS flag is set once the RC oscillator start-up time is achieved.

- **MOSCRCF: 8/16/24 MHz RC Oscillator Frequency Selection**

At startup, the RC oscillator frequency is 8 MHz.

Value	Name	Description
0	8_MHz	The RC oscillator frequency is at 8 MHz (default)
1	16_MHz	The RC oscillator frequency is at 16 MHz
2	24_MHz	The RC oscillator frequency is at 24 MHz

Note: MOSCRCF must be changed only if MOSCRCS is set in the PMC_SR. Therefore MOSCRCF and MOSCRCEN cannot be changed at the same time.

- **MOSCXTST: 3 to 20 MHz Crystal Oscillator Start-up Time**

Specifies the number of slow clock cycles multiplied by 8 for the crystal oscillator start-up time.

- **KEY: Write Access Password**

Value	Name	Description
0x37	PASSWD	Writing any other value in this field aborts the write operation. Always reads as 0.

- **MOSCSEL: Main Clock Oscillator Selection**

0: The 8/16/24 MHz RC oscillator is selected.

1: The 3 to 20 MHz crystal oscillator is selected.

- **CFDEN: Clock Failure Detector Enable**

0: The clock failure detector is disabled.

1: The clock failure detector is enabled.

Note: 1. The 32 kHz (typical) RC oscillator is automatically enabled when CFDEN=1.

- **XT32KFME: 32768 Hz Crystal Oscillator Frequency Monitoring Enable**

0: The 32768 Hz crystal oscillator frequency monitoring is disabled.

1: The 32768 Hz crystal oscillator frequency monitoring is enabled.

18.20.8 PMC Clock Generator Main Clock Frequency Register

Name: CKGR_MCFR

Address: 0x400E0424

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **MAINF: Main Clock Frequency**

Gives the number of main clock cycles within 16 slow clock periods. To calculate the frequency of the measured clock: $f_{MAINCK} = (MAINF \times f_{SLCK}) / 16$

where frequency is in MHz.

- **MAINFRDY: Main Clock Frequency Measure Ready**

0: MAINF value is not valid or the measured oscillator is disabled or a measure has just been started by means of RCMEAS.

1: The measured oscillator has been enabled previously and MAINF value is available.

Note: To ensure that a correct value is read on the MAINF field, the MAINFRDY flag must be read at 1 then another read access must be performed on the register to get a stable value on the MAINF field.

- **RCMEAS: Restart Main Clock Source Frequency Measure (write-only)**

0: No effect.

1: Restarts measuring of the frequency of the main clock source. MAINF will carry the new frequency as soon as a low to high transition occurs on the MAINFRDY flag.

The measure is performed on the main frequency (i.e. not limited to RC oscillator only), but if the main clock frequency source is the 3 to 20 MHz crystal oscillator, the restart of measuring is not needed because of the well known stability of crystal oscillators.

18.20.9 PMC Clock Generator PLLA Register

Name: CKGR_PLLAR

Address: 0x400E0428

Access: Read/Write

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

23	22	21	20	19	18	17	16
MULA							

15	14	13	12	11	10	9	8
–	–	PLLACOUNT					

7	6	5	4	3	2	1	0
PLLAEN							

Possible limitations on PLLA input frequencies and multiplier factors should be checked before using the PMC.

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **PLLAEN: PLLA Control**

0: PLLA is disabled

1: PLLA is enabled

2–255: Forbidden

- **PLLACOUNT: PLLA Counter**

Specifies the number of Slow Clock cycles before the LOCKA bit is set in PMC_SR after CKGR_PLLAR is written.

- **MULA: PLLA Multiplier**

0: The PLLA is deactivated (PLLA also disabled if DIVA = 0).

8 up to 7500 = The PLLA Clock frequency is the PLLA input frequency multiplied by MULA + 1.

Unlisted values are forbidden.

- **ZERO: Must Be Written to 0**

Bit 29 must always be written to 0 when programming the CKGR_PLLAR.

18.20.10 PMC Clock Generator PLLB Register

Name: CKGR_PLLBR

Address: 0x400E042C

Access: Read/Write

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

23	22	21	20	19	18	17	16
MULB							

15	14	13	12	11	10	9	8
–	–	PLLBCOUNT					

7	6	5	4	3	2	1	0
PLLBEN							

Possible limitations on PLLB input frequencies and multiplier factors should be checked before using the PMC.

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **PLLBEN: PLLB Control**

0: PLLB is disabled

1: PLLB is enabled

2–255: Forbidden

- **PLLBCOUNT: PLLB Counter**

Specifies the number of Slow Clock cycles before the LOCKB bit is set in PMC_SR after CKGR_PLLBR is written.

- **MULB: PLLB Multiplier**

0: The PLLB is deactivated (PLLB also disabled if DIVB = 0).

8 up to 2400: The PLLB Clock frequency is the PLLB input frequency multiplied by MULB + 1.

Unlisted values are forbidden.

- **ZERO: Must Be Written to 0**

18.20.11 PMC Master Clock Register

Name: PMC_MCKR

Address: 0x400E0430

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	PLLBDIV2	PLLADIV2	–	–	–	–
7	6	5	4	3	2	1	0
–	PRES			–	–	CSS	

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **CSS: Master Clock Source Selection**

Value	Name	Description
0	SLOW_CLK	Slow Clock is selected
1	MAIN_CLK	Main Clock is selected
2	PLLA_CLK	PLLA Clock is selected
3	PLLB_CLK	PLLB Clock is selected

- **PRES: Processor Clock Prescaler**

Value	Name	Description
0	CLK_1	Selected clock
1	CLK_2	Selected clock divided by 2
2	CLK_4	Selected clock divided by 4
3	CLK_8	Selected clock divided by 8
4	CLK_16	Selected clock divided by 16
5	CLK_32	Selected clock divided by 32
6	CLK_64	Selected clock divided by 64
7	CLK_3	Selected clock divided by 3

- **PLLADIV2: PLLA Divisor by 2**

PLLADIV2	PLLA Clock Division
0	PLLA clock frequency is divided by 1.
1	PLLA clock frequency is divided by 2.

- **PLLBDIV2: PLLB Divisor by 2**

PLLBDIV2	PLLB Clock Division
0	PLLB clock frequency is divided by 1.
1	PLLB clock frequency is divided by 2.

18.20.12 PMC USB Clock Register

Name: PMC_USB

Address: 0x400E0438

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **USBS: USB Input Clock Selection**

0: USB Clock Input is PLLA.

1: USB Clock Input is PLLB

- **USBDIV: Divider for USB Clock**

USB Clock is Input clock divided by USBDIV + 1.

18.20.13 PMC Programmable Clock Register

Name: PMC_PCKx

Address: 0x400E0440

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **CSS: Master Clock Source Selection**

Value	Name	Description
0	SLOW_CLK	Slow Clock is selected
1	MAIN_CLK	Main Clock is selected
2	PLLA_CLK	PLLA Clock is selected
3	PLLB_CLK	PLLB Clock is selected
4	MCK	Master Clock is selected

- **PRES: Programmable Clock Prescaler**

0–255: Selected clock is divided by PRES + 1.

18.20.14 PMC Interrupt Enable Register

Name: PMC_IER

Address: 0x400E0460

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	XT32KERR	–	–	CFDEV	MOSCRCS	MOSCSELS
15	14	13	12	11	10	9	8
PCKRDY7	PCKRDY6	PCKRDY5	PCKRDY4	PCKRDY3	PCKRDY2	PCKRDY1	PCKRDY0
7	6	5	4	3	2	1	0
–	–	–	–	MCKRDY	LOCKB	LOCKA	MOSCXTS

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **MOSCXTS: 3 to 20 MHz Crystal Oscillator Status Interrupt Enable**
- **LOCKA: PLLA Lock Interrupt Enable**
- **LOCKB: PLLB Lock Interrupt Enable**
- **MCKRDY: Master Clock Ready Interrupt Enable**
- **PCKRDYx: Programmable Clock Ready x Interrupt Enable**
- **MOSCSELS: Main Clock Source Oscillator Selection Status Interrupt Enable**
- **MOSCRCS: 8/16/24 MHz RC Oscillator Status Interrupt Enable**
- **CFDEV: Clock Failure Detector Event Interrupt Enable**
- **XT32KERR: 32768 Hz Crystal Oscillator Error Interrupt Enable**

18.20.15 PMC Interrupt Disable Register

Name: PMC_IDR

Address: 0x400E0464

Access: Write-only

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

23	22	21	20	19	18	17	16
–	–	XT32KERR	–	–	CFDEV	MOSCRCS	MOSCSELS

15	14	13	12	11	10	9	8
PCKRDY7	PCKRDY6	PCKRDY5	PCKRDY4	PCKRDY3	PCKRDY2	PCKRDY1	PCKRDY0

7	6	5	4	3	2	1	0
–	–	–	–	MCKRDY	LOCKB	LOCKA	MOSCXTS

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Disables the corresponding interrupt.

- **MOSCXTS: 3 to 20 MHz Crystal Oscillator Status Interrupt Disable**
- **LOCKA: PLLA Lock Interrupt Disable**
- **LOCKB: PLLB Lock Interrupt Disable**
- **MCKRDY: Master Clock Ready Interrupt Disable**
- **PCKRDYx: Programmable Clock Ready x Interrupt Disable**
- **MOSCSELS: Main Clock Source Oscillator Selection Status Interrupt Disable**
- **MOSCRCS: 8/16/24 MHz RC Oscillator Status Interrupt Disable**
- **CFDEV: Clock Failure Detector Event Interrupt Disable**
- **XT32KERR: 32768 Hz Oscillator Error Interrupt Disable**

18.20.16 PMC Status Register

Name: PMC_SR

Address: 0x400E0468

Access: Read-only

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

23	22	21	20	19	18	17	16
–	–	XT32KERR	FOS	CFDS	CFDEV	MOSCRCS	MOSCSELS

15	14	13	12	11	10	9	8
PCKRDY7	PCKRDY6	PCKRDY5	PCKRDY4	PCKRDY3	PCKRDY2	PCKRDY1	PCKRDY0

7	6	5	4	3	2	1	0
OSCSELS	–	–	–	MCKRDY	LOCKB	LOCKA	MOSCXTS

- **MOSCXTS: 3 to 20 MHz Crystal Oscillator Status**

0: 3 to 20 MHz crystal oscillator is not stabilized.

1: 3 to 20 MHz crystal oscillator is stabilized.

- **LOCKA: PLLA Lock Status**

0: PLLA is not locked

1: PLLA is locked.

- **LOCKB: PLLB Lock Status**

0: PLLB is not locked

1: PLLB is locked.

- **MCKRDY: Master Clock Status**

0: Master Clock is not ready.

1: Master Clock is ready.

- **OSCSELS: Slow Clock Oscillator Selection**

0: Embedded 32 kHz RC oscillator is selected.

1: 32768 Hz crystal oscillator is selected.

- **PCKRDYx: Programmable Clock Ready Status**

0: Programmable Clock x is not ready.

1: Programmable Clock x is ready.

- **MOSCSELS: Main Clock Source Oscillator Selection Status**

0: Selection is in progress.

1: Selection is done.

- **MOSCRCS: 8/16/24 MHz RC Oscillator Status**

0: 8/16/24 MHz RC oscillator is not stabilized.

1: 8/16/24 MHz RC oscillator is stabilized.

- **CFDEV: Clock Failure Detector Event**

0: No clock failure detection of the 3 to 20 MHz crystal oscillator has occurred since the last read of PMC_SR.

1: At least one clock failure detection of the 3 to 20 MHz crystal oscillator has occurred since the last read of PMC_SR.

- **CFDS: Clock Failure Detector Status**

0: A clock failure of the 3 to 20 MHz crystal oscillator is not detected.

1: A clock failure of the 3 to 20 MHz crystal oscillator is detected.

- **FOS: Clock Failure Detector Fault Output Status**

0: The fault output of the clock failure detector is inactive.

1: The fault output of the clock failure detector is active.

- **XT32KERR: 32768 Hz Crystal Oscillator Error**

0: The frequency of the 32768 Hz crystal oscillator is correct (32768 Hz +/- 1%) or the monitoring is disabled.

1: The frequency of the 32768 Hz crystal oscillator is incorrect or has been incorrect for an elapsed period of time since the monitoring has been enabled.

18.20.17 PMC Interrupt Mask Register

Name: PMC_IMR

Address: 0x400E046C

Access: Read-only

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

23	22	21	20	19	18	17	16
–	–	XT32KERR	–	–	CFDEV	MOSCRCS	MOSCSELS

15	14	13	12	11	10	9	8
–	PCKRDY6	PCKRDY5	PCKRDY4	PCKRDY3	PCKRDY2	PCKRDY1	PCKRDY0

7	6	5	4	3	2	1	0
–	–	–	–	MCKRDY	LOCKB	LOCKA	MOSCXTS

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **MOSCXTS: 3 to 20 MHz Crystal Oscillator Status Interrupt Mask**
- **LOCKA: PLLA Lock Interrupt Mask**
- **LOCKB: PLLB Lock Interrupt Mask**
- **MCKRDY: Master Clock Ready Interrupt Mask**
- **PCKRDYx: Programmable Clock Ready x Interrupt Mask**
- **MOSCSELS: Main Clock Source Oscillator Selection Status Interrupt Mask**
- **MOSCRCS: 8/16/24 MHz RC Oscillator Status Interrupt Mask**
- **CFDEV: Clock Failure Detector Event Interrupt Mask**
- **XT32KERR: 32768 Hz Oscillator Error Interrupt Mask**

18.20.18 PMC Fast Startup Mode Register

Name: PMC_FSMR

Address: 0x400E0470

Access: Read/Write

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

23	22	21	20	19	18	17	16
FFLPM	FLPM		LPM	–	USBAL	RTCAL	RTTAL

15	14	13	12	11	10	9	8
FSTT15	FSTT14	FSTT13	FSTT12	FSTT11	FSTT10	FSTT9	FSTT8

7	6	5	4	3	2	1	0
FSTT7	FSTT6	FSTT5	FSTT4	FSTT3	FSTT2	FSTT1	FSTT0

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **FSTT0–FSTT15: Fast Startup Input Enable 0 to 15**

0: The corresponding wakeup input has no effect on the PMC.

1: The corresponding wakeup input enables a fast restart signal to the PMC.

- **RTTAL: RTT Alarm Enable**

0: The RTT alarm has no effect on the PMC.

1: The RTT alarm enables a fast restart signal to the PMC.

- **RTCAL: RTC Alarm Enable**

0: The RTC alarm has no effect on the PMC.

1: The RTC alarm enables a fast restart signal to the PMC.

- **USBAL: USB Alarm Enable**

0: The USB alarm has no effect on the PMC.

1: The USB alarm enables a fast restart signal to the PMC.

- **LPM: Low-power Mode**

0: The WaitForInterrupt (WFI) or the WaitForEvent (WFE) instruction of the processor makes the processor enter Sleep mode.

1: The WaitForEvent (WFE) instruction of the processor makes the system to enter Wait mode.

- **FFLPM: Force Flash Low-power Mode**

0: The Flash Low-power mode, defined in the FLPM field, is automatically applied when in Wait mode and released when going back to Active mode.

1: The Flash Low-power mode is user defined by the FLPM field and immediately applied.

- **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

18.20.19 PMC Fast Startup Polarity Register

Name: PMC_FSPR

Address: 0x400E0474

Access: Read/Write

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

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

15	14	13	12	11	10	9	8
FSTP15	FSTP14	FSTP13	FSTP12	FSTP11	FSTP10	FSTP9	FSTP8

7	6	5	4	3	2	1	0
FSTP7	FSTP6	FSTP5	FSTP4	FSTP3	FSTP2	FSTP1	FSTP0

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **FSTPx: Fast Startup Input Polarityx**

Defines the active polarity of the corresponding wakeup input. If the corresponding wakeup input is enabled and at the FSTP level, it enables a fast restart signal.

18.20.20 PMC Fault Output Clear Register

Name: PMC_FOCR

Address: 0x400E0478

Access: Write-only

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

- **FOCLR: Fault Output Clear**

Clears the clock failure detector fault output.

18.20.21 PMC Write Protection Mode Register

Name: PMC_WPMR

Address: 0x400E04E4

Access: Read/Write

31	30	29	28	27	26	25	24
WPKEY							

23	22	21	20	19	18	17	16
WPKEY							

15	14	13	12	11	10	9	8
WPKEY							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WPEN

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x504D43 (“PMC” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x504D43 (“PMC” in ASCII).

See Section 18.19 ”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.

18.20.22 PMC Write Protection Status Register

Name: PMC_WPSR

Address: 0x400E04E8

Access: Read-only

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

23	22	21	20	19	18	17	16
WPVSRC							

15	14	13	12	11	10	9	8
WPVSRC							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WPVS

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of the PMC_WPSR.

1: A write protection violation has occurred since the last read of the PMC_WPSR. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protection Violation Source**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

18.20.23 PMC Peripheral Clock Enable Register 1

Name: PMC_PCER1

Address: 0x400E0500

Access: Write-only

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

23	22	21	20	19	18	17	16
–	–	–	–	–	–	PID49	PID48

15	14	13	12	11	10	9	8
PID47	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **PIDx: Peripheral Clock x Enable**

0: No effect.

1: Enables the corresponding peripheral clock.

Notes:
1. The values for PIDx are defined in the section “Peripheral Identifiers”.
2. Programming the control bits of the Peripheral ID that are not implemented has no effect on the behavior of the PMC.

18.20.24 PMC Peripheral Clock Disable Register 1

Name: PMC_PCDR1

Address: 0x400E0504

Access: Write-only

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

23	22	21	20	19	18	17	16
–	–	–	–	–	–	PID49	PID48

15	14	13	12	11	10	9	8
PID47	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

This register can only be written if the WPEN bit is cleared in the “PMC Write Protection Mode Register” .

- **PIDx: Peripheral Clock x Disable**

0: No effect.

1: Disables the corresponding peripheral clock.

Note: The values for PIDx are defined in the section “Peripheral Identifiers”.

18.20.25 PMC Peripheral Clock Status Register 1

Name: PMC_PCSR1

Address: 0x400E0508

Access: Read-only

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

23	22	21	20	19	18	17	16
–	–	–	–	–	–	PID49	PID48

15	14	13	12	11	10	9	8
PID47	–	–	–	–	–	–	–

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: The values for PIDx are defined in the section “Peripheral Identifiers”.

18.20.26 PMC Peripheral Control Register

Name: PMC_PCR

Address: 0x400E050C

Access: Read/Write

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

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

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

7	6	5	4	3	2	1	0
–	–	PID					

- **PID: Peripheral ID**

Peripheral ID selection from PID2 to PID63.

PID2 to PID63 refer to identifiers as defined in the section “Peripheral Identifiers”.

Not all PID can be configured with divided clock.

Only the following PID can be configured with divided clock: FLEXCOM0-7, PDMIC, TC0-5, ADC.

- **CMD: Command**

0: Read mode.

1: Write mode.

- **DIV: Divisor Value**

Value	Name	Description
0	PERIPH_DIV_MCK	Peripheral clock is MCK
1	PERIPH_DIV2_MCK	Peripheral clock is MCK/2
2	PERIPH_DIV4_MCK	Peripheral clock is MCK/4
3	PERIPH_DIV8_MCK	Peripheral clock is MCK/8

DIV must not be changed while peripheral is in use or when the peripheral clock is enabled.

- **EN: Enable**

0: Selected Peripheral clock is disabled.

1: Selected Peripheral clock is enabled.

18.20.27 PMC Oscillator Calibration Register

Name: PMC_OCR

Address: 0x400E0510

Access: Read/Write

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

23	22	21	20	19	18	17	16
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 the “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.

18.20.28 PMC SleepWalking Enable Register 0

Name: PMC_SLPWK_ER0

Address: 0x400E0514

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 the “PMC Write Protection Mode Register” .

- **PIDx: Peripheral x SleepWalking Enable**

0: No effect.

1: The asynchronous partial wakeup (SleepWalking) function of the corresponding peripheral is enabled.

Not all PIDs can be configured with asynchronous partial wakeup.

Only the following PID can be configured with asynchronous partial wakeup: FLEXCOM0-7, ADC.

The clock of the peripheral must be enabled before using its asynchronous partial wakeup (SleepWalking) function (its associated PIDx field in “PMC Peripheral Clock Status Register 0” is set to ‘1’).

Note: The values for PIDx are defined in section “Peripheral Identifiers”.

18.20.29 PMC SleepWalking Disable Register 0

Name: PMC_SLPWK_DR0

Address: 0x400E0518

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 the “PMC Write Protection Mode Register” .

- **PIDx: Peripheral x SleepWalking Disable**

0: No effect.

1: The asynchronous partial wakeup (SleepWalking) function of the corresponding peripheral is disabled.

Not all PIDs can be configured with asynchronous partial wakeup.

Only the following PIDs can be configured with asynchronous partial wakeup: FLEXCOM0-7, ADC.

Note: The values for PIDx are defined in the section “Peripheral Identifiers”.

18.20.30 PMC SleepWalking Status Register 0

Name: PMC_SLPWK_SR0

Address: 0x400E051C

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 x SleepWalking Status**

0: The asynchronous partial wakeup (SleepWalking) function of the peripheral is currently disabled or the peripheral enabled for asynchronous partial wakeup (SleepWalking) cleared the PIDx bit upon detection of a wakeup condition.

1: The asynchronous partial wakeup (SleepWalking) function of the peripheral is currently enabled.

Not all PIDs can be configured with asynchronous partial wakeup.

Only the following PIDs can be configured with asynchronous partial wakeup: FLEXCOM0-7, ADC.

Note: The values for PIDx are defined in the section “Peripheral Identifiers”.

18.20.31 PMC SleepWalking Activity Status Register 0

Name: PMC_SLPWK_ASR0

Address: 0x400E0520

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 x Activity Status**

0: The peripheral x is not presently active. The asynchronous partial wakeup (SleepWalking) function can be activated.

1: The peripheral x is presently active. The asynchronous partial wakeup (SleepWalking) function must not be activated.

Only the following PIDs can be configured with asynchronous partial wakeup: FLEXCOM0-7, ADC.
All other PIDs are always read at 0.

Note: The values for PIDx are defined in the section “Peripheral Identifiers”.

18.20.32 PLL Maximum Multiplier Value Register

Name: PMC_PMMR

Address: 0x400E0530

Access: Read/Write

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

23	22	21	20	19	18	17	16
PLLB_MMAX							

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

7	6	5	4	3	2	1	0
PLLA_MMAX							

This register can only be written if the WPEN bit is cleared in the "PMC Write Protection Mode Register" .

- **PLLA_MMAX: PLLA Maximum Allowed Multiplier Value**

Defines the maximum value of multiplication factor that can be sent to PLLA. Any value of the MULA field (see "PMC Clock Generator PLLA Register") above PLLA_MMAX is saturated to PLLA_MMAX. PLLA_MMAX write operation is cancelled in the following cases:

- The value of MULA is currently saturated by PLLA_MMAX
- The user is trying to write a value of PLLA_MMAX that is smaller than the current value of MULA

- **PLLB_MMAX: PLLB Maximum Allowed Multiplier Value**

Defines the maximum value of multiplication factor that can be sent to PLLB. Any value of the MULB field (see "PMC Clock Generator PLLB Register") above PLLB_MMAX is saturated to PLLB_MMAX. PLLB_MMAX write operation is cancelled in the following cases:

- The value of MULB is currently saturated by PLLB_MMAX.
- The user is trying to write a value of PLLB_MMAX that is smaller than the current value of MULB.

19. Reset Controller (RSTC)

19.1 Description

The Reset Controller (RSTC), driven by power-on reset (POR) cells, software, external reset pin and peripheral events, handles all the resets of the system without any external components. It reports which reset occurred last.

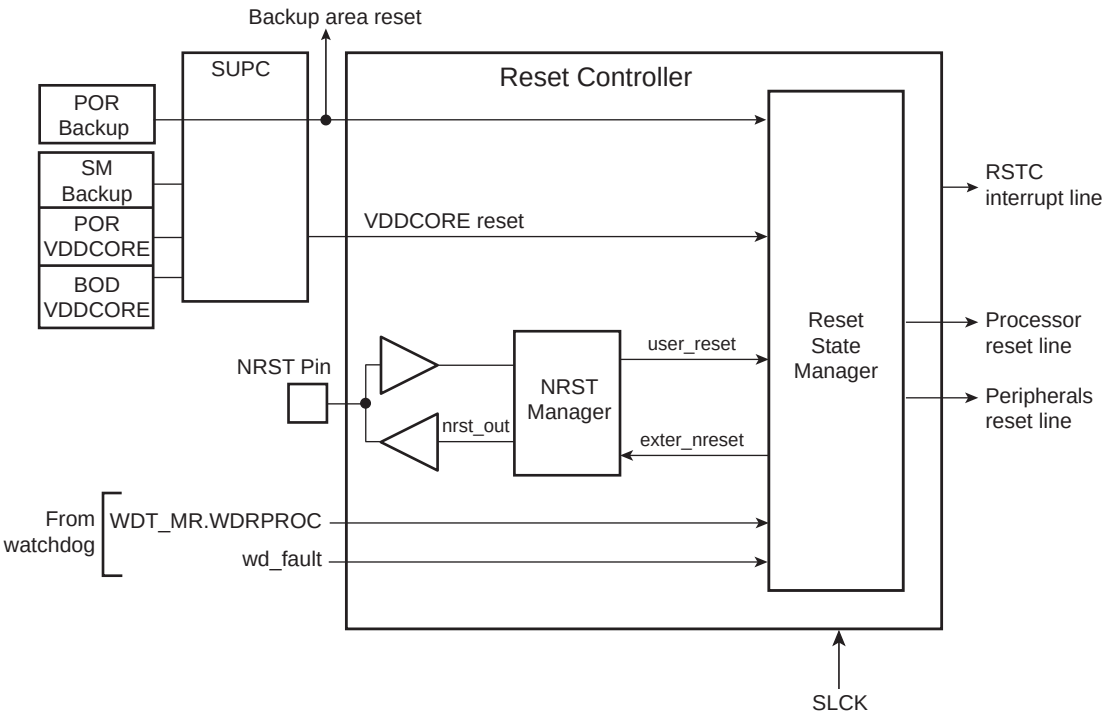
The RSTC also drives independently or simultaneously the external reset and the peripheral and processor resets.

19.2 Embedded Characteristics

- Driven by Embedded Power-on Reset, Software, External Reset Pin and Peripheral Events
- Management of All System Resets, Including
  - External Devices through the NRST Pin
  - Processor
  - Peripheral Set
- Reset Source Status
  - Status of the Last Reset
  - Either VDDCORE and VDDIO POR Reset, Software Reset, User Reset, Watchdog Reset, 32.768 kHz Crystal Oscillator Failure Detection Reset
- External Reset Signal Control and Shaping

19.3 Block Diagram

Figure 19-1. Reset Controller Block Diagram



19.4 Functional Description

19.4.1 Overview

The RSTC is made up of an NRST manager and a reset state manager. It runs at SLCK frequency and generates the following reset signals:

- proc_nreset: Processor reset line (also resets the Watchdog Timer)
- periph_nreset: Affects the whole set of embedded peripherals
- nrst_out: Drives the NRST pin

These reset signals are asserted by the RSTC, either on events generated by peripherals, events on NRST pin, or on software action. The reset state manager controls the generation of reset signals and provides a signal to the NRST manager when an assertion of the NRST pin is required.

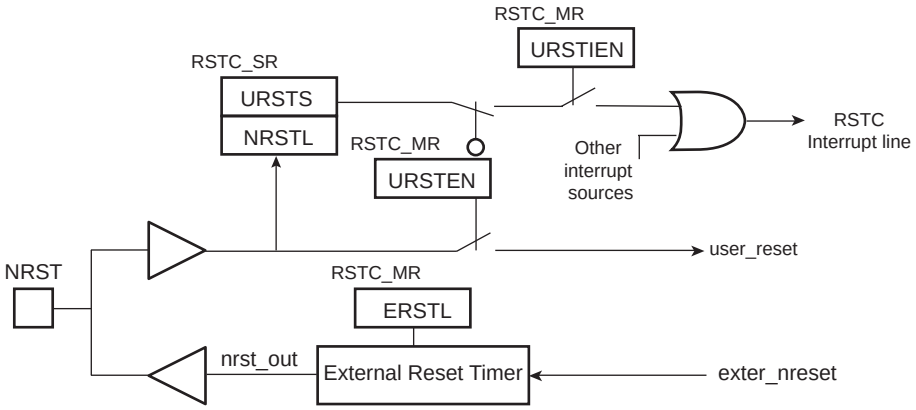
The NRST manager shapes the NRST assertion during a programmable time, thus controlling external device resets.

The RSTC Mode register (RSTC_MR), used to configure the RSTC, is powered with VDDIO, so that its configuration is saved as long as VDDIO is on.

19.4.2 NRST Manager

The NRST manager samples the NRST input pin and drives this pin low when required by the reset state manager. Figure 19-2 shows the block diagram of the NRST manager.

Figure 19-2. NRST Manager



19.4.2.1 NRST Signal or Interrupt

The NRST manager samples the NRST pin at SLCK speed. When the NRST line is low for more than three clock cycles, a User Reset is reported to the reset state manager. The NRST pin must be asserted for at least 1 SLCK clock cycle to ensure execution of a user reset.

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

The level of the pin NRST can be read at any time in the bit NRSTL in the RSTC Status Register (RSTC_SR). As soon as the NRST pin is asserted, RSTC_SR. URSTS is written to ‘1’. This bit is cleared only when the RSTC_SR is read.

The RSTC can also be programmed to generate an interrupt instead of generating a reset. To do so, RSTC_MR.URSTIEN must be set.

19.4.2.2 NRST External Reset Control

The reset state manager asserts the signal exter_nreset to assert the NRST pin. When this occurs, the “nrst_out” signal is driven low by the NRST manager for a time programmed by RSTC_MR.ERSTL. This assertion duration, named External Reset Length, lasts $2^{(ERSTL+1)}$ SLCK cycles. This gives the approximate duration of an assertion between 60 µs and 2 seconds. Note that ERSTL at ‘0’ defines a two-cycle duration for the NRST pulse.

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

RSTC_MR is backed up, making it possible to use the value of ERSTL to shape the system powerup reset for devices requiring a longer startup time than that of the MCU.

19.4.3 Reset States

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

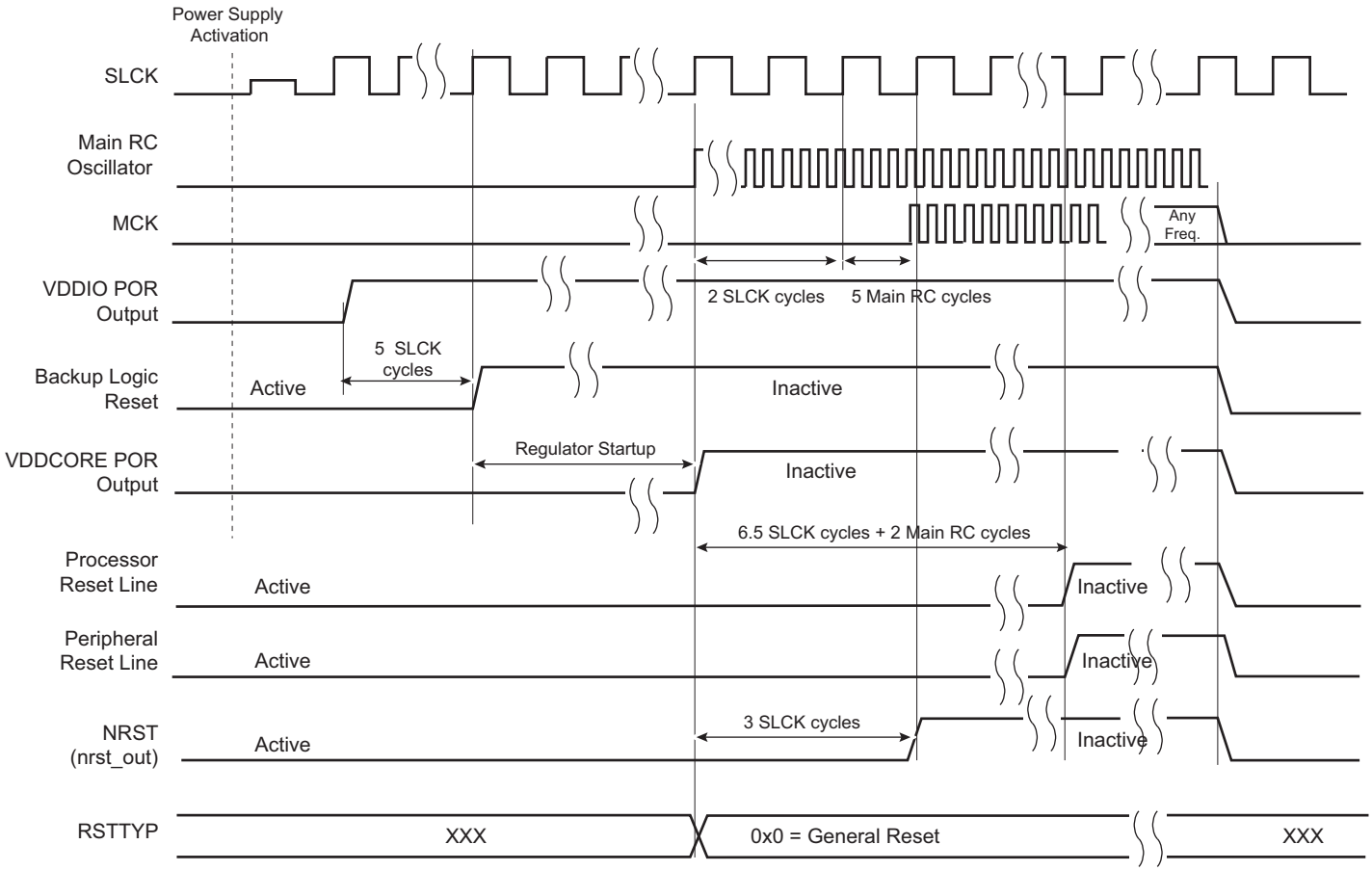
19.4.3.1 General Reset

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 RSTC_SR.RSTTYP reports a general reset. As the RSTC_MR is written to '0', the NRST line rises two cycles after the vddcore_nreset, as ERSTL defaults at value 0x0.

Figure 19-3 shows how the general reset affects the reset signals.

Figure 19-3. General Reset Timing Diagram



19.4.3.2 Backup Reset

A backup reset occurs when the chip exits from Backup mode. While exiting Backup mode, the vddcore_nreset signal is asserted by the Supply Controller.

Field RSTC_SR.RSTTYP is updated to report a backup reset.

19.4.3.3 32.768 kHz Crystal Oscillator Failure Detection Reset

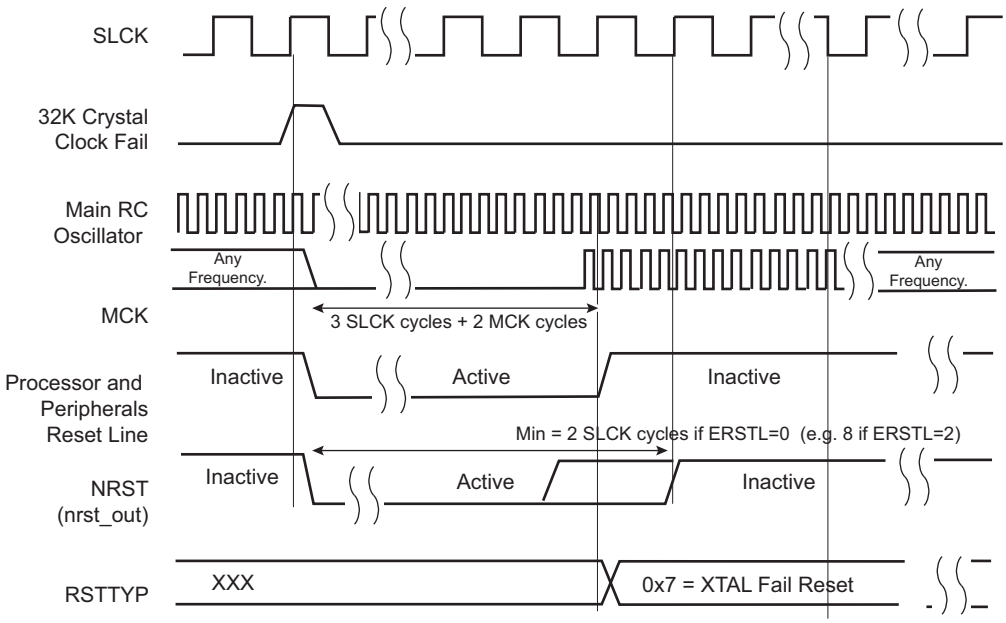
The 32.768 kHz crystal oscillator failure detection reset is done when the 32.768 kHz crystal oscillator frequency monitoring circuitry in the PMC detects a failure and RSTC_MR.SCKSW is written to '1'. This reset lasts three slow clock cycles.

When RSTC_MR.SCKSW is written to ‘0’, the 32.768 kHz crystal oscillator fault has no impact on the RSTC.

During the 32.768 kHz crystal oscillator failure detection reset, the processor reset and the peripheral reset are asserted. The NRST line is also asserted, depending on the value of RSTC_MR.ERSTL.

When the 32.768 kHz crystal oscillator failure generates a VDDCORE reset, PMC_SR.XT32KERR is automatically cleared by the peripheral and core reset.

Figure 19-4. 32.768 kHz Crystal Oscillator Failure Detection Reset Timing Diagram



19.4.3.4 Watchdog Reset

The watchdog reset is entered when a watchdog fault occurs. This reset lasts three SLCK cycles.

When in watchdog reset, assertion of the reset signals depends on the value of WDT_MR.WDRPROC:

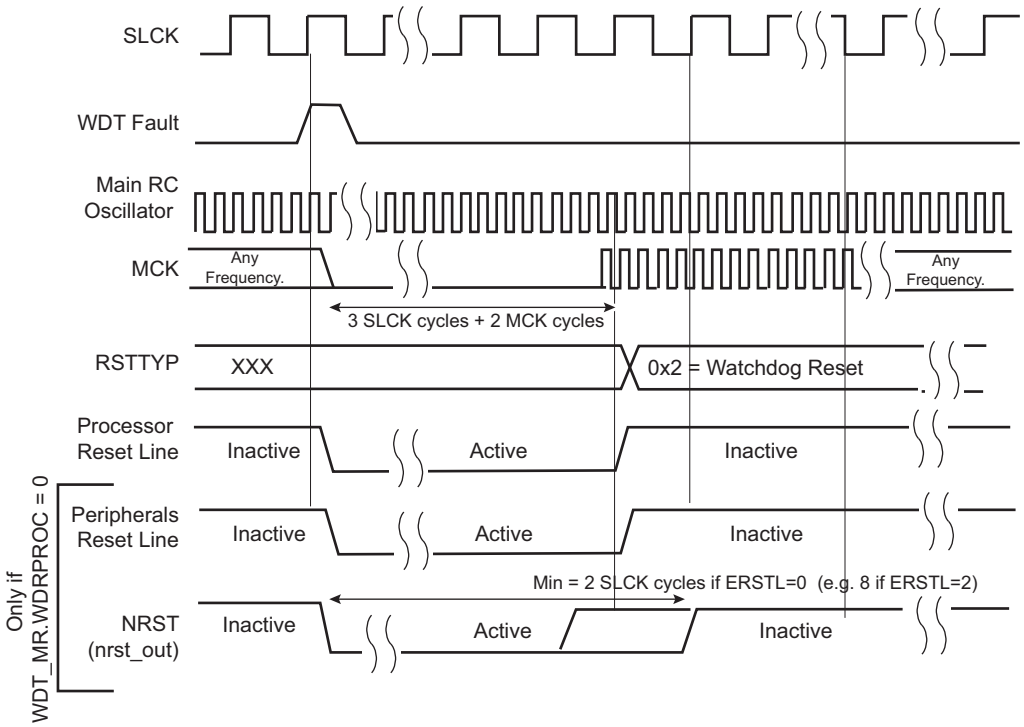
- If WDRPROC = 0, the processor reset and the peripheral reset are asserted. The NRST line is also asserted, depending on how field RSTC_MR.ERSTL is programmed. However, the resulting low level on NRST does not result in a user reset state.
- If WDRPROC = 1, only the processor reset is asserted.

The Watchdog Timer is reset by the proc_nreset signal. As the watchdog fault always causes a processor reset if WDT_MR.WDRSTEN is written to '1', the Watchdog Timer is always reset after a watchdog reset, and the Watchdog is enabled by default and with a period set to a maximum.

When WDT_MR.WDRSTEN is written to '0', the watchdog fault has no impact on the RSTC.

After a watchdog overflow occurs, the report on the RSTC_SR.RSTTYP may differ (either WDT_RST or USER_RST) depending on the external components driving the NRST pin. For example, if the NRST line is driven through a resistor and a capacitor (NRST pin debouncer), the reported value is USER_RST if the low to high transition is greater than one SLCK cycle.

Figure 19-5. Watchdog Reset Timing Diagram



19.4.3.5 Software Reset

The RSTC offers commands to assert the different reset signals. These commands are performed by writing the Control register (RSTC_CR) with the following bits at ‘1’:

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

The software reset is entered if at least one of these bits is written to ‘1’ by the software. All these commands can be performed independently or simultaneously. The software reset lasts three SLCK cycles.

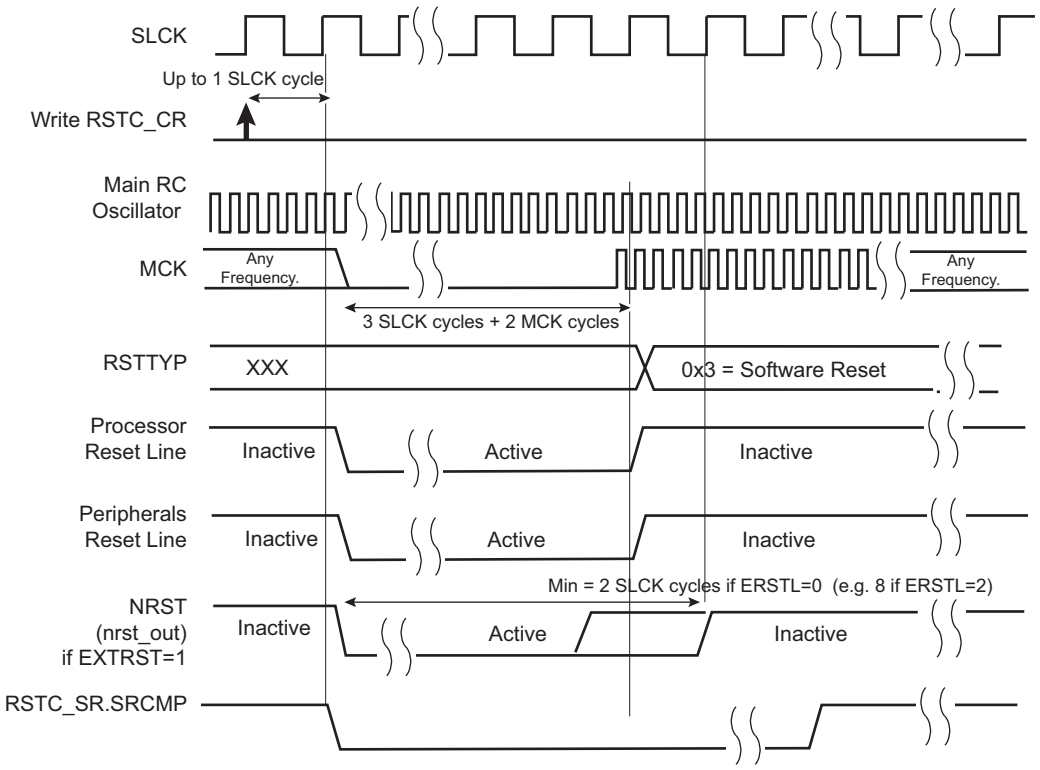
The internal reset signals are asserted as soon as the register write is performed. This is detected on the Master Clock (MCK). They are released when the software reset has ended, i.e., synchronously to SLCK.

If EXTRST is written to ‘1’, the nrst_out signal is asserted depending on the configuration of RSTC_MR.ERSTL. However, the resulting falling edge on NRST does not lead to a user reset.

If and only if the RSTC_CR.PROCRST is written to ‘1’, the RSTC reports the software status in field RSTC_SR.RSTTYP. Other software resets are not reported in RSTTYP.

As soon as a software operation is detected, RSTC_SR.SRCMP is written to ‘1’. SRCMP is cleared at the end of the software reset. No other software reset can be performed while SRCMP is written to ‘1’, and writing any value in the RSTC_CR has no effect.

Figure 19-6. Software Reset Timing Diagram



19.4.3.6 User Reset

A user reset is generated when a low level is detected on the NRST pin and RSTC_MR.URSTEN is at ‘1’. The NRST input signal is resynchronized with SLCK to ensure proper behavior of the system. Thus, the NRST pin must be asserted for at least 1 SLCK clock cycle to ensure execution of a user reset.

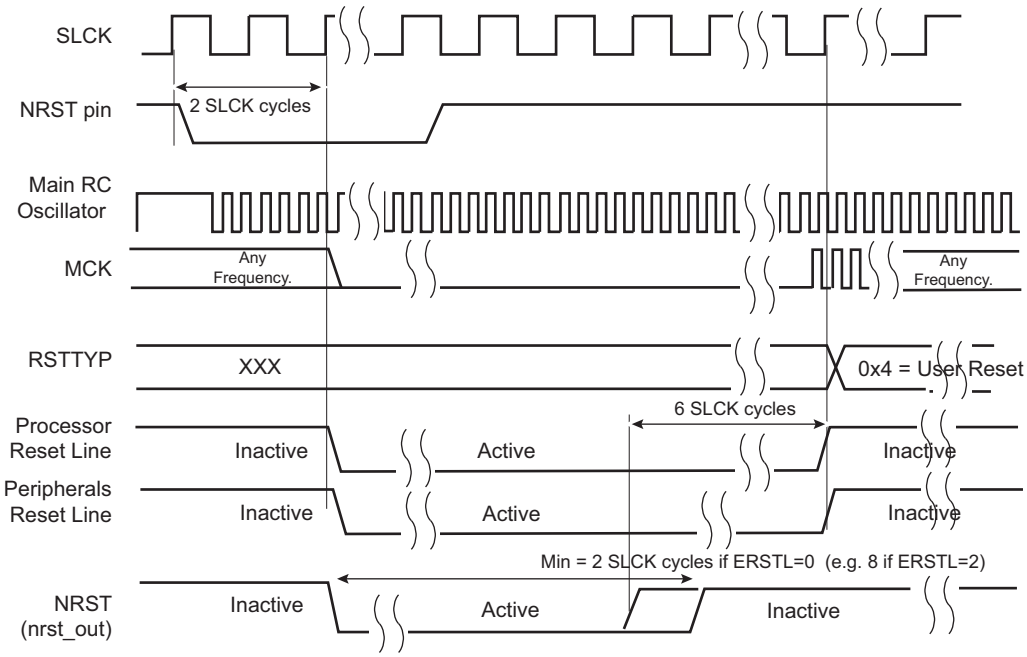
The user reset is triggered 2 SLCK cycles after a low level is detected on NRST. The processor reset and the peripheral reset are asserted.

The user reset ends when NRST rises, after a two-cycle resynchronization time and a three-cycle processor startup. The processor clock is reenabled as soon as NRST is confirmed high.

When the processor reset signal is released, RSTC_SR.RSTTYP is loaded with the value ‘4’, indicating a user reset.

The NRST manager guarantees that the NRST line is asserted for External Reset Length SLCK cycles, as configured in RSTC_MR.ERSTL. However, if NRST does not rise after External Reset Length because it is driven low externally, the internal reset lines remain asserted until NRST actually rises.

Figure 19-7. User Reset Timing Diagram



19.4.4 Reset State Priorities

The reset state manager manages the priorities among the different reset sources. The resets are listed in order of priority as follows:

1. General reset
2. Backup reset
3. 32.768 kHz Crystal Failure Detection reset
4. Watchdog reset
5. Software reset
6. User reset

Specific cases are listed below:

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

19.5 Reset Controller (RSTC) User Interface

Table 19-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	RSTC_CR	Write-only	–
0x04	Status Register	RSTC_SR	Read-only	0x0000_0000[1]
0x08	Mode Register	RSTC_MR	Read/Write	0x0000 0001

Note: 1. This value assumes that a general reset has been performed, subject to change if other types of reset are generated.

19.5.1 RSTC Control Register

Name: RSTC_CR

Address: 0x400E1400

Access: Write-only

31	30	29	28	27	26	25	24
KEY							

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

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

- **PROCRST: Processor Reset**

0: No effect.

1: If KEY = 0xA5, resets the processor.

- **PERRST: Peripheral Reset**

0: No effect.

1: If KEY = 0xA5, resets the peripherals.

- **EXTRST: External Reset**

0: No effect.

1: If KEY = 0xA5, asserts the NRST pin.

- **KEY: System Reset Key**

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

19.5.2 RSTC Status Register

Name: RSTC_SR

Address: 0x400E1404

Access: Read-only

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

- **URSTS: User Reset Status**

A high-to-low transition of the NRST pin sets the URSTS. This transition is also detected on the MCK rising edge. If the user reset is disabled (URSTEN = 0 in RSTC_MR) and if the interrupt is enabled by RSTC_MR.URSTIEN, URSTS triggers an interrupt. Reading the RSTC_SR resets URSTS and clears the interrupt.

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

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

- **RSTTYP: Reset Type**

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

Value	Name	Description
0	GENERAL_RST	First powerup reset
1	BACKUP_RST	Return from Backup mode
2	WDT_RST	Watchdog fault occurred
3	SOFT_RST	Processor reset required by the software
4	USER_RST	NRST pin detected low
5	–	Reserved
6	–	Reserved
7	SLCK_XTAL_RST	32.768 kHz crystal failure detection fault occurred

- **NRSTL: NRST Pin Level**

Registers the NRST pin level sampled on each MCK rising edge.

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

When set, this bit indicates that a software reset command is in progress and that no further software reset should be performed until the end of the current one. This bit is automatically cleared at the end of the current software reset.

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

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

19.5.3 RSTC Mode Register

Name: RSTC_MR

Address: 0x400E1408

Access: Read/Write

31	30	29	28	27	26	25	24
KEY							

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

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

7	6	5	4	3	2	1	0
–	–	–	URSTIEN	–	–	SCKSW	URSTEN

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

- **URSTEN: User Reset Enable**

0: The detection of a low level on the NRST pin does not generate a user reset.

1: The detection of a low level on the NRST pin triggers a user reset.

- **SCKSW: Slow Clock Switching**

0: The detection of a 32.768 kHz crystal failure has no effect.

1: The detection of a 32.768 kHz crystal failure resets the logic supplied by VDDCORE.

- **URSTIEN: User Reset Interrupt Enable**

0: RSTC_SR.USRTS at ‘1’ has no effect on the RSTC interrupt line.

1: RSTC_SR.USRTS at ‘1’ asserts the RSTC interrupt line if URSTEN = 0.

- **ERSTL: External Reset Length**

This field defines the external reset length. The external reset is asserted during a time of $2^{(ERSTL+1)}$ SLCK cycles. This allows assertion duration to be programmed between 60 µs and 2 seconds. Note that synchronization cycles must also be considered when calculating the actual reset length as previously described.

- **KEY: Write Access Password**

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

20. Watchdog Timer (WDT)

20.1 Description

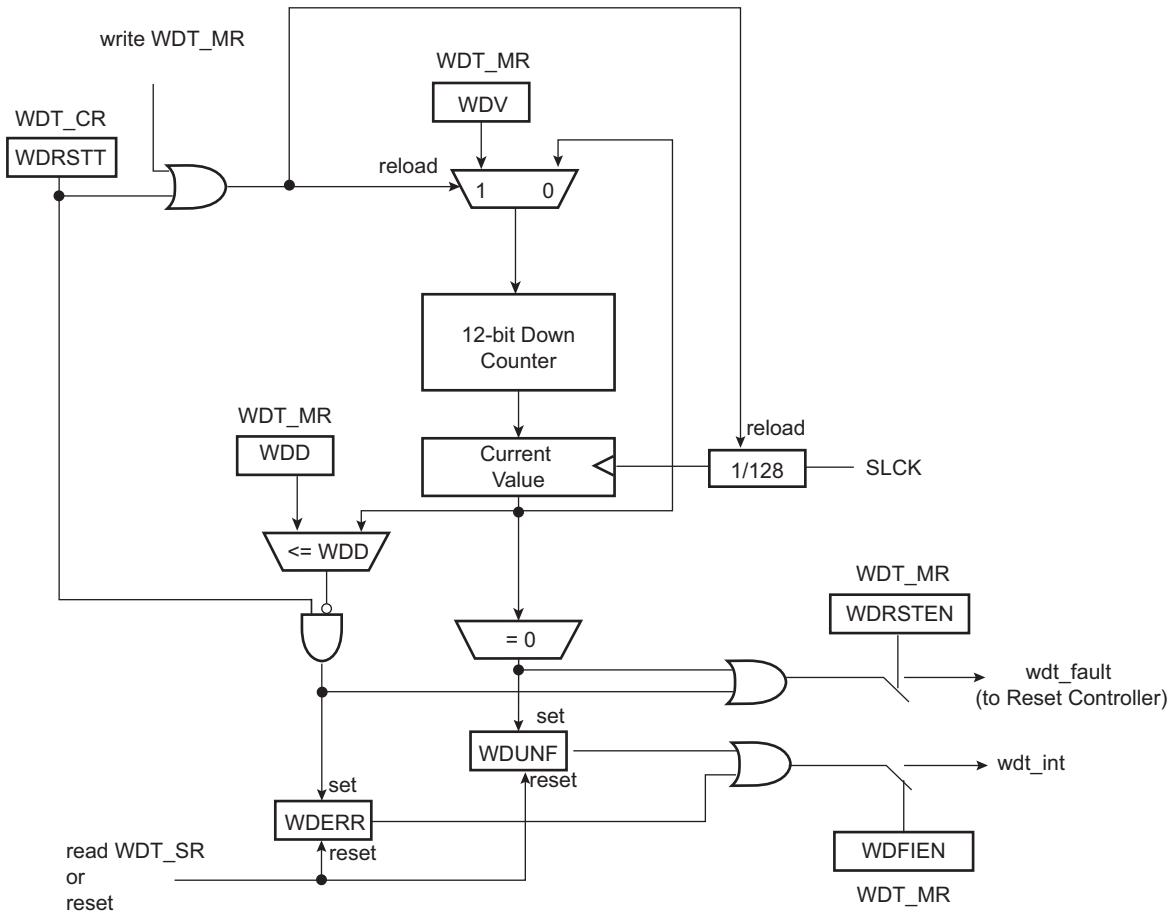
The Watchdog Timer (WDT) is used to prevent system lock-up if the software becomes trapped in a deadlock. It features a 12-bit down counter that allows a watchdog period of up to 16 seconds (slow clock around 32 kHz). It can generate a general reset or a processor reset only. In addition, it can be stopped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

20.2 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

20.3 Block Diagram

Figure 20-1. Watchdog Timer Block Diagram



20.4 Functional Description

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

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

After a processor reset, the value of WDV is 0xFFF, corresponding to the maximum value of the counter with the external reset generation enabled (field WDRSTEN at 1 after a backup reset). This means that a default watchdog is running at reset, i.e., at power-up. The user can either disable the WDT by setting bit WDT_MR.WDDIS or reprogram the WDT to meet the maximum watchdog period the application requires.

When setting the WDDIS bit, and while it is set, the fields WDV and WDD must not be modified.

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

WDT_MR can be written only once. Only a processor reset resets it. Writing WDT_MR reloads the timer with the newly programmed mode parameters.

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

The reload of the watchdog must occur while the watchdog counter is within a window between 0 and WDD. WDD is defined in WDT_MR.

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

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

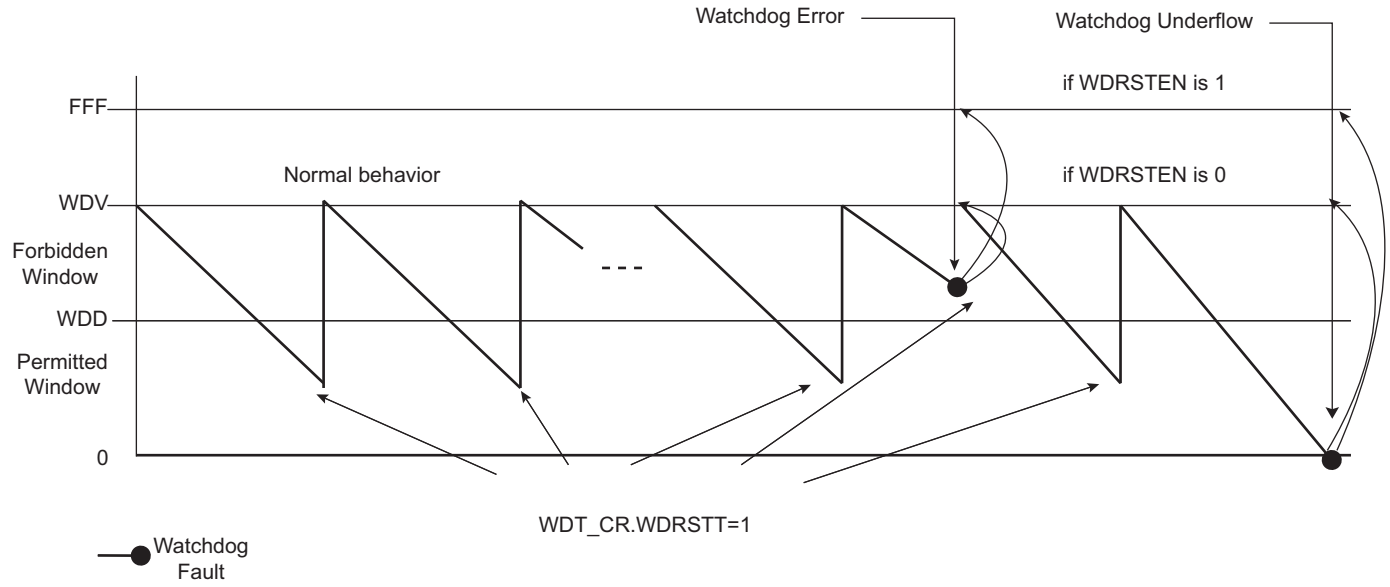
The status bits WDUNF (Watchdog Underflow) and WDERR (Watchdog Error) trigger an interrupt, provided the bit WDT_MR.WDFIEN is set. The signal “wdt_fault” to the Reset Controller causes a watchdog reset if the WDRSTEN bit is set as already explained in the Reset Controller documentation. In this case, the processor and the Watchdog Timer are reset, and the WDERR and WDUNF flags are reset.

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

Writing WDT_MR reloads and restarts the down counter.

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

Figure 20-2. Watchdog Behavior



20.5 Watchdog Timer (WDT) User Interface

Table 20-1. 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

20.5.1 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

Note: The WDT_CR register values must not be modified within three slow clock periods following a restart of the watchdog performed by a write access in WDT_CR. Any modification will cause the watchdog to trigger an end of period earlier than expected.

- **WDRSTT: Watchdog Restart**

0: No effect.

1: Restarts the watchdog if KEY is written to 0xA5.

- **KEY: Password**

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

20.5.2 Watchdog Timer Mode Register

Name: WDT_MR

Address: 0x400E1454

Access: Read/Write Once

31	30	29	28	27	26	25	24
–	–	WDIDLEHLT	WDDBGHLT	WDD			
23	22	21	20	19	18	17	16
WDD							
15	14	13	12	11	10	9	8
WDDIS	WDRPROC	WDRSTEN	WDFIEN	WDV			
7	6	5	4	3	2	1	0
WDV							

Notes:
1. The first write access prevents any further modification of the value of this register. Read accesses remain possible.
2. The WDT_MR register values must not be modified within three slow clock periods following a restart of the watchdog performed by a write access in WDT_CR. Any modification will cause the watchdog to trigger an end of period earlier than expected.

- **WDV: Watchdog Counter Value**

Defines the value loaded in the 12-bit watchdog counter.

- **WDFIEN: Watchdog Fault Interrupt Enable**

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

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

- **WDRSTEN: Watchdog Reset Enable**

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

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

- **WDRPROC: Watchdog Reset Processor**

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

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

- **WDDIS: Watchdog Disable**

0: Enables the Watchdog Timer.

1: Disables the Watchdog Timer.

Note: When setting the WDDIS bit, and while it is set, the fields WDV and WDD must not be modified.

- **WDD: Watchdog Delta Value**

Defines the permitted range for reloading the Watchdog Timer.

If the Watchdog Timer value is less than or equal to WDD, setting bit WDT_CR.WDRSTT restarts the timer.

If the Watchdog Timer value is greater than WDD, setting bit WDT_CR.WDRSTT causes a watchdog error.

- **WDDBGHLT: Watchdog Debug Halt**

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

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

- **WDIDLEHLT: Watchdog Idle Halt**

0: The watchdog runs when the system is in idle state.

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

20.5.3 Watchdog Timer Status Register

Name: WDT_SR

Address: 0x400E1458

Access Read-only

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

- **WDUNF: Watchdog Underflow (cleared on read)**

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

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

- **WDERR: Watchdog Error (cleared on read)**

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

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

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
- Automatic Circular Buffer Mode
- Transfer Bus Error Report

21.3 Peripheral DMA Controller Channels

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

Table 21-1. Peripheral DMA Controller

Instance Name	Channel T/R	Channel Number
USART7	Transmit	29
SPI7	Transmit	29
TWI7	Transmit	29
MEM2MEM	Transmit	28
SPI5	Transmit	27
USART5	Transmit	27
TWI5	Transmit	27
TWI4	Transmit	26
USART4	Transmit	26
SPI4	Transmit	26
TWI6	Transmit	25
USART6	Transmit	25
SPI6	Transmit	25
USART0	Transmit	24
TWI0	Transmit	24

Table 21-1. Peripheral DMA Controller (Continued)

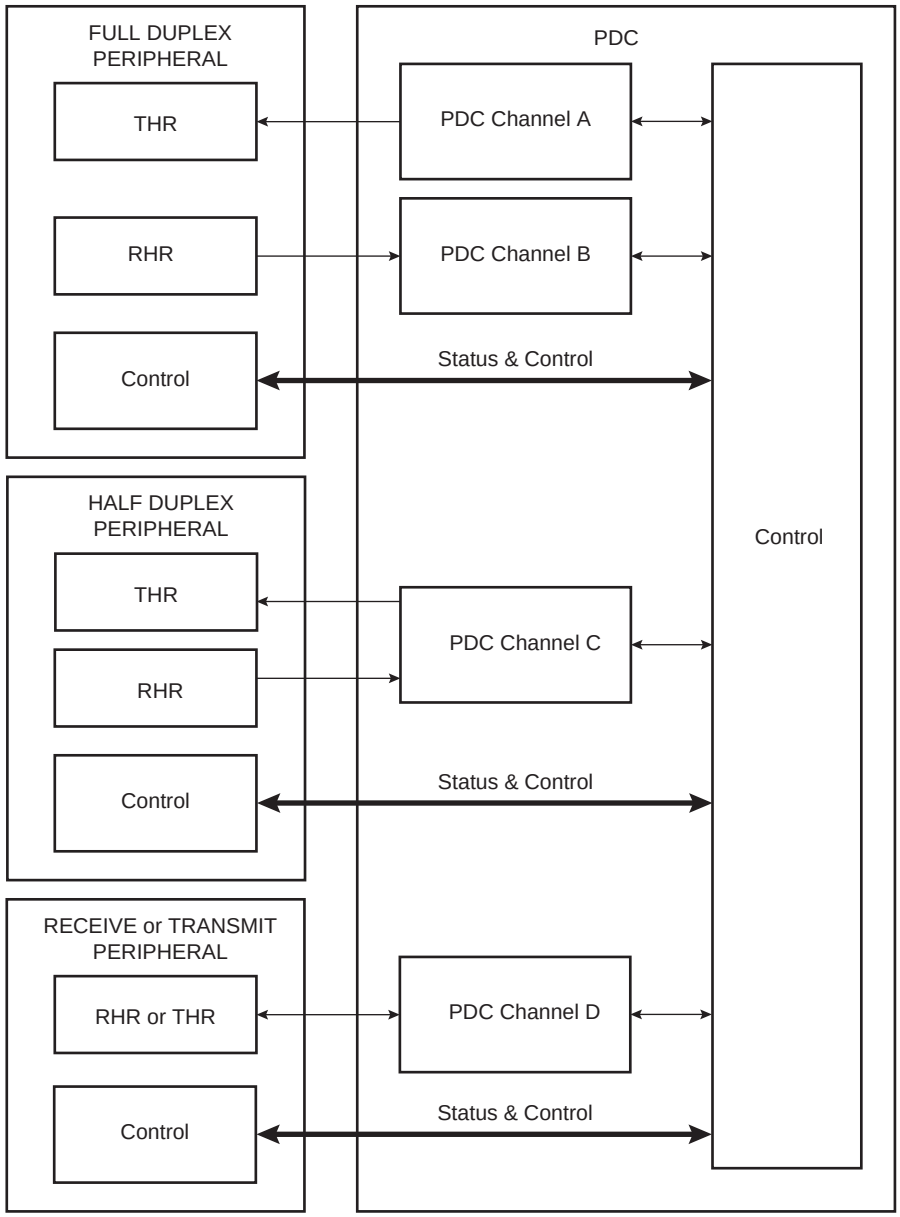
Instance Name	Channel T/R	Channel Number
SPI0	Transmit	24
USART1	Transmit	23
TWI1	Transmit	23
SPI1	Transmit	23
USART2	Transmit	22
TWI2	Transmit	22
SPI2	Transmit	22
I2SC1	Transmit	21, 20
I2SC0	Transmit	19, 18
USART3	Transmit	17
SPI3	Transmit	17
TWI3	Transmit	17
USART7	Receive	16
SPI7	Receive	16
TWI7	Receive	16
MEM2MEM	Receive	15
TC0::TC0	Receive	14
SPI5	Receive	13
USART5	Receive	13
TWI5	Receive	13
TWI4	Receive	12
USART4	Receive	12
SPI4	Receive	12
TWI6	Receive	11
USART6	Receive	11
SPI6	Receive	11
USART0	Receive	10
TWI0	Receive	10
SPI0	Receive	10
USART1	Receive	9
TWI1	Receive	9
SPI1	Receive	9
USART2	Receive	8
TWI2	Receive	8
SPI2	Receive	8
PDMIC1	Receive	7
PDMIC0	Receive	6

Table 21-1. Peripheral DMA Controller (Continued)

Instance Name	Channel T/R	Channel Number
I2SC1	Receive	4,5
I2SC0	Receive	2,3
ADC	Receive	1
USART3	Receive	0
SPI3	Receive	0
TWI3	Receive	0

21.4 Block Diagram

Figure 21-1. Block Diagram



21.5 Functional Description

21.5.1 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 21.5.3 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.

21.5.2 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.

21.5.3 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. When the Circular buffer mode is activated, the register set {next counter, next pointer} is not reset when the next counter value is copied to the current counter, both next and current registers must be written to the same value. At the end of this transfer, the PDC channel sets the appropriate flags in the Peripheral Status register.

The following list gives an overview of how status register flags behave depending on the counters' values:

- ENDRX flag is set when the PDC Receive Counter Register (PERIPH_RCR) reaches zero.
- RXBUFF flag is set when both PERIPH_RCR and the PDC Receive Next Counter Register (PERIPH_RNCR) reach zero.
- ENDTX flag is set when the PDC Transmit Counter Register (PERIPH_TCR) reaches zero.
- TXBUFE flag is set when both PERIPH_TCR and the PDC Transmit Next Counter Register (PERIPH_TNCR) reach zero.

These status flags are described in the Transfer Status Register (PERIPH_PTSR).

21.5.4 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.

In case of invalid memory address resulting from a badly programmed PDC transmit or receive pointer, the bus matrix does not perform the requested access and signals a bus error to the PDC. This transfer bus error drives the PERIPH_PTSR.ERR bit high to flag the error in the Transfer Status Register.

21.5.5 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.

21.5.5.1 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.

21.5.5.2 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.

21.5.5.3 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.

21.5.5.4 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.

21.6 Peripheral DMA Controller (PDC) User Interface

Table 21-2. Register Mapping

Offset	Register	Name[1]	Access	Reset
0x00	Receive Pointer Register	PERIPH_RPR	Read/Write	0
0x04	Receive Counter Register	PERIPH_RCR	Read/Write	0
0x08	Transmit Pointer Register	PERIPH_TPR	Read/Write	0
0x0C	Transmit Counter Register	PERIPH_TCR	Read/Write	0
0x10	Receive Next Pointer Register	PERIPH_RNPR	Read/Write	0
0x14	Receive Next Counter Register	PERIPH_RNCR	Read/Write	0
0x18	Transmit Next Pointer Register	PERIPH_TNPR	Read/Write	0
0x1C	Transmit Next Counter Register	PERIPH_TNCR	Read/Write	0
0x20	Transfer Control Register	PERIPH_PTCR	Write-only	–
0x24	Transfer Status Register	PERIPH_PTSR	Read-only	0

Note: 1. PERIPH: Ten registers are mapped in the peripheral memory space at the same offset. These can be configured by the user depending on the function and the desired peripheral.

21.6.1 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.

21.6.2 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.

21.6.3 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.

21.6.4 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.

21.6.5 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.

21.6.6 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.

21.6.7 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.

21.6.8 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.

21.6.9 Transfer Control Register

Name: PERIPH_PTCR

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	ERRCLR
23	22	21	20	19	18	17	16
–	–	–	–	TXCBDIS	TXCBEN	RXCBDIS	RXCBEN
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.

- **RXCBEN: Receiver Circular Buffer Enable**

0: No effect.

1: Enables the PDC circular buffer for the receiver operation.

- **RXCBDIS: Receiver Circular Buffer Disable**

0: No effect.

1: Disables the PDC circular buffer for the receiver operation.

- **TXCBEN: Transmitter Circular Buffer Enable**

0: No effect.

1: Enables the PDC circular buffer for the transmitter operation.

- **TXCBDIS: Transmitter Circular Buffer Disable**

0: No effect.

1: Enables the PDC circular buffer for the transmitter operation.

- **ERRCLR: Transfer Bus Error Clear**

0: No effect.

1: Clears the transfer bus error status bit.

21.6.10 Transfer Status Register

Name: PERIPH_PTSR

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	ERR
23	22	21	20	19	18	17	16
–	–	–	–	–	TXCBEN	–	RXCBEN
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.

- **RXCBEN: Receiver Circular Buffer Enable**

0: PDC Receiver circular buffer mode is disabled.

1: PDC Receiver circular buffer mode is enabled.

- **TXCBEN: Transmitter Circular Buffer Enable**

0: PDC Transmitter circular buffer mode is disabled.

1: PDC Transmitter circular buffer mode is enabled.

- **ERR: Transfer Bus Error**

0: PDC accesses are performed on valid memory address since the last write of ERRCLR bit in PERIPH_PTCR.

1: PDC transmit or receive pointer (or next pointer) is programmed with an invalid memory address since the last write of ERRCLR bit in PERIPH_PTCR.

22. Memory to Memory (MEM2MEM)

22.1 Description

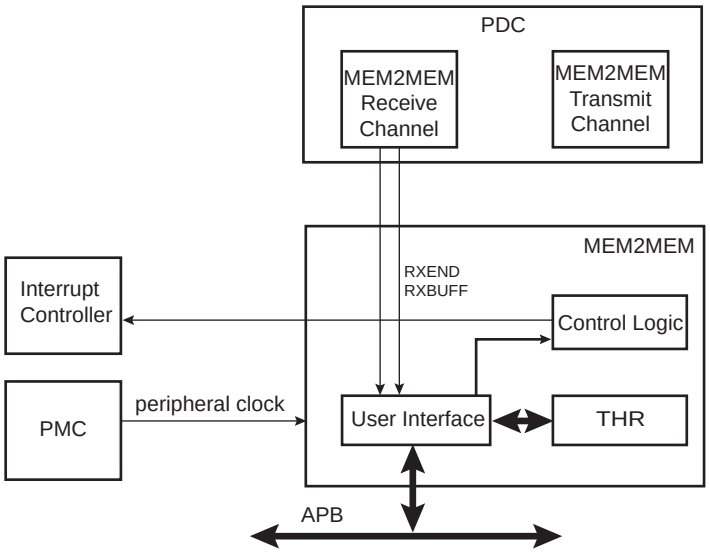
The Memory to Memory (MEM2MEM) module allows the Peripheral DMA Controller (PDC) to perform memory to memory transfer without CPU intervention. The transfer size can be configured in byte, half-word or word. Two PDC channels are required to perform the transfer; one channel defines the source of the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

22.2 Embedded Characteristics

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

22.3 Block Diagram

Figure 22-1. Memory to Memory Block Diagram



22.4 Product Dependencies

22.4.1 Power Management

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

22.4.2 Interrupt Sources

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 22-1. Peripheral IDs

Instance	ID
MEM2MEM	15

22.5 Functional Description

The memory to memory transfer requires two 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 two PDC channels exchange data through the Memory to Memory Transfer Holding Register (MEM2MEM_THR) 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 the Memory to Memory Mode Register (MEM2MEM_MR). Word transfer (32-bit) is the default size.

The transfer ends when either RXEND rises and/or RXBUFF rises in the Memory to Memory Interrupt Status Register (MEM2MEM_ISR).

An interrupt can be triggered at the end of transfer by configuring the Memory to Memory Interrupt Enable Register (MEM2MEM_IER). Refer to the PDC section of the product datasheet for detailed information.

22.6 Memory to Memory (MEM2MEM) User Interface

Table 22-2. 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	0x0000_0000
0x18–0xFC	Reserved	–	–	–
0x100–0x124	Reserved for PDC Registers	–	–	–

22.6.1 Memory to Memory Transfer Holding Register

Name: MEM2MEM_THR

Address: 0x40028000

Access: 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.

22.6.2 Memory to Memory Mode Register

Name: MEM2MEM_MR

Address: 0x40028004

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
–	–	–	–	–	–	TSIZE	

- **TSIZE: Transfer Size**

Value	Name	Description
0x0	T_8BIT	The buffer size is defined in byte.
0x1	T_16BIT	The buffer size is defined in half-word (16-bit).
0x2	T_32BIT	The buffer size is defined in word (32-bit). Default value.

22.6.3 Memory to Memory Interrupt Enable Register

Name: MEM2MEM_IER

Address: 0x40028008

Access: Write-only

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

- **RXEND: End of Transfer Interrupt Enable**

0: No effect

1: Enables the corresponding interrupt.

- **RXBUFF: Buffer Full Interrupt Enable**

0: No effect

1: Enables the corresponding interrupt.

22.6.4 Memory to Memory Interrupt Disable Register

Name: MEM2MEM_IDR

Address: 0x4002800C

Access: Write-only

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

- **RXEND: End of Transfer Interrupt Disable**

0: No effect

1: Disables the corresponding interrupt.

- **RXBUFF: Buffer Full Interrupt Disable**

0: No effect

1: Disables the corresponding interrupt.

22.6.5 Memory to Memory Interrupt Mask Register

Name: MEM2MEM_IMR

Address: 0x40028010

Access: Read-only

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

- **RXEND: End of Transfer Interrupt Mask**

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

- **RXBUFF: Buffer Full Interrupt Mask**

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

22.6.6 Memory to Memory Interrupt Status Register

Name: MEM2MEM_ISR

Address: 0x40028014

Access: Read-only

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

- **RXEND: End of Transfer**

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

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

- **RXBUFF: Buffer Full**

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

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

23. Enhanced Embedded Flash Controller (EEFC)

23.1 Description

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

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

23.2 Embedded Characteristics

- Increases Performance in Thumb-2 Mode with 128-bit or 64-bit-wide Memory Interface up to 96 MHz
- Code Loop Optimization
- 64 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 Page
- Provides Unique Identifier
- Provides 512-byte User Signature Area
- Supports Erasing before Programming
- Locking and Unlocking Operations
- Supports Read of the Calibration Bits
- Register Write Protection

23.3 Product Dependencies

23.3.1 Power Management

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

23.3.2 Interrupt Sources

The EEFC interrupt line is connected to the interrupt controller. Using the EEFC interrupt requires the interrupt controller to be programmed first. The EEFC interrupt is generated only if the value of EEFC_FMR.FRDY is ‘1’.

Table 23-1. Peripheral IDs

Instance	ID
EFC	6

23.4 Functional Description

23.4.1 Embedded Flash Organization

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

- One memory plane organized in several pages of the same size for the code
- A separate 2 x 512-byte memory area which includes the unique chip identifier
- A separate 512-byte memory area for the user signature
- Two 128-bit or 64-bit read buffers used for code read optimization
- One 128-bit or 64-bit read buffer used for data read optimization
- One write buffer that manages page programming. The write buffer size is equal to the page size. This buffer is write-only and accessible all along the 1 Mbyte address space, so that each word can be written to its final address.
- Several lock bits used to protect write/erase operation on several pages (lock region). A lock bit is associated with a lock region composed of several pages in the memory plane.
- Several bits that may be set and cleared through the EEFC interface, called general-purpose non-volatile memory bits (GPNVM bits)

The embedded Flash size, the page size, the organization of lock regions and the definition of GPNVM bits are specific to the device. The EEFC returns a descriptor of the Flash controller after a ‘Get Flash Descriptor’ command has been issued by the application (see Section 23.4.3.1 ”Get Flash Descriptor Command”).

Figure 23-1. Flash Memory Areas

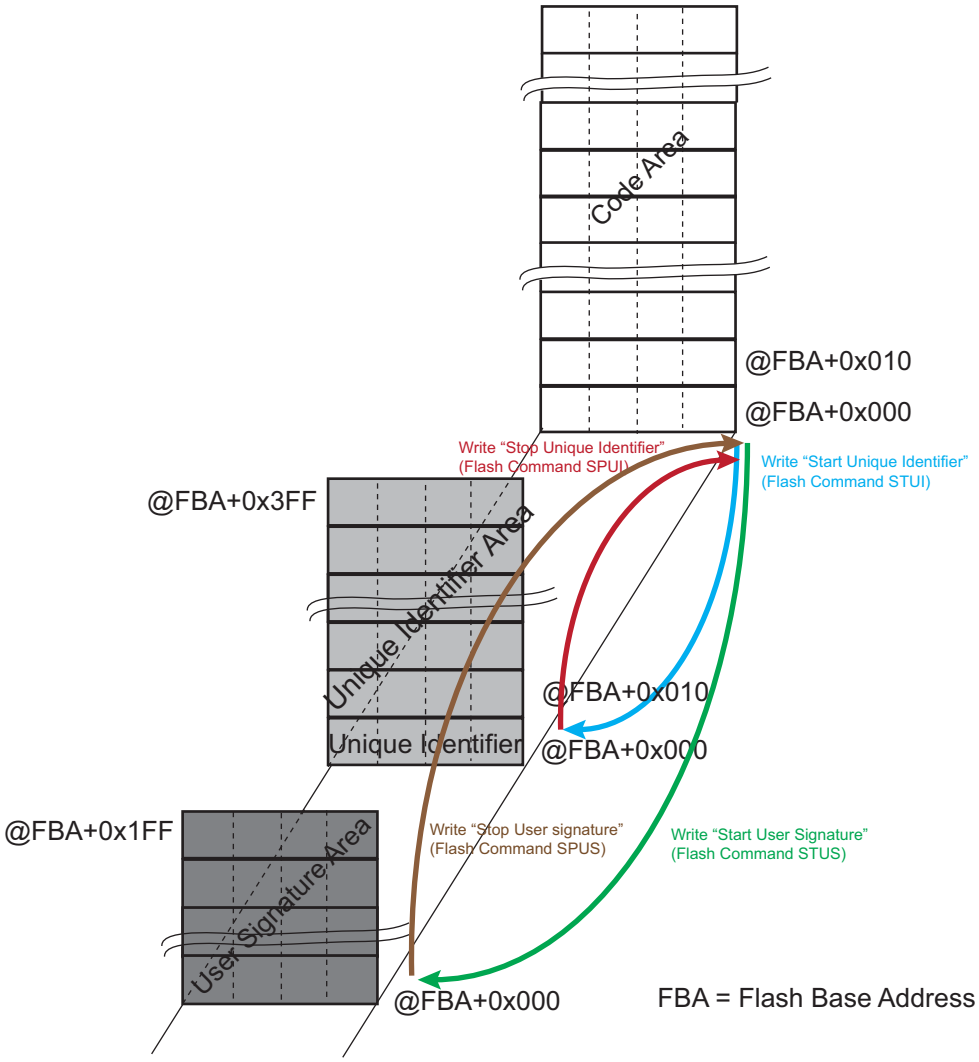
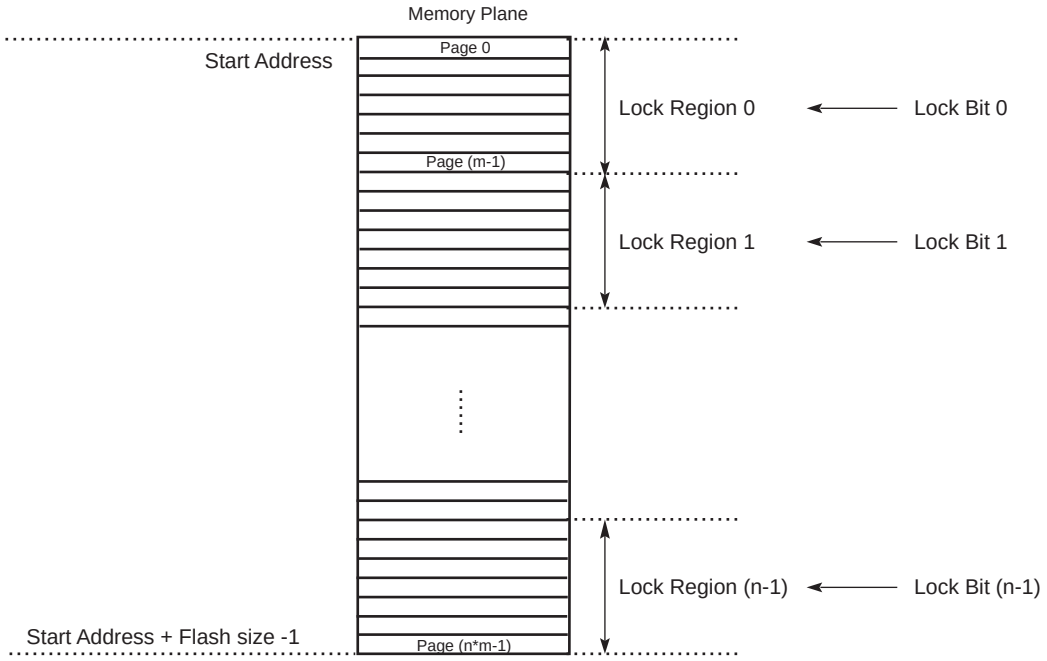


Figure 23-2. Organization of Embedded Flash for Code



23.4.2 Read Operations

An optimized controller manages embedded Flash reads, thus increasing performance when the processor is running in Thumb-2 mode by means of the 128- or 64-bit-wide memory interface.

The Flash memory is accessible through 8-, 16- and 32-bit reads.

As the Flash block size is smaller than the address space reserved for the internal memory area, the embedded Flash wraps around the address space and appears to be repeated within it.

The read operations can be performed with or without wait states. Wait states must be programmed in the field FWS in the Flash Mode register (EEFC_FMR). Defining FWS as 0 enables the single-cycle access of the embedded Flash. For more details, refer to the section “Electrical Characteristics” of this datasheet.

23.4.2.1 128- or 64-bit Access Mode

By default, the read accesses of the Flash are performed through a 128-bit wide memory interface. It improves system performance especially when two or three wait states are needed.

For systems requiring only 1 wait state, or to focus on current consumption rather than performance, the user can select a 64-bit wide memory access via the bit EEFC_FMR.FAM.

For more details, refer to the section “Electrical Characteristics” of this datasheet.

23.4.2.2 Code Read Optimization

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

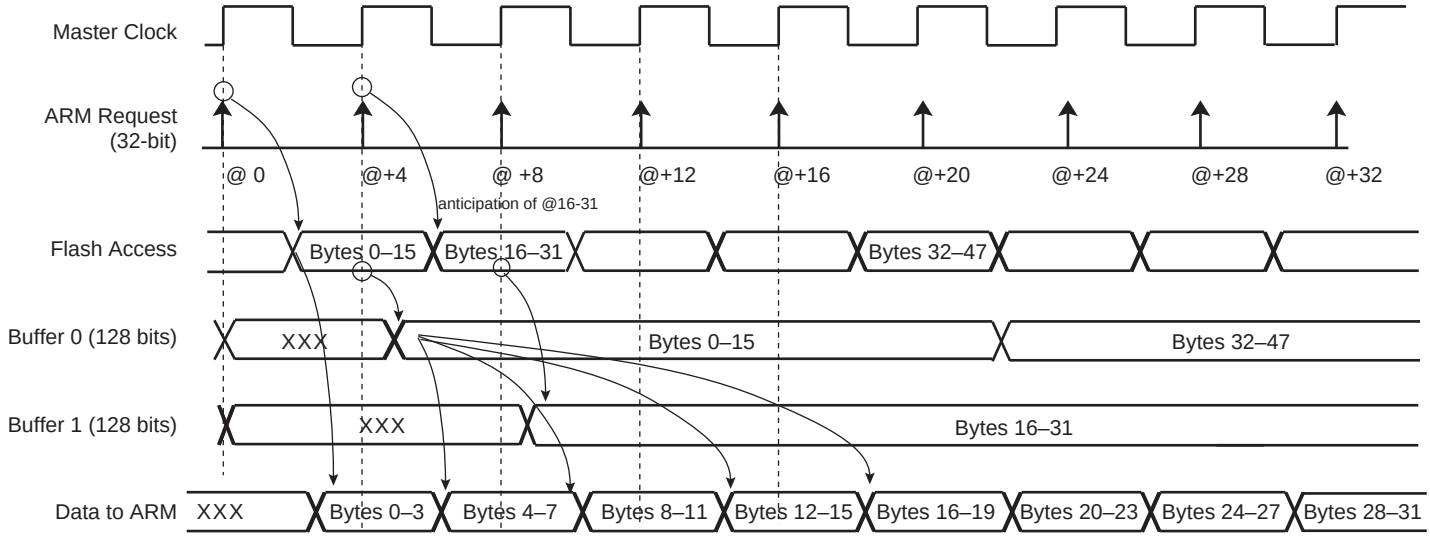
A system of 2 x 128-bit or 2 x 64-bit buffers is added in order to optimize sequential code fetch.

Note: Immediate consecutive code read accesses are not mandatory to benefit from this optimization.

The sequential code read optimization is enabled by default. If the bit EEFC_FMR.SCOD is set, these buffers are disabled and the sequential code read is no longer optimized.

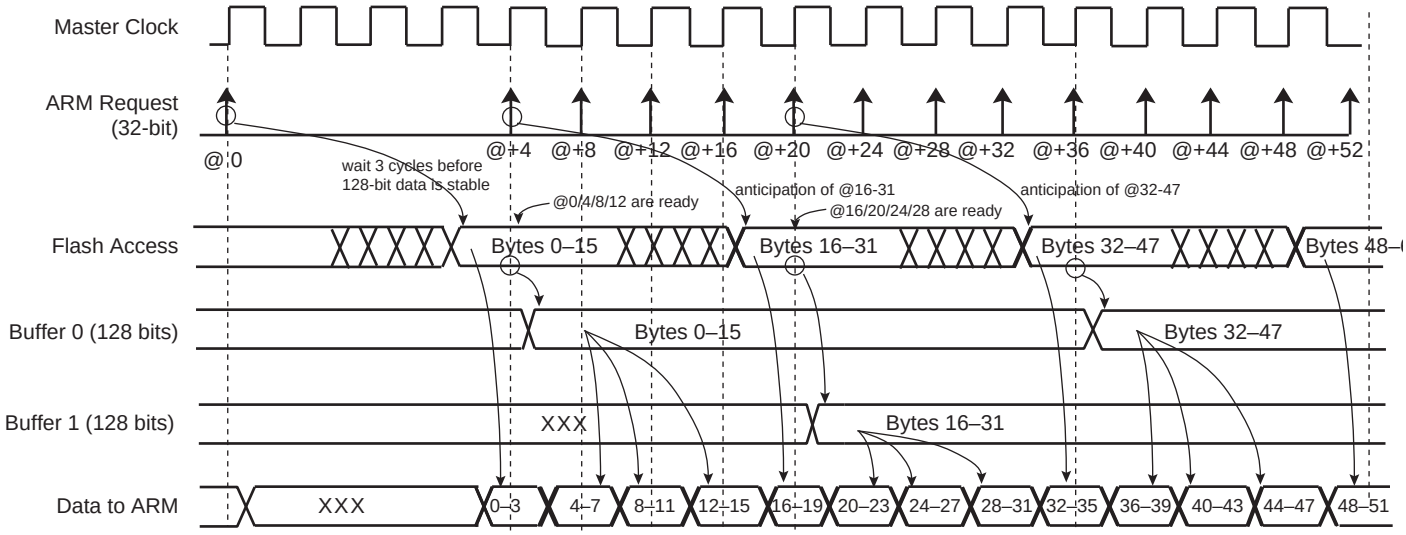
Another system of 2 x 128-bit or 2 x 64-bit buffers is added in order to optimize loop code fetch. Refer to Section 23.4.2.3 ”Code Loop Optimization” for more details.

Figure 23-3. Code Read Optimization for FWS = 0



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

Figure 23-4. Code Read Optimization for FWS = 3



Note: When FWS is between 1 and 3, in case of sequential reads, the first access takes (FWS + 1) cycles. The following accesses take only one cycle.

23.4.2.3 Code Loop Optimization

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

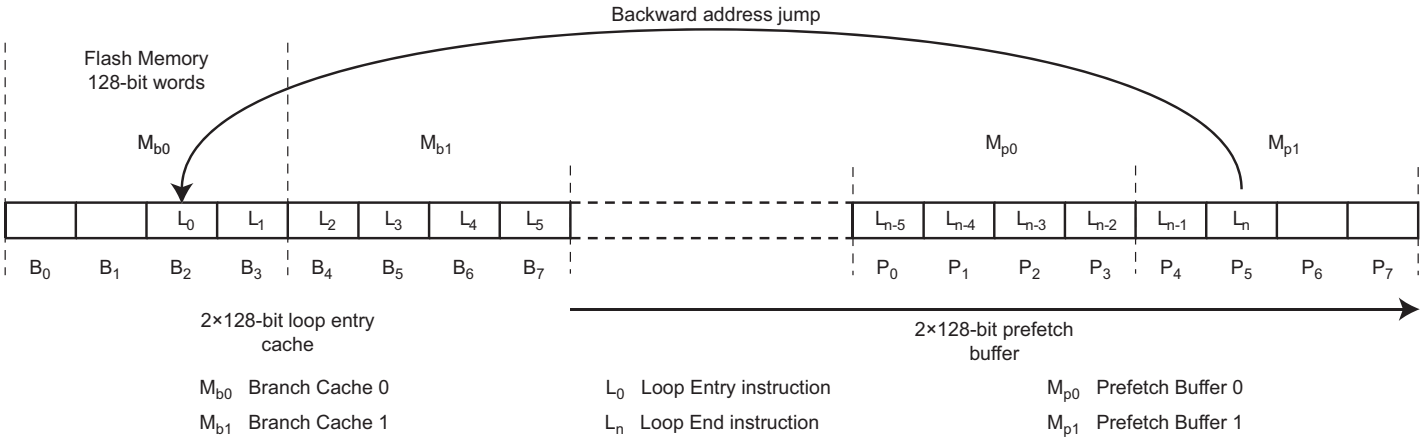
When a backward jump is inserted in the code, the pipeline of the sequential optimization is broken and becomes inefficient. In this case, the loop code read optimization takes over from the sequential code read optimization to prevent the insertion of wait states. The loop code read optimization is enabled by default. In EEFC_FMR, if the bit CLOE is reset to 0 or the bit SCOD is set, these buffers are disabled and the loop code read is not optimized.

When code loop optimization is enabled, if inner loop body instructions L_0 to L_n are positioned from the 128-bit Flash memory cell M_{b0} to the memory cell M_{p1}, after recognition of a first backward branch, the first two Flash memory cells M_{b0} and M_{b1} targeted by this branch are cached for fast access from the processor at the next loop iteration.

Then by combining the sequential prefetch (described in Section 23.4.2.2 ”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 23-5 illustrates code loop optimization.

Figure 23-5. Code Loop Optimization

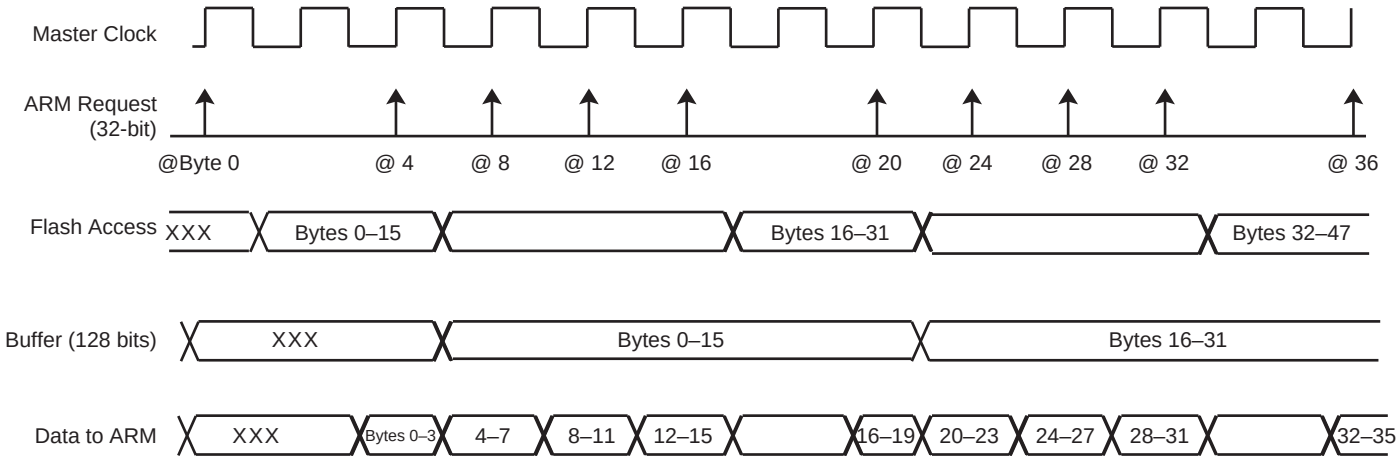


23.4.2.4 Data Read Optimization

The organization of the Flash in 128 bits or 64 bits is associated with two 128-bit or 64-bit prefetch buffers and one 128-bit or 64-bit data read buffer, thus providing maximum system performance. This buffer is added in order to store the requested data plus all the data contained in the 128-bit or 64-bit aligned data. This speeds up sequential data reads if, for example, FWS is equal to 1 (see Figure 23-6). The data read optimization is enabled by default. If the bit EEFC_FMR.SCOD is set, this buffer is disabled and the data read is no longer optimized.

Note: No consecutive data read accesses are mandatory to benefit from this optimization.

Figure 23-6. Data Read Optimization for FWS = 1



23.4.3 Flash Commands

The EEFC offers a set of commands to manage programming the Flash memory, locking and unlocking lock regions, consecutive programming, locking and full Flash erasing, etc.

The commands are listed in the following table.

Table 23-2. Set of Commands

Command	Value	Mnemonic
Get Flash descriptor	0x00	GETD
Write page	0x01	WP
Write page and lock	0x02	WPL
Erase page and write page	0x03	EWP
Erase page and write page then lock	0x04	EWPL
Erase all	0x05	EA
Erase pages	0x07	EPA
Set lock bit	0x08	SLB
Clear lock bit	0x09	CLB
Get lock bit	0x0A	GLB
Set GPNVM bit	0x0B	SGPB
Clear GPNVM bit	0x0C	CGPB
Get GPNVM bit	0x0D	GGPB
Start read unique identifier	0x0E	STUI
Stop read unique identifier	0x0F	SPUI
Get CALIB bit	0x10	GCALB
Erase sector	0x11	ES
Write user signature	0x12	WUS
Erase user signature	0x13	EUS
Start read user signature	0x14	STUS
Stop read user signature	0x15	SPUS

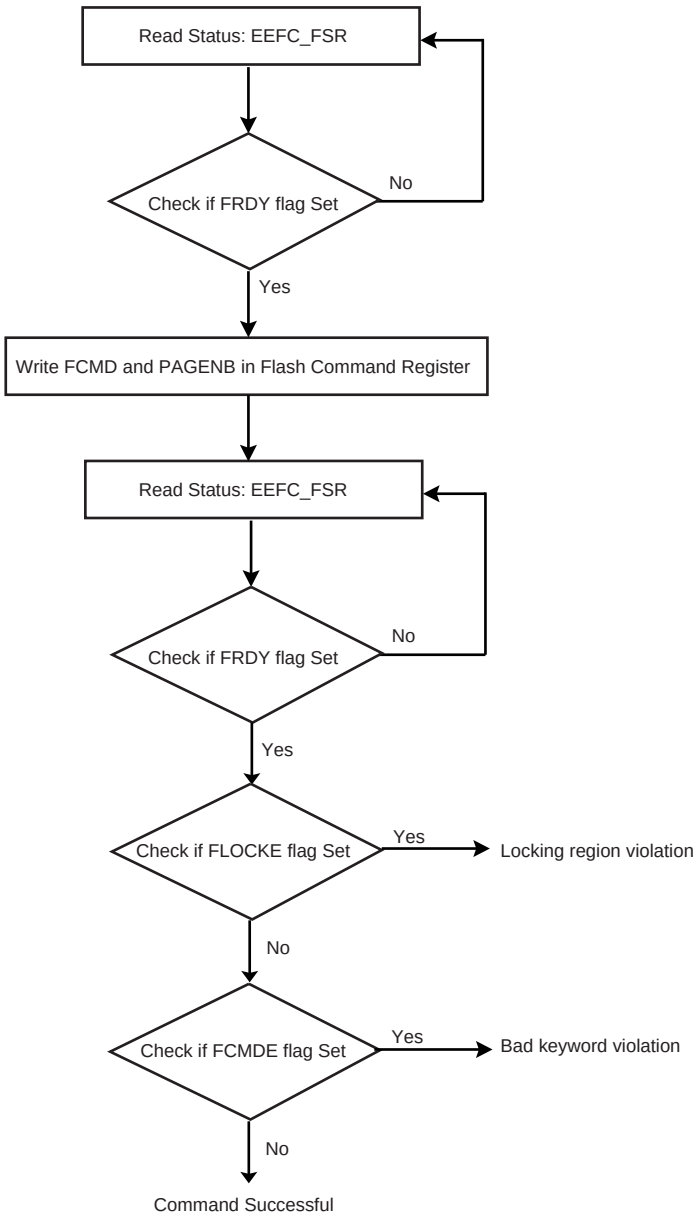
In order to execute one of these commands, select the required command using the FCMD field in the Flash Command register (EEFC_FCR). As soon as EEFC_FCR is written, the FRDY flag and the FVALUE field in the Flash Result register (EEFC_FRR) are automatically cleared. Once the current command has completed, the FRDY flag is automatically set. If an interrupt has been enabled by setting the bit EEFC_FMR.FRDY, the corresponding interrupt line of the interrupt controller is activated. (Note that this is true for all commands except for the STUI command. The FRDY flag is not set when the STUI command has completed.)

All the commands are protected by the same keyword, which must be written in the eight highest bits of EEFC_FCR.

Writing EEFC_FCR with data that does not contain the correct key and/or with an invalid command has no effect on the whole memory plane, but the FCMDE flag is set in the Flash Status register (EEFC_FSR). This flag is automatically cleared by a read access to EEFC_FSR.

When the current command writes or erases a page in a locked region, the command has no effect on the whole memory plane, but the FLOCKE flag is set in EEFC_FSR. This flag is automatically cleared by a read access to EEFC_FSR.

Figure 23-7. Command State Chart



23.4.3.1 Get Flash Descriptor Command

This command provides the system with information on the Flash organization. The system can take full advantage of this information. For instance, a device could be replaced by one with more Flash capacity, and so the software is able to adapt itself to the new configuration.

To get the embedded Flash descriptor, the application writes the GETD command in EEFC_FCR. The first word of the descriptor can be read by the software application in EEFC_FRR as soon as the FRDY flag in EEFC_FSR rises. The next reads of EEFC_FRR provide the following word of the descriptor. If extra read operations to EEFC_FRR are done after the last word of the descriptor has been returned, the EEFC_FRR value is 0 until the next valid command.

Table 23-3. 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

23.4.3.2 Write Commands

Several commands are used to program the Flash.

Only 0 values can be programmed using Flash technology; 1 is the erased value. In order to program words in a page, the page must first be erased. Commands are available to erase the full memory plane or a given number of pages. With the EWP and EWPL commands, a page erase is done automatically before a page programming.

After programming, the page (the entire lock region) can be locked to prevent miscellaneous write or erase sequences. The lock bit can be automatically set after page programming using WPL or EWPL commands.

Data to be programmed in the Flash must be written in an internal latch buffer before writing the programming command in EEFC_FCR. Data can be written at their final destination address, as the latch buffer is mapped into the Flash memory address space and wraps around within this Flash address space.

Byte and half-word AHB accesses to the latch buffer are not allowed. Only 32-bit word accesses are supported.

32-bit words must be written continuously, in either ascending or descending order. Writing the latch buffer in a random order is not permitted. This prevents mapping a C-code structure to the latch buffer and accessing the data of the structure in any order. It is instead recommended to fill in a C-code structure in SRAM and copy it in the latch buffer in a continuous order.

Write operations in the latch buffer are performed with the number of wait states programmed for reading the Flash.

The latch buffer is automatically re-initialized, i.e., written with logical ‘1’, after execution of each programming command.

The programming sequence is the following:

1. Write the data to be programmed in the latch buffer.
2. Write the programming command in EEFC_FCR. This automatically clears the bit EEFC_FSR.FRDY.
3. When Flash programming is completed, the bit EEFC_FSR.FRDY rises. If an interrupt has been enabled by setting the bit EEFC_FMR.FRDY, the interrupt line of the EEFC is activated.

Three errors can be detected in EEFC_FSR after a programming sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Lock Error: The page to be programmed belongs to a locked region. A command must be run previously to unlock the corresponding region.
- Flash Error: When programming is completed, the WriteVerify test of the Flash memory has failed.

Only one page can be programmed at a time. It is possible to program all the bits of a page (full page programming) or only some of the bits of the page (partial page programming).

Depending on the number of bits to be programmed within the page, the EEFC adapts the write operations required to program the Flash.

When a ‘Write Page’ (WP) command is issued, the EEFC starts the programming sequence and all the bits written at 0 in the latch buffer are cleared in the Flash memory array.

During programming, i.e., until EEFC_FSR.FDRY rises, access to the Flash is not allowed.

Full Page Programming

To program a full page, all the bits of the page must be erased before writing the latch buffer and issuing the WP command. The latch buffer must be written in ascending order, starting from the first address of the page. See Figure 23-8 "Full Page Programming".

Partial Page Programming

To program only part of a page using the WP command, the following constraints must be respected:

- Data to be programmed must be contained in integer multiples of 64-bit address-aligned words.
- 64-bit words can be programmed only if all the corresponding bits in the Flash array are erased (at logical value ‘1’).

See Figure 23-9 "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.

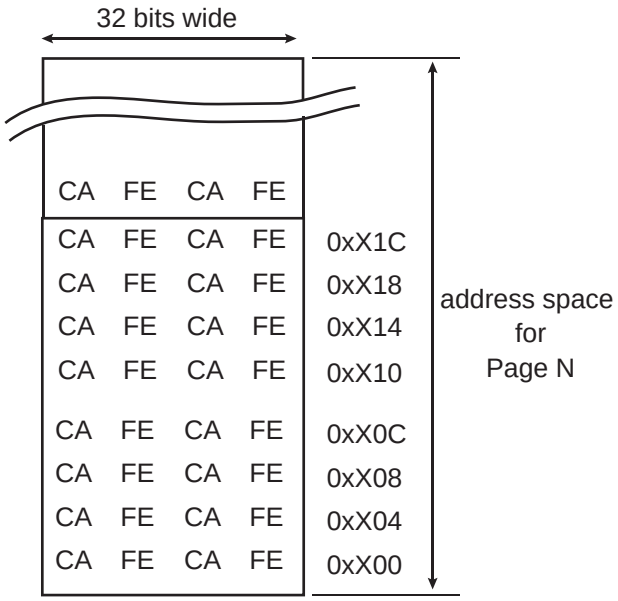
See Figure 23-10 "Optimized Partial Page Programming".

Programming Bytes

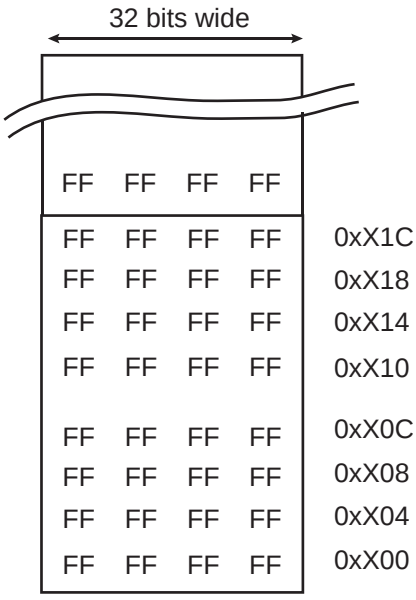
Individual bytes can be programmed using the Partial page programming mode.
In this case, an area of 64 bits must be reserved for each byte.

Refer to Figure 23-11 "Programming Bytes in the Flash".

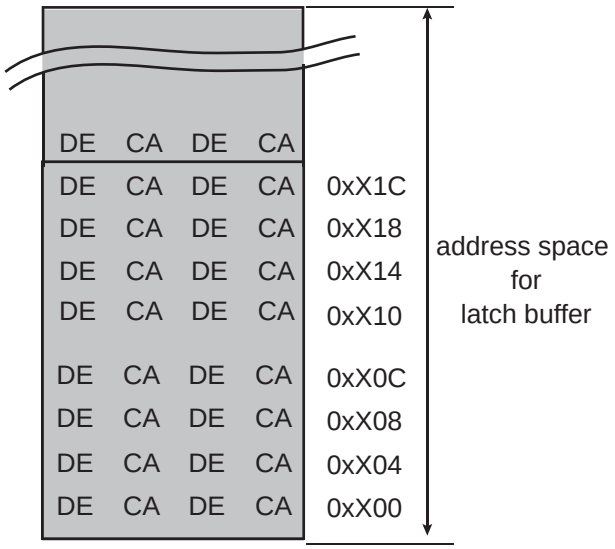
Figure 23-8. Full Page Programming



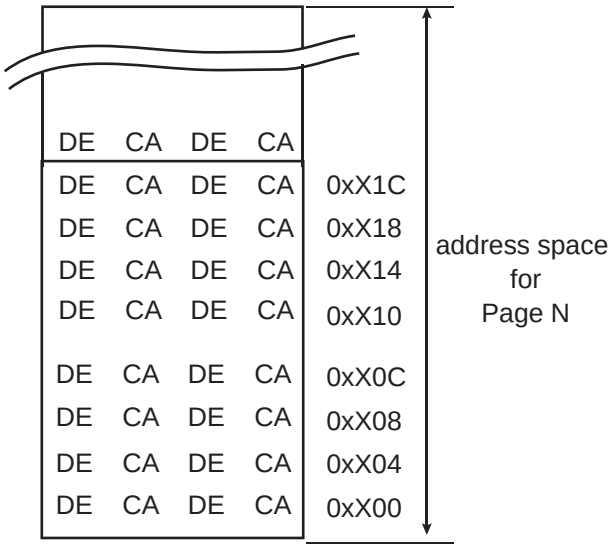
Before programming: Unerased page in Flash array



Step 1: Flash array after page erase



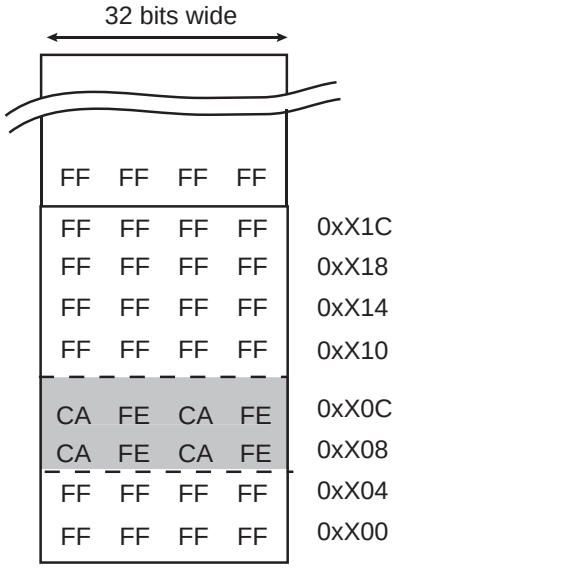
Step 2: Writing a page in the latch buffer



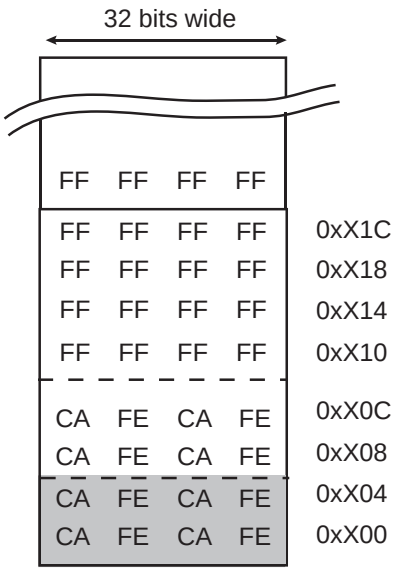
Step 3: Page in Flash array after issuing WP command and FRDY=1

Figure 23-9. Partial Page Programming

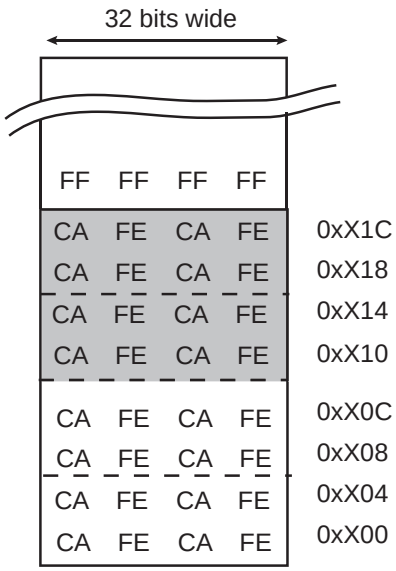
Step 1: Flash array after page erase



Step 2: Flash array after programming
64-bit at address 0xX08 (write latch buffer + WP)

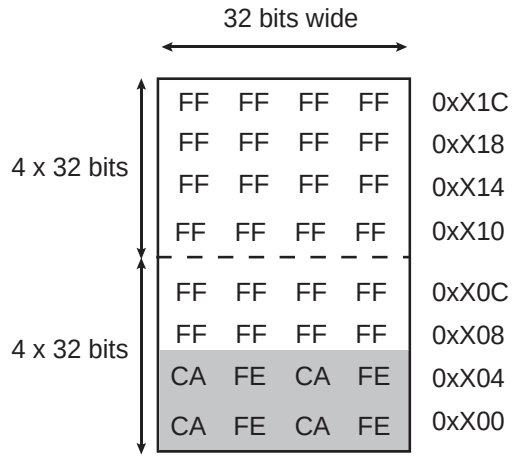


Step 3: Flash array after programming
a second 64-bit data at address 0xX00
(write latch buffer + WP)

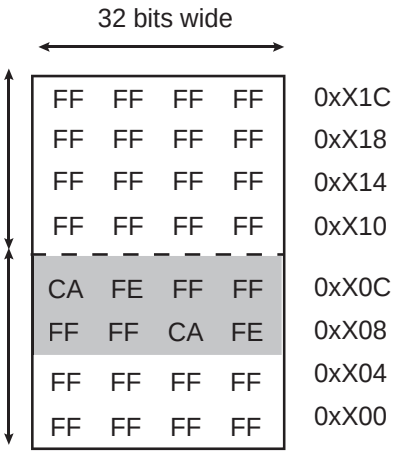


Step 4: Flash array after programming
a 128-bit data word at address 0xX10
(write latch buffer + WP)

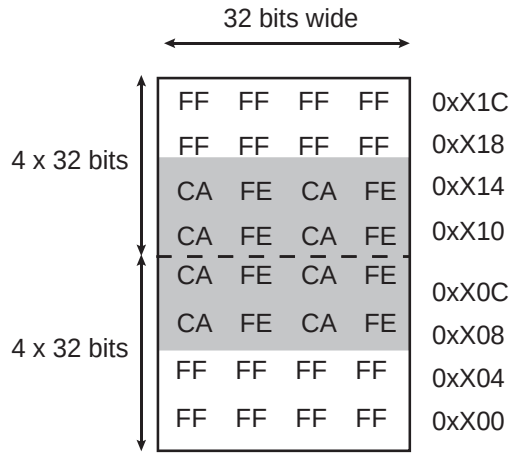
Figure 23-10. Optimized Partial Page Programming



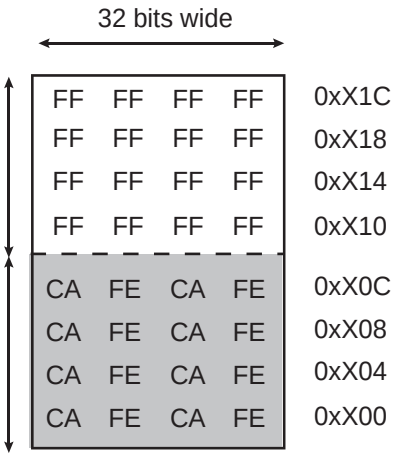
Case 1: 2 x 32 bits modified, not crossing 128-bit boundary
User programs WP, Flash Controller sends Write Word
=> Only 1 word programmed => programming period reduced



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

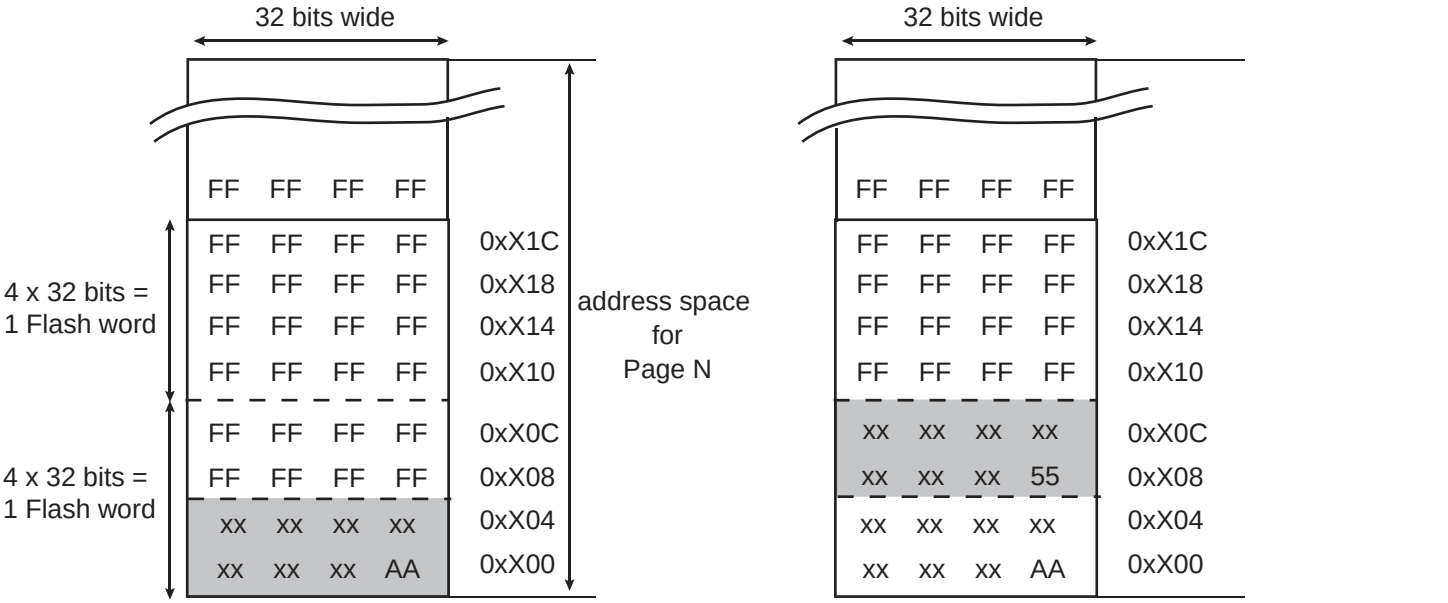


Case 3: 4 x 32 bits modified across 128-bit boundary
User programs WP, Flash Controller sends WP
=> Whole page programmed



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 23-11. 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.

23.4.3.3 Erase Commands

Erase commands are allowed only on unlocked regions. Depending on the Flash memory, several commands can be used to erase the Flash:

- Erase All Memory (EA): All memory is erased. The processor must not fetch code from the Flash memory.
- Erase Pages (EPA): 8 or 16 pages are erased in the Flash sector selected. The first page to be erased is specified in the FARG[15:2] field of the EEFC_FCR. The first page number must be a multiple of 8, 16 or 32 depending on the number of pages to erase at the same time.
- Erase Sector (ES): A full memory sector is erased. Sector size depends on the Flash memory. EEFC_FCR.FARG must be set with a page number that is in the sector to be erased.

Note: 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 executed, the processor accesses are stalled until the EPA command is completed. To avoid stalling the processor, the code can be run out of internal SRAM.

The erase sequence is the following:

1. Erase starts as soon as one of the erase commands and the FARG field are written in EEFC_FCR.
   - For the EPA command, the two lowest bits of the FARG field define the number of pages to be erased (FARG[1:0]):

Table 23-4. EEFC_FCR.FARG Field for EPA Command

FARG[1:0]	Number of pages to be erased with EPA command
0	4 pages (only valid for small 8 KB sectors)
1	8 pages (only valid for small 8 KB sectors)
2	16 pages
3	32 pages (not valid for small 8 KB sectors)

2. When erasing is completed, the bit EEFC_FSR.FRDY rises. If an interrupt has been enabled by setting the bit EEFC_FMR.FRDY, the interrupt line of the interrupt controller is activated.

Three errors can be detected in EEFC_FSR after an erasing sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Lock Error: At least one page to be erased belongs to a locked region. The erase command has been refused, no page has been erased. A command must be run previously to unlock the corresponding region.
- Flash Error: At the end of the erase period, the EraseVerify test of the Flash memory has failed.

23.4.3.4 Lock Bit Protection

Lock bits are associated with several pages in the embedded Flash memory plane. This defines lock regions in the embedded Flash memory plane. They prevent writing/erasing protected pages.

The lock sequence is the following:

1. Execute the ‘Set Lock Bit’ command by writing EEFC_FCR.FCMD with the SLB command and EEFC_FCR.FARG with a page number to be protected.
2. When the locking completes, the bit EEFC_FSR.FRDY rises. If an interrupt has been enabled by setting the bit EEFC_FMR.FRDY, the interrupt line of the interrupt controller is activated.
3. The result of the SLB command can be checked running a ‘Get Lock Bit’ (GLB) command.

Note: The value of the FARG argument passed together with SLB command must not exceed the higher lock bit index available in the product.

Two errors can be detected in EEFC_FSR after a programming sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Flash Error: At the end of the programming, the EraseVerify or WriteVerify test of the Flash memory has failed.

It is possible to clear lock bits previously set. After the lock bits are cleared, the locked region can be erased or programmed. The unlock sequence is the following:

1. Execute the ‘Clear Lock Bit’ command by writing EEFC_FCR.FCMD with the CLB command and EEFC_FCR.FARG with a page number to be unprotected.
2. When the unlock completes, the bit EEFC_FSR.FRDY rises. If an interrupt has been enabled by setting the bit EEFC_FMR.FRDY, the interrupt line of the interrupt controller is activated.

Note: The value of the FARG argument passed together with CLB command must not exceed the higher lock bit index available in the product.

Two errors can be detected in EEFC_FSR after a programming sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Flash Error: At the end of the programming, the EraseVerify or WriteVerify test of the Flash memory has failed.

The status of lock bits can be returned by the EEFC. The ‘Get Lock Bit’ sequence is the following:

1. Execute the ‘Get Lock Bit’ command by writing EEFC_FCR.FCMD with the GLB command. Field EEFC_FCR.FARG is meaningless.
2. Lock bits can be read by the software application in EEFC_FRR. The first word read corresponds to the 32 first lock bits, next reads providing the next 32 lock bits as long as it is meaningful. Extra reads to EEFC_FRR return 0.

For example, if the third bit of the first word read in EEFC_FRR is set, the third lock region is locked.

Two errors can be detected in EEFC_FSR after a programming sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Flash Error: At the end of the programming, the EraseVerify or WriteVerify test of the Flash memory has failed.

Note: Access to the Flash in read is permitted when a ‘Set Lock Bit’, ‘Clear Lock Bit’ or ‘Get Lock Bit’ command is executed.

23.4.3.5 GPNVM Bit

GPNVM bits do not interfere with the embedded Flash memory plane. For more details, refer to the section “Memories” of this datasheet.

The ‘Set GPNVM Bit’ sequence is the following:

1. Execute the ‘Set GPNVM Bit’ command by writing EEFC_FCR.FCMD with the SGPB command and EEFC_FCR.FARG with the number of GPNVM bits to be set.
2. When the GPNVM bit is set, the bit EEFC_FSR.FRDY rises. If an interrupt was enabled by setting the bit EEFC_FMR.FRDY, the interrupt line of the interrupt controller is activated.
3. The result of the SGPB command can be checked by running a ‘Get GPNVM Bit’ (GGPB) command.

Note: The value of the FARG argument passed together with SGPB command must not exceed the higher GPNVM index available in the product. Flash data content is not altered if FARG exceeds the limit. Command Error is detected only if FARG is greater than 8.

Two errors can be detected in EEFC_FSR after a programming sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Flash Error: At the end of the programming, the EraseVerify or WriteVerify test of the Flash memory has failed.

It is possible to clear GPNVM bits previously set. The ‘Clear GPNVM Bit’ sequence is the following:

1. Execute the ‘Clear GPNVM Bit’ command by writing EEFC_FCR.FCMD with the CGPB command and EEFC_FCR.FARG with the number of GPNVM bits to be cleared.
2. When the clear completes, the bit EEFC_FSR.FRDY rises. If an interrupt has been enabled by setting the bit EEFC_FMR.FRDY, the interrupt line of the interrupt controller is activated.

Note: The value of the FARG argument passed together with CGPB command must not exceed the higher GPNVM index available in the product. Flash data content is not altered if FARG exceeds the limit. Command Error is detected only if FARG is greater than 8.

Two errors can be detected in EEFC_FSR after a programming sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Flash Error: At the end of the programming, the EraseVerify or WriteVerify test of the Flash memory has failed.

The status of GPNVM bits can be returned by the EEFC. The sequence is the following:

1. Execute the ‘Get GPNVM Bit’ command by writing EEFC_FCR.FCMD with the GGPB command. Field EEFC_FCR.FARG is meaningless.
2. GPNVM bits can be read by the software application in EEFC_FRR. The first word read corresponds to the 32 first GPNVM bits, following reads provide the next 32 GPNVM bits as long as it is meaningful. Extra reads to EEFC_FRR return 0.

For example, if the third bit of the first word read in EEFC_FRR is set, the third GPNVM bit is active.

One error can be detected in EEFC_FSR after a programming sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.

Note: Access to the Flash in read is permitted when a ‘Set GPNVM Bit’, ‘Clear GPNVM Bit’ or ‘Get GPNVM Bit’ command is executed.

23.4.3.6 Calibration Bit

Calibration bits do not interfere with the embedded Flash memory plane.

The calibration bits cannot be modified.

The status of calibration bits are returned by the EEFC. The sequence is the following:

1. Execute the ‘Get CALIB Bit’ command by writing EEFC_FCR.FCMD with the GCALB command. Field EEFC_FCR.FARG is meaningless.
2. Calibration bits can be read by the software application in EEFC_FRR. The first word read corresponds to the first 32 calibration bits. The following reads provide the next 32 calibration bits as long as it is meaningful. Extra reads to EEFC_FRR return 0.

The 16/24 MHz internal RC oscillator is calibrated in production. This calibration can be read through the GCALB command. Table 23-5 shows the bit implementation.

The RC calibration for the 8 MHz is set to ‘1000000’.

Table 23-5. Calibration Bit Indexes

Description	EEFC_FRR Bits
Low-power regulator trimming value (1)	[117–114]
16 MHz RC calibration output	[28–22]
24 MHz RC calibration output	[38–32]

Note: 1. Refer to bits SUPC_PWMR.LPOWER0–3 in the section “Supply Controller (SUPC)”.

23.4.3.7 Security Bit Protection

When the security bit is enabled, access to the Flash through the SWD interface or through the Fast Flash Programming interface is forbidden. This ensures the confidentiality of the code programmed in the Flash.

The security bit is GPNVM0.

Disabling the security bit can only be achieved by asserting the ERASE pin at ‘1’, and after a full Flash erase is performed. When the security bit is deactivated, all accesses to the Flash are permitted.

23.4.3.8 Unique Identifier Area

Each device is programmed with a 2*512-bytes unique identifier area . See Figure 23-1 "Flash Memory Areas".

The sequence to read the unique identifier area is the following:

1. Execute the ‘Start Read Unique Identifier’ command by writing EEFC_FCR.FCMD with the STUI command. Field EEFC_FCR.FARG is meaningless.
2. Wait until the bit EEFC_FSR.FRDY falls to read the unique identifier area. The unique identifier field is located in the first 128 bits of the Flash memory mapping. The ‘Start Read Unique Identifier’ command reuses some addresses of the memory plane for code, but the unique identifier area is physically different from the memory plane for code.
3. To stop reading the unique identifier area, execute the ‘Stop Read Unique Identifier’ command by writing EEFC_FCR.FCMD with the SPUI command. Field EEFC_FCR.FARG is meaningless.
4. When the SPUI command has been executed, the bit EEFC_FSR.FRDY rises. If an interrupt was enabled by setting the bit EEFC_FMR.FRDY, the interrupt line of the interrupt controller is activated.

Note that during the sequence, the software cannot be fetched from the Flash.

23.4.3.9 User Signature Area

Each product contains a user signature area of 512-bytes. It can be used for storage. Read, write and erase of this area is allowed.

See Figure 23-1 "Flash Memory Areas".

The sequence to read the user signature area is the following:

1. Execute the ‘Start Read User Signature’ command by writing EEFC_FCR.FCMD with the STUS command. Field EEFC_FCR.FARG is meaningless.
2. Wait until the bit EEFC_FSR.FRDY falls to read the user signature area. The user signature area is located in the first 512 bytes of the Flash memory mapping. The ‘Start Read User Signature’ command reuses some addresses of the memory plane but the user signature area is physically different from the memory plane
3. To stop reading the user signature area, execute the ‘Stop Read User Signature’ command by writing EEFC_FCR.FCMD with the SPUS command. Field EEFC_FCR.FARG is meaningless.
4. When the SPUI command has been executed, the bit EEFC_FSR.FRDY rises. If an interrupt was enabled by setting the bit EEFC_FMR.FRDY, the interrupt line of the interrupt controller is activated.

Note that during the sequence, the software cannot be fetched from the Flash or from the second plane in case of dual plane.

One error can be detected in EEFC_FSR after this sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.

The sequence to write the user signature area is the following:

1. Write the full page, at any page address, within the internal memory area address space.
2. Execute the ‘Write User Signature’ command by writing EEFC_FCR.FCMD with the WUS command. Field EEFC_FCR.FARG is meaningless.
3. When programming is completed, the bit EEFC_FSR.FRDY rises. If an interrupt has been enabled by setting the bit EEFC_FMR.FRDY, the corresponding interrupt line of the interrupt controller is activated.

Two errors can be detected in EEFC_FSR after this sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Flash Error: At the end of the programming, the WriteVerify test of the Flash memory has failed.

The sequence to erase the user signature area is the following:

1. Execute the ‘Erase User Signature’ command by writing EEFC_FCR.FCMD with the EUS command. Field EEFC_FCR.FARG is meaningless.
2. When programming is completed, the bit EEFC_FSR.FRDY rises. If an interrupt has been enabled by setting the bit EEFC_FMR.FRDY, the corresponding interrupt line of the interrupt controller is activated.

Two errors can be detected in EEFC_FSR after this sequence:

- Command Error: A bad keyword has been written in EEFC_FCR.
- Flash Error: At the end of the programming, the EraseVerify test of the Flash memory has failed.

23.4.4 Register Write Protection

To prevent any single software error from corrupting EEFC behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the “EEFC Write Protection Mode Register” (EEFC_WPMR).

The following register can be write-protected:

- “EEFC Flash Mode Register”

23.5 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 23-6. Register Mapping

Offset	Register	Name	Access	Reset State
0x00	EEFC Flash Mode Register	EEFC_FMR	Read/Write	0x0400_0000
0x04	EEFC Flash Command Register	EEFC_FCR	Write-only	–
0x08	EEFC Flash Status Register	EEFC_FSR	Read-only	0x0000_0001
0x0C	EEFC Flash Result Register	EEFC_FRR	Read-only	0x0
0x10–0x14	Reserved	–	–	–
0x18–0xE0	Reserved	–	–	–
0xE4	Write Protection Mode Register	EEFC_WPMR	Read/Write	0x0

23.5.1 EEFC Flash Mode Register

Name: EEFC_FMR

Address: 0x400E0A00

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the “EEFC Write Protection Mode Register” .

- **FRDY: Flash Ready Interrupt Enable**

0: Flash ready does not generate an interrupt.

1: Flash ready (to accept a new command) generates an interrupt.

- **FWS: Flash Wait State**

This field defines the number of wait states for read and write operations:

FWS = Number of cycles for Read/Write operations - 1

- **SCOD: Sequential Code Optimization Disable**

0: The sequential code optimization is enabled.

1: The sequential code optimization is disabled.

No Flash read should be done during change of this field.

- **FAM: Flash Access Mode**

0: 128-bit access in Read mode only to enhance access speed.

1: 64-bit access in Read mode only to enhance power consumption.

No Flash read should be done during change of this field.

- **CLOE: Code Loop Optimization Enable**

0: The opcode loop optimization is disabled.

1: The opcode loop optimization is enabled.

No Flash read should be done during change of this field.

23.5.2 EEFC Flash Command Register

Name: EEFC_FCR

Address: 0x400E0A04

Access: Write-only

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

- **FCMD: Flash Command**

Value	Name	Description
0x00	GETD	Get Flash descriptor
0x01	WP	Write page
0x02	WPL	Write page and lock
0x03	EWP	Erase page and write page
0x04	EWPL	Erase page and write page then lock
0x05	EA	Erase all
0x07	EPA	Erase pages
0x08	SLB	Set lock bit
0x09	CLB	Clear lock bit
0x0A	GLB	Get lock bit
0x0B	SGPB	Set GPNVM bit
0x0C	CGPB	Clear GPNVM bit
0x0D	GGPB	Get GPNVM bit
0x0E	STUI	Start read unique identifier
0x0F	SPUI	Stop read unique identifier
0x10	GCALB	Get CALIB bit
0x11	ES	Erase sector
0x12	WUS	Write user signature
0x13	EUS	Erase user signature
0x14	STUS	Start read user signature
0x15	SPUS	Stop read user signature

- **FARG: Flash Command Argument**

GETD, GLB, GGPB, STUI, SPUI, GCALB, WUS, EUS, STUS, SPUS, EA	Commands requiring no argument, including Erase all command	FARG is meaningless, must be written with 0
ES	Erase sector command	FARG must be written with any page number within the sector to be erased
EPA	Erase pages command	FARG[1:0] defines the number of pages to be erased The start page must be written in FARG[15:2]. FARG[1:0] = 0: Four pages to be erased. FARG[15:2] = Page_Number / 4 FARG[1:0] = 1: Eight pages to be erased. FARG[15:3] = Page_Number / 8, FARG[2] = 0 FARG[1:0] = 2: Sixteen pages to be erased. FARG[15:4] = Page_Number / 16, FARG[3:2] = 0 FARG[1:0] = 3: Thirty-two pages to be erased. FARG[15:5] = Page_Number / 32, FARG[4:2] = 0 Refer to Table 23-4 “EEFC_FCR.FARG Field for EPA Command”.
WP, WPL, EWP, EWPL	Programming commands	FARG must be written with the page number to be programmed
SLB, CLB	Lock bit commands	FARG defines the page number to be locked or unlocked
SGPB, CGPB	GPNVM commands	FARG defines the GPNVM number to be programmed

- **FKEY: Flash Writing Protection Key**

Value	Name	Description
0x5A	PASSWD	The 0x5A value enables the command defined by the bits of the register. If the field is written with a different value, the write is not performed and no action is started.

23.5.3 EEFC Flash Status Register

Name: EEFC_FSR

Address: 0x400E0A08

Access: Read-only

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

- **FRDY: Flash Ready Status (cleared when Flash is busy)**

0: The EEFC is busy.

1: The EEFC is ready to start a new command.

When set, this flag triggers an interrupt if the FRDY flag is set in EEFC_FMR.

This flag is automatically cleared when the EEFC is busy.

- **FCMDE: Flash Command Error Status (cleared on read or by writing EEFC_FCR)**

0: No invalid commands and no bad keywords were written in EEFC_FMR.

1: An invalid command and/or a bad keyword was/were written in EEFC_FMR.

- **FLOCKE: Flash Lock Error Status (cleared on read)**

0: No programming/erase of at least one locked region has happened since the last read of EEFC_FSR.

1: Programming/erase of at least one locked region has happened since the last read of EEFC_FSR.

This flag is automatically cleared when EEFC_FSR is read or EEFC_FCR is written.

- **FLERR: Flash Error Status (cleared when a programming operation starts)**

0: No Flash memory error occurred at the end of programming (EraseVerify or WriteVerify test has passed).

1: A Flash memory error occurred at the end of programming (EraseVerify or WriteVerify test has failed).

23.5.4 EEFC Flash Result Register

Name: EEFC_FRR

Address: 0x400E0A0C

Access: Read-only

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

- **FVALUE: Flash Result Value**

The result of a Flash command is returned in this register. If the size of the result is greater than 32 bits, the next resulting value is accessible at the next register read.

23.5.5 EEFC Write Protection Mode Register

Name: EEFC_WPMR

Address: 0x400E0AE4

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 0x454643 (EFC in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x454643 (EFC in ASCII).

See Section 23.4.4 ”Register Write Protection” for the list of registers that can be protected.

- **WPKEY: Write Protection Key**

Value	Name	Description
0x454643	PASSWD	Writing any other value in this field aborts the write operation. Always reads as 0.

24. Timer Counter (TC)

24.1 Description

A Timer Counter (TC) module includes three identical TC channels. The number of implemented TC modules is device-specific.

Each TC channel can be independently programmed to perform a wide range of functions including frequency measurement, event counting, interval measurement, pulse generation, delay timing and pulse width modulation.

Each channel has three external clock inputs, five internal clock inputs and two multi-purpose input/output signals which can be configured by the user. Each channel drives an internal interrupt signal which can be programmed to generate processor interrupts.

The TC 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

24.2 Embedded Characteristics

- Total number of TC channels implemented on this device: 6
- TC channel size: 16-bit
- Wide range of functions including:
  - Frequency measurement
  - Event counting
  - Interval measurement
  - Pulse generation
  - Delay timing
  - Pulse Width Modulation
  - Up/down capabilities
  - 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
- Read of the Capture registers by the PDC
- Register Write Protection

24.3 Block Diagram

Table 24-1. Timer Counter Clock Assignment

Name	Definition
TIMER_CLOCK1	MCK/2
TIMER_CLOCK2	MCK/8
TIMER_CLOCK3	MCK/32
TIMER_CLOCK4	MCK/128
TIMER_CLOCK5	PCK3

Figure 24-1. Timer Counter Block Diagram

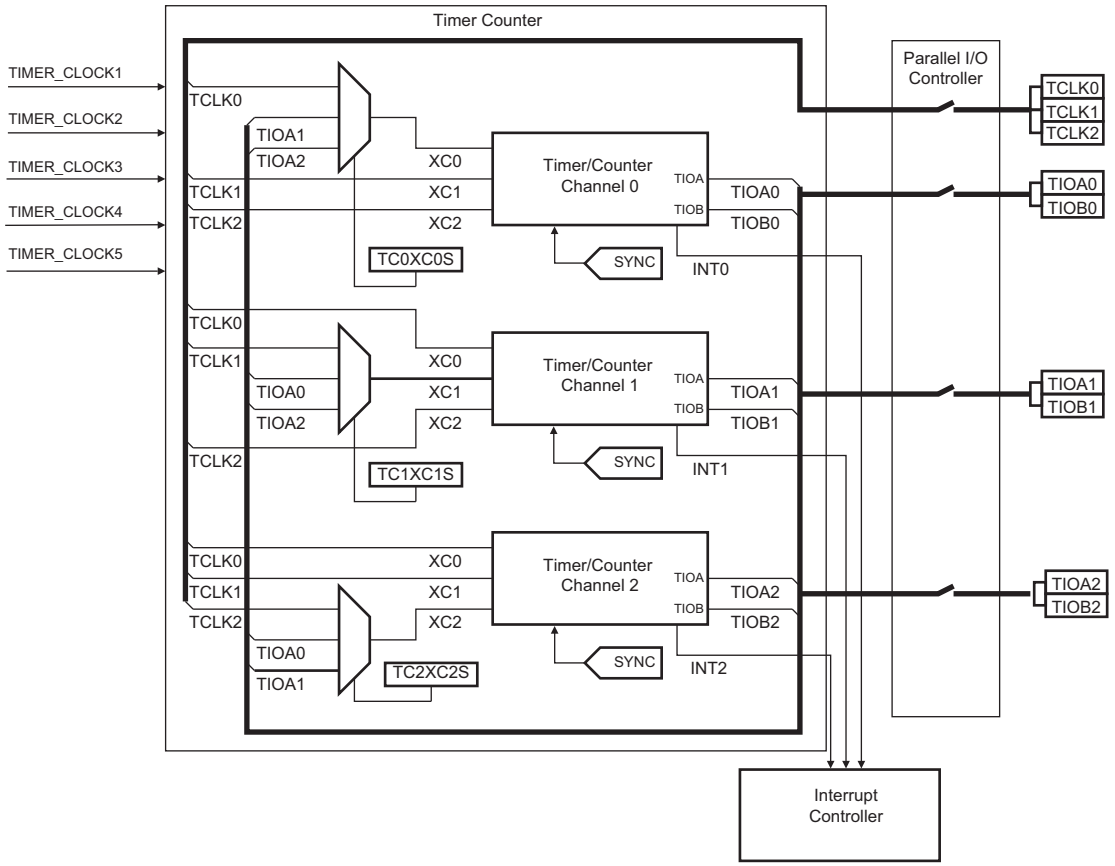


Table 24-2. Channel Signal Description

Signal Name	Description
XC0, XC1, XC2	External Clock Inputs
TIOAx	Capture Mode: Timer Counter Input Waveform Mode: Timer Counter Output
TIOBx	Capture Mode: Timer Counter Input Waveform Mode: Timer Counter Input/Output
INT	Interrupt Signal Output (internal signal)
SYNC	Synchronization Input Signal (from configuration register)

24.4 Pin List

Table 24-3. Pin List

Pin Name	Description	Type
TCLK0–TCLK2	External Clock Input	Input
TIOA0–TIOA2	I/O Line A	I/O
TIOB0–TIOB2	I/O Line B	I/O

24.5 Product Dependencies

24.5.1 I/O Lines

The pins used for interfacing the compliant external devices may be multiplexed with PIO lines. The programmer must first program the PIO controllers to assign the TC pins to their peripheral functions.

Table 24-4. I/O Lines

Instance	Signal	I/O Line	Peripheral
TC0	TCLK0	PA2	A
TC0	TCLK1	PA19	A
TC0	TCLK2	PA20	A
TC0	TIOA0	PA0	B
TC0	TIOA1	PA23	B
TC0	TIOA2	PA21	A
TC0	TIOB0	PA1	B
TC0	TIOB1	PA16	B
TC0	TIOB2	PA22	A

24.5.2 Power Management

The TC is clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the Timer Counter clock of each channel.

24.5.3 Interrupt Sources

The TC has an interrupt line per channel connected to the interrupt controller. Handling the TC interrupt requires programming the interrupt controller before configuring the TC.

Table 24-5. Peripheral IDs

Instance	ID
TC0	23
TC1	24

24.6 Functional Description

24.6.1 Description

All channels of the Timer Counter are independent and identical in operation. The registers for channel programming are listed in Table 24-6 “Register Mapping”.

24.6.2 16-bit Counter

Each 16-bit channel is organized around a 16-bit counter. The value of the counter is incremented at each positive edge of the selected clock. When the counter has reached the value 2^{16}-1 and passes to zero, an overflow occurs and the COVFS bit in the TC Status Register (TC_SR) is set.

The current value of the counter is accessible in real time by reading the TC Counter Value Register (TC_CV). The counter can be reset by a trigger. In this case, the counter value passes to zero on the next valid edge of the selected clock.

24.6.3 Clock Selection

At block level, input clock signals of each channel can be connected either to the external inputs TCLKx, or to the internal I/O signals TIOAx for chaining(1) by programming the TC Block Mode Register (TC_BMR). See Figure 24-2.

Each channel can independently select an internal or external clock source for its counter(2):

- External clock signals: XC0, XC1 or XC2
- Internal clock signals: MCK/2, MCK/8, MCK/32, MCK/128, PCK3

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 24-3.

Notes:
1. In Waveform mode, to chain two timers, it is mandatory to initialize some parameters:
   - Configure TIOx outputs to 1 or 0 by writing the required value to TC_CMR.ASWTRG.
   - Bit TC_BCR.SYNC must be written to 1 to start the channels at the same time.
2. In all cases, if an external clock or asynchronous internal clock PCK3 is used, the duration of each of its levels must be longer than the peripheral clock period, so the clock frequency will be at least 2.5 times lower than the peripheral clock.

Figure 24-2. Clock Chaining Selection

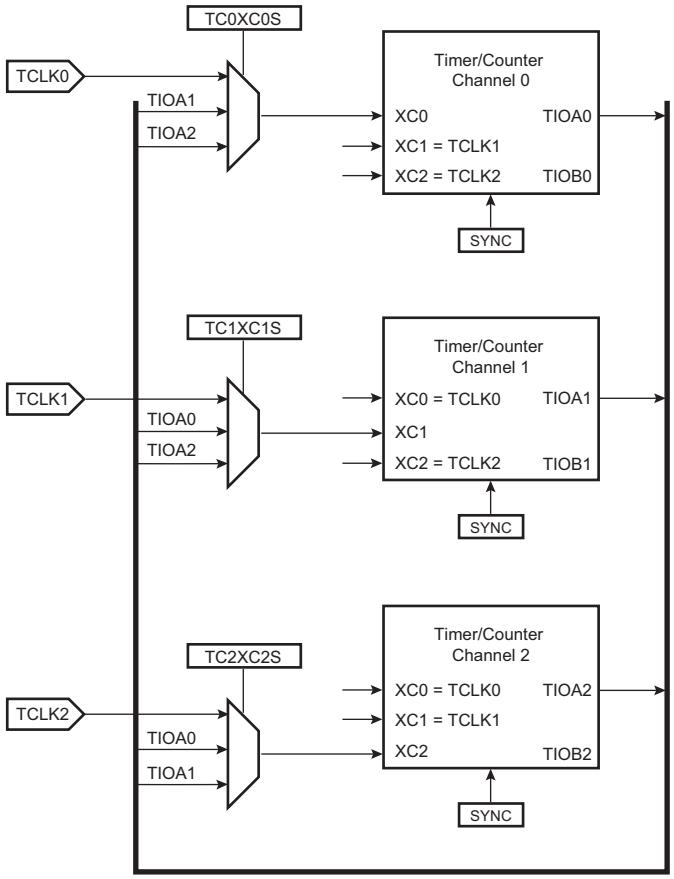
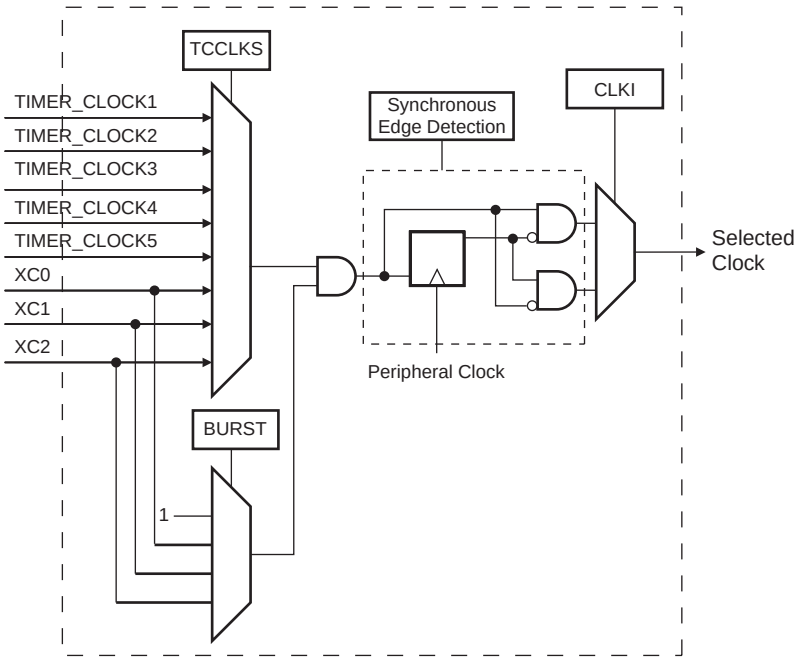


Figure 24-3. Clock Selection

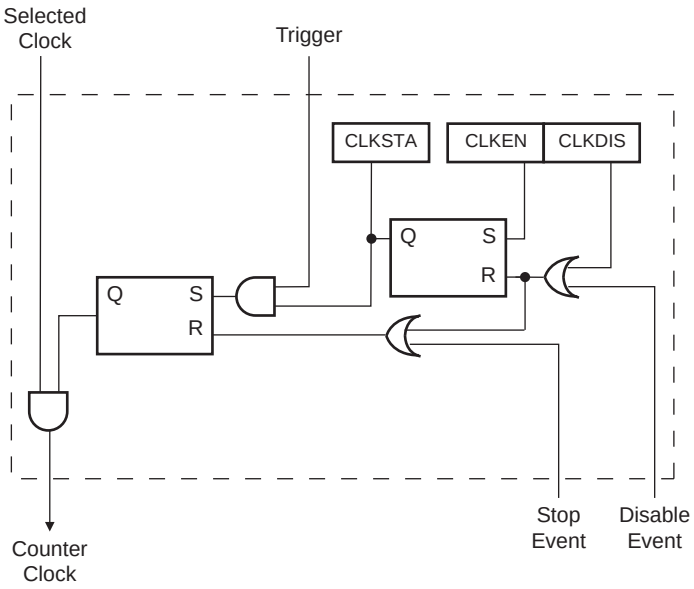


24.6.4 Clock Control

The clock of each counter can be controlled in two different ways: it can be enabled/disabled and started/stopped. See Figure 24-4.

- The clock can be enabled or disabled by the user with the CLKEN and the CLKDIS commands in the TC Channel Control Register (TC_CCR). In Capture mode it can be disabled by an RB load event if LDBDIS is set to 1 in the TC_CMR. In Waveform mode, it can be disabled by an RC Compare event if CPCDIS is set to 1 in TC_CMR. When disabled, the start or the stop actions have no effect: only a CLKEN command in the TC_CCR can re-enable the clock. When the clock is enabled, the CLKSTA bit is set in the TC_SR.
- The clock can also be started or stopped: a trigger (software, synchro, external or compare) always starts the clock. The clock can be stopped by an RB load event in Capture mode (LDBSTOP = 1 in TC_CMR) or an RC compare event in Waveform mode (CPCSTOP = 1 in TC_CMR). The start and the stop commands are effective only if the clock is enabled.

Figure 24-4. Clock Control



24.6.5 Operating Modes

Each channel can operate independently in two different modes:

- Capture mode provides measurement on signals.
- Waveform mode provides wave generation.

The TC operating mode is programmed with the WAVE bit in the TC_CMR.

In Capture mode, TIOAx and TIOBx are configured as inputs.

In Waveform mode, TIOAx is always configured to be an output and TIOBx is an output if it is not selected to be the external trigger.

24.6.6 Trigger

A trigger resets the counter and starts the counter clock. Three types of triggers are common to both modes, and a fourth external trigger is available to each mode.

Regardless of the trigger used, it will be taken into account at the following active edge of the selected clock. This means that the counter value can be read differently from zero just after a trigger, especially when a low frequency signal is selected as the clock.

The following triggers are common to both modes:

- Software Trigger: Each channel has a software trigger, available by setting SWTRG in TC_CCR.
- SYNC: Each channel has a synchronization signal SYNC. When asserted, this signal has the same effect as a software trigger. The SYNC signals of all channels are asserted simultaneously by writing TC_BCR (Block Control) with SYNC set.
- Compare RC Trigger: RC is implemented in each channel and can provide a trigger when the counter value matches the RC value if CPCTRG is set in the TC_CMR.

The channel can also be configured to have an external trigger. In Capture mode, the external trigger signal can be selected between TIOAx and TIOBx. In Waveform mode, an external event can be programmed on one of the following signals: TIOBx, XC0, XC1 or XC2. This external event can then be programmed to perform a trigger by setting bit ENETRG in the TC_CMR.

If an external trigger is used, the duration of the pulses must be longer than the peripheral clock period in order to be detected.

24.6.7 Capture Mode

Capture mode is entered by clearing the WAVE bit in the TC_CMR.

Capture mode allows the TC channel to perform measurements such as pulse timing, frequency, period, duty cycle and phase on TIOAx and TIOBx signals which are considered as inputs.

Figure 24-6 shows the configuration of the TC channel when programmed in Capture mode.

24.6.8 Capture Registers A and B

Registers A and B (RA and RB) are used as capture registers. They can be loaded with the counter value when a programmable event occurs on the signal TIOAx.

The LDRA field in the TC_CMR defines the TIOAx selected edge for the loading of register A, and the LDRB field defines the TIOAx selected edge for the loading of Register B.

The subsampling ratio defined by the SBSMPLR field in TC_CMR is applied to these selected edges, so that the loading of Register A and Register B occurs once every 1, 2, 4, 8 or 16 selected edges.

RA is loaded only if it has not been loaded since the last trigger or if RB has been loaded since the last loading of RA.

RB is loaded only if RA has been loaded since the last trigger or the last loading of RB.

Loading RA or RB before the read of the last value loaded sets the Overrun Error Flag (LOVRS bit) in the TC_SR. In this case, the old value is overwritten.

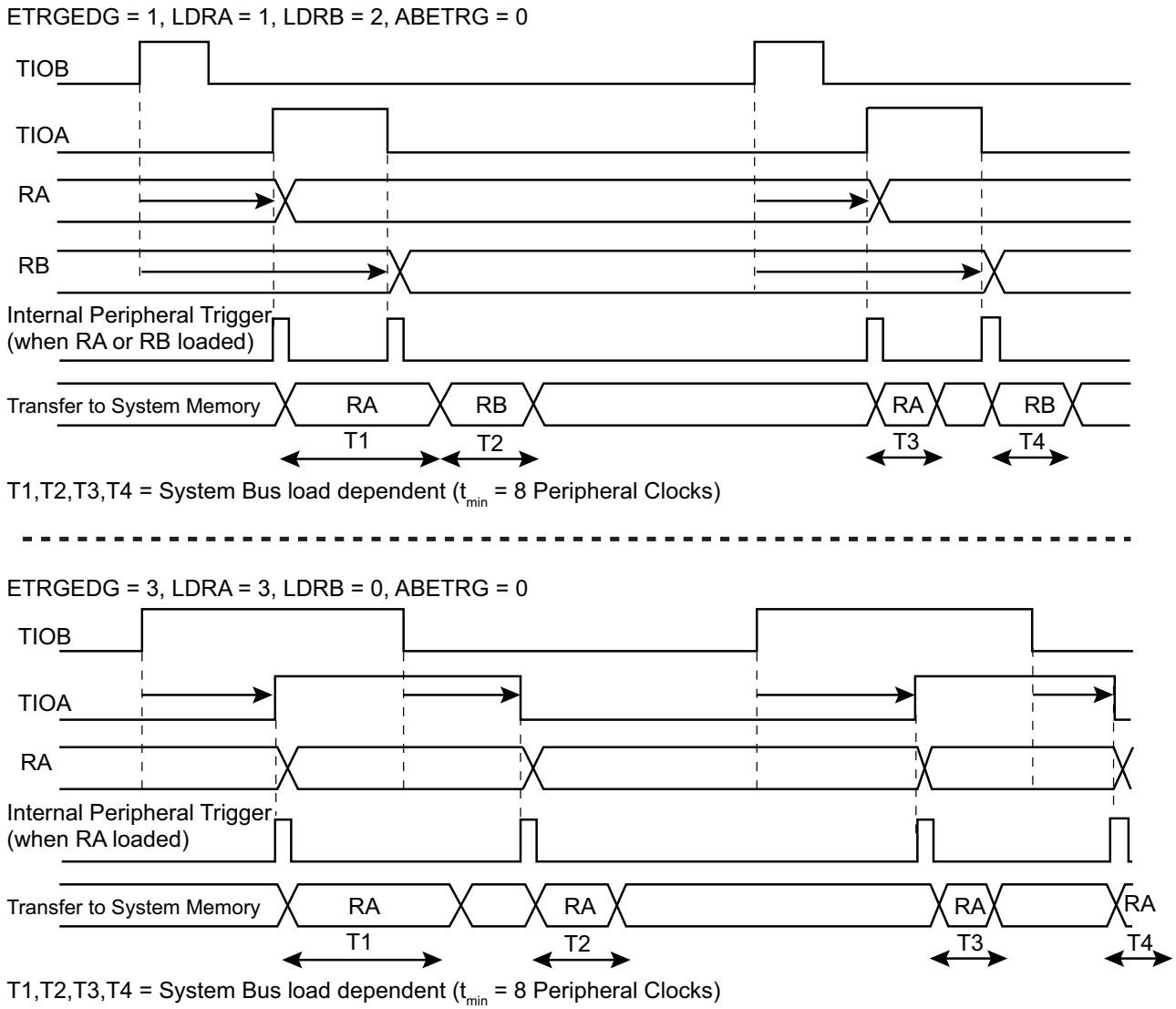
When DMA is used, the RAB register address must be configured as source address of the transfer. The RAB register provides the next unread value from Register A and Register B. It may be read by the DMA after a request has been triggered upon loading Register A or Register B.

24.6.9 Transfer with PDC in Capture Mode

The PDC can perform access from the TC to system memory in Capture mode only.

Figure 24-5 illustrates how TC_RA and TC_RB can be loaded in the system memory without CPU intervention.

Figure 24-5. Example of Transfer with PDC in Capture Mode

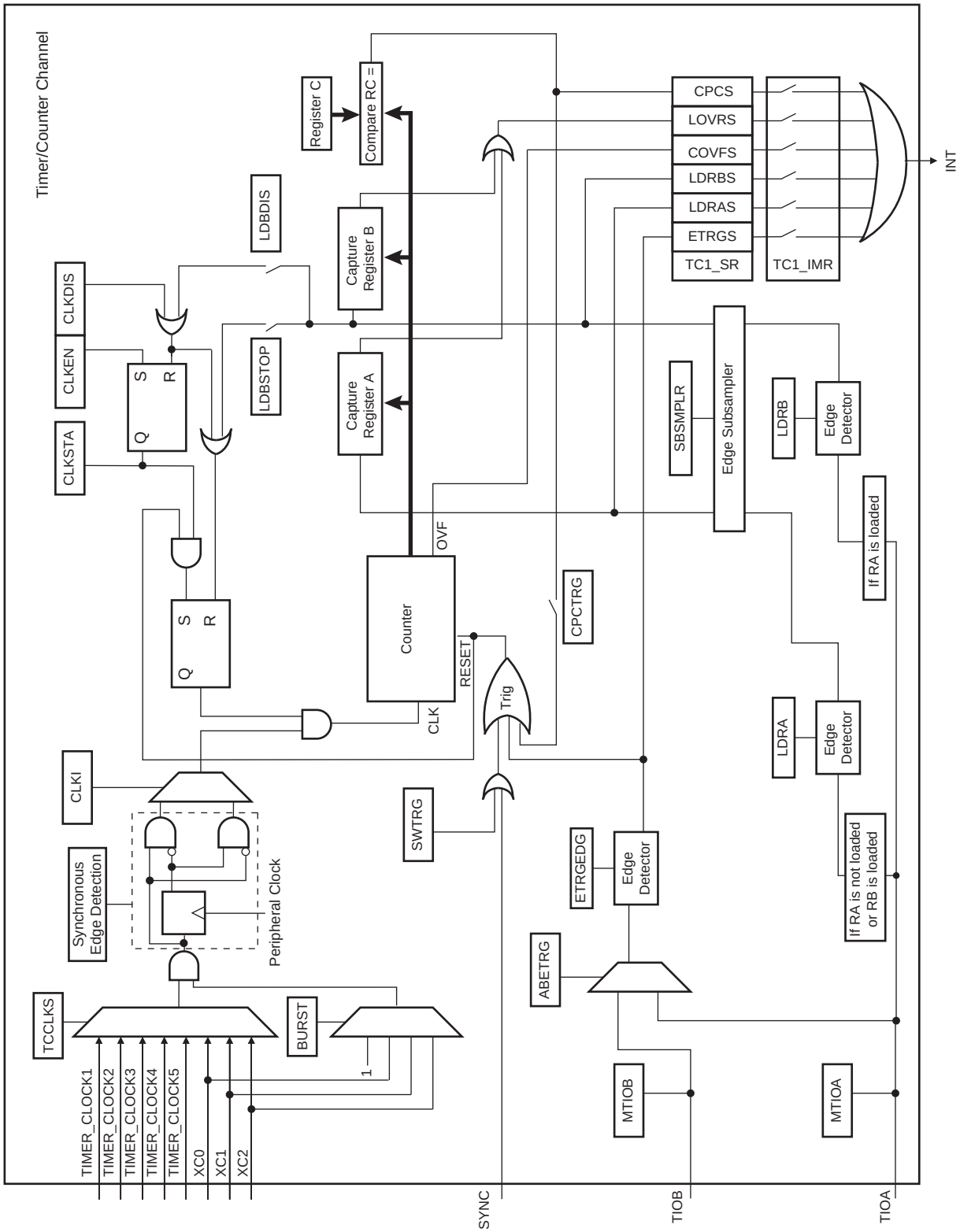


24.6.10 Trigger Conditions

In addition to the SYNC signal, the software trigger and the RC compare trigger, an external trigger can be defined.

The ABETRG bit in the TC_CMR selects TIOAx or TIOBx input signal as an external trigger . 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 24-6. Capture Mode



24.6.11 Waveform Mode

Waveform mode is entered by setting the TC_CMRx.WAVE bit.

In Waveform mode, the TC channel generates one or two PWM signals with the same frequency and independently programmable duty cycles, or generates different types of one-shot or repetitive pulses.

In this mode, TIOAx is configured as an output and TIOBx is defined as an output if it is not used as an external event (EEVT parameter in TC_CMR).

Figure 24-7 shows the configuration of the TC channel when programmed in Waveform operating mode.

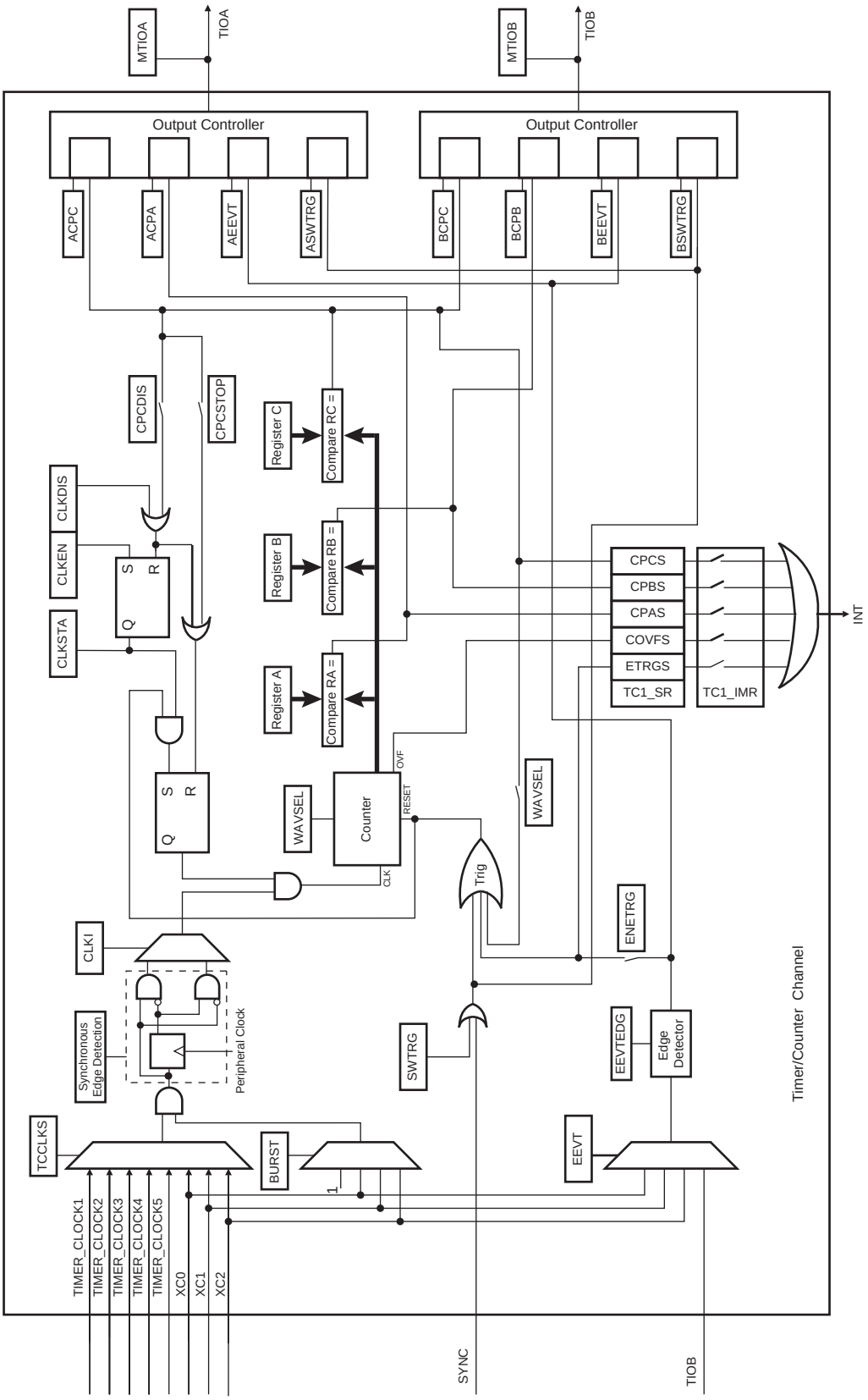
24.6.12 Waveform Selection

Depending on the WAVSEL parameter in TC_CMR, the behavior of TC_CV varies.

With any selection, TC_RA, TC_RB and TC_RC can all be used as compare registers.

RA Compare is used to control the TIOAx output, RB Compare is used to control the TIOBx output (if correctly configured) and RC Compare is used to control TIOAx and/or TIOBx outputs.

Figure 24-7. Waveform Mode



24.6.12.1 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 24-8.

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 24-9.

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 24-8. WAVSEL = 00 without Trigger

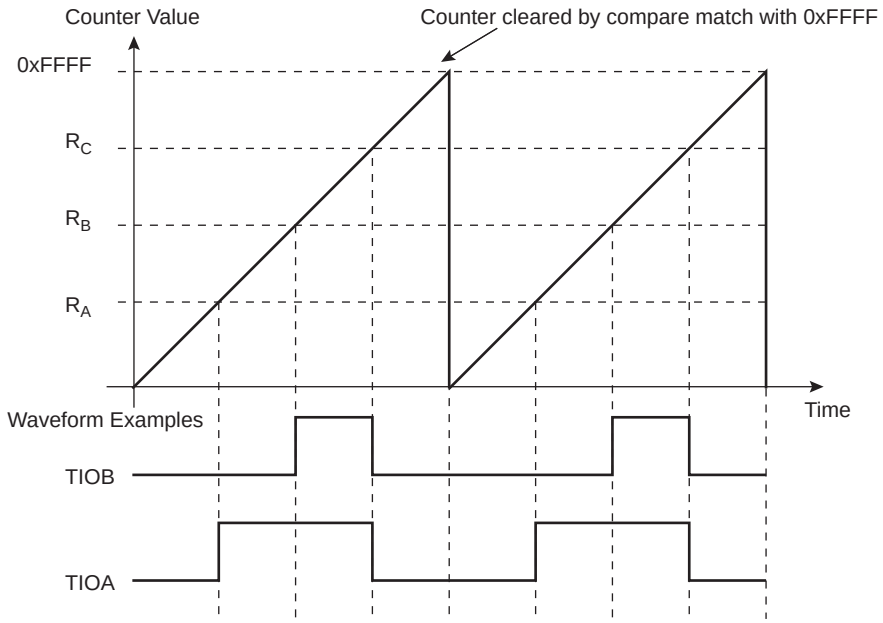
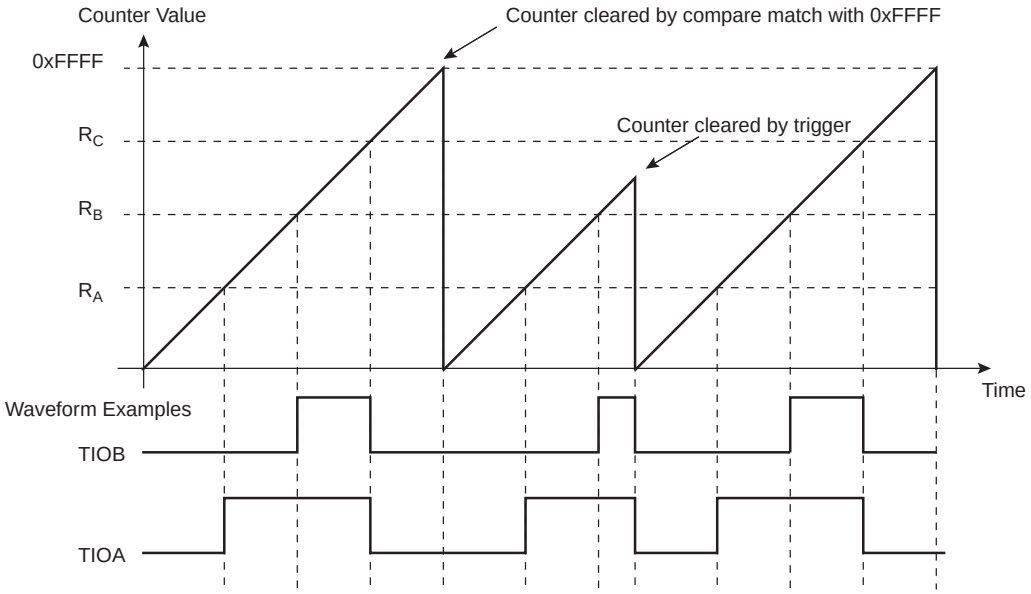


Figure 24-9. WAVSEL = 00 with Trigger



24.6.12.2 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 24-10.

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 24-11.

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 24-10. WAVSEL = 10 without Trigger

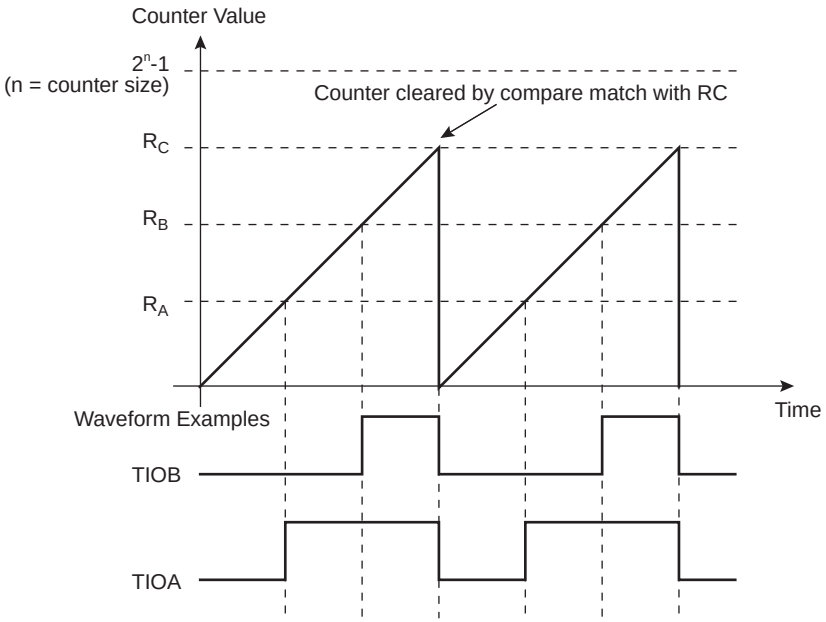
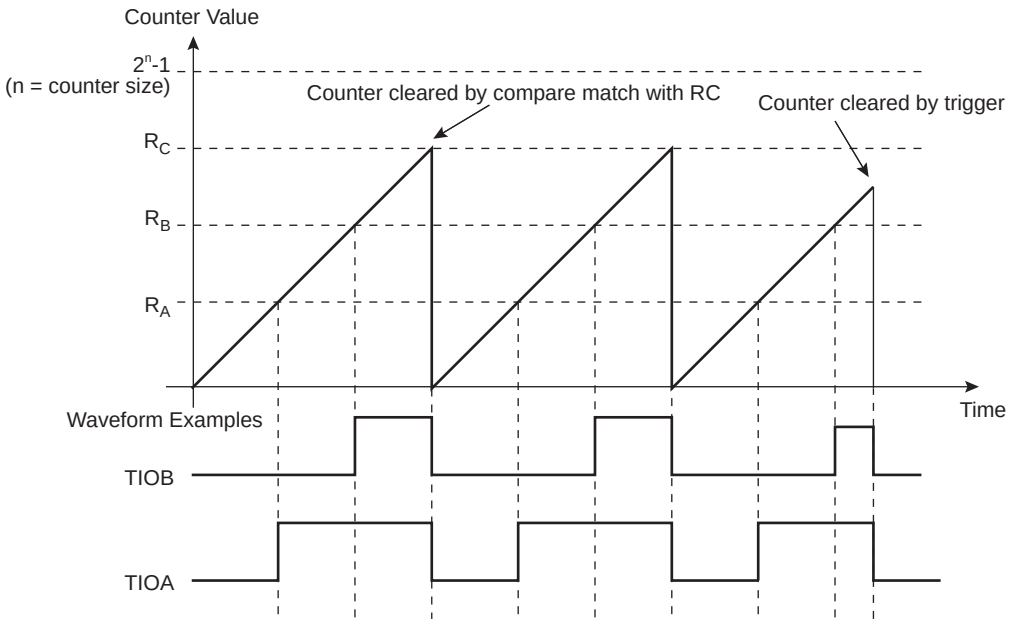


Figure 24-11. WAVSEL = 10 with Trigger



24.6.12.3 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 24-12.

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 24-13.

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 24-12. WAVSEL = 01 without Trigger

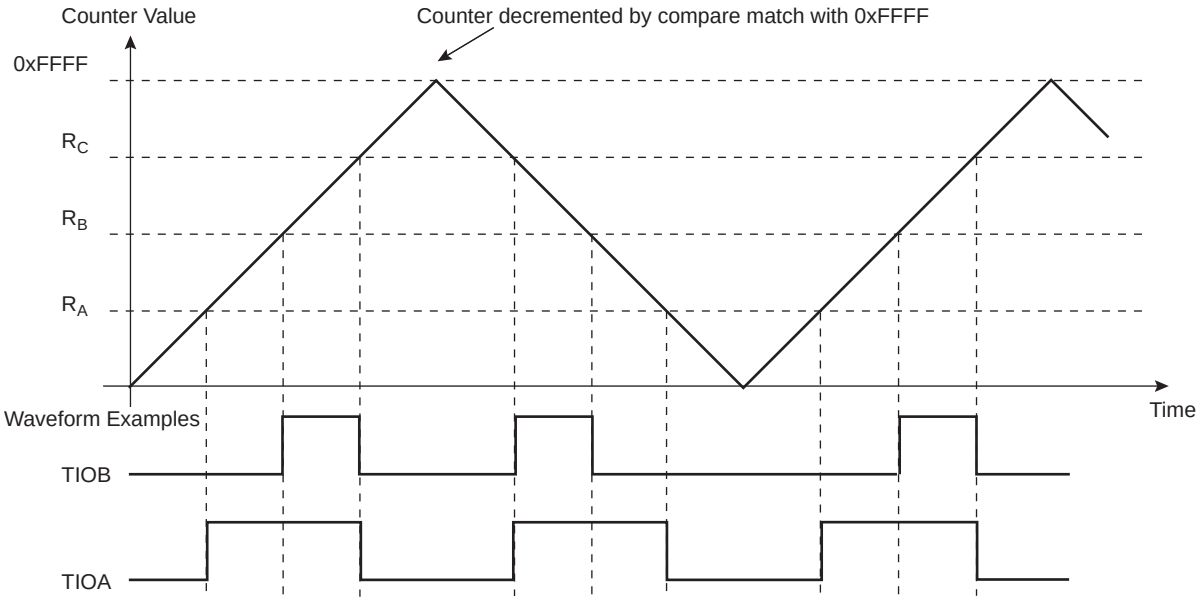
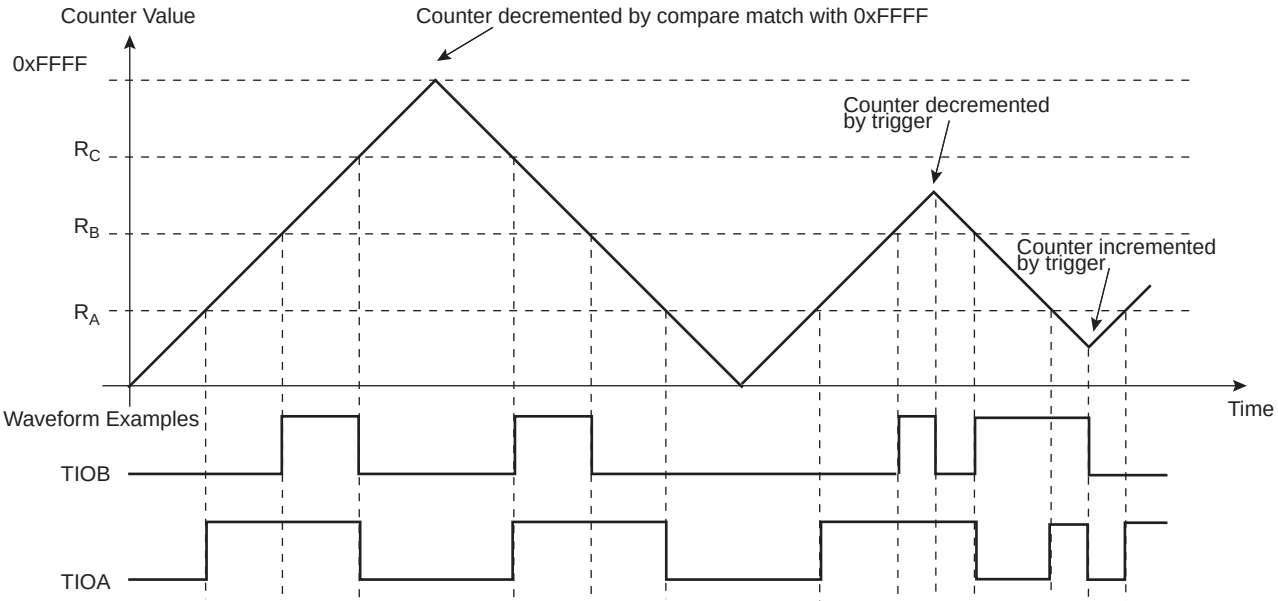


Figure 24-13. WAVSEL = 01 with Trigger



24.6.12.4 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 24-14.

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 24-15.

RC Compare can stop the counter clock (CPCSTOP = 1) and/or disable the counter clock (CPCDIS = 1).

Figure 24-14. WAVSEL = 11 without Trigger

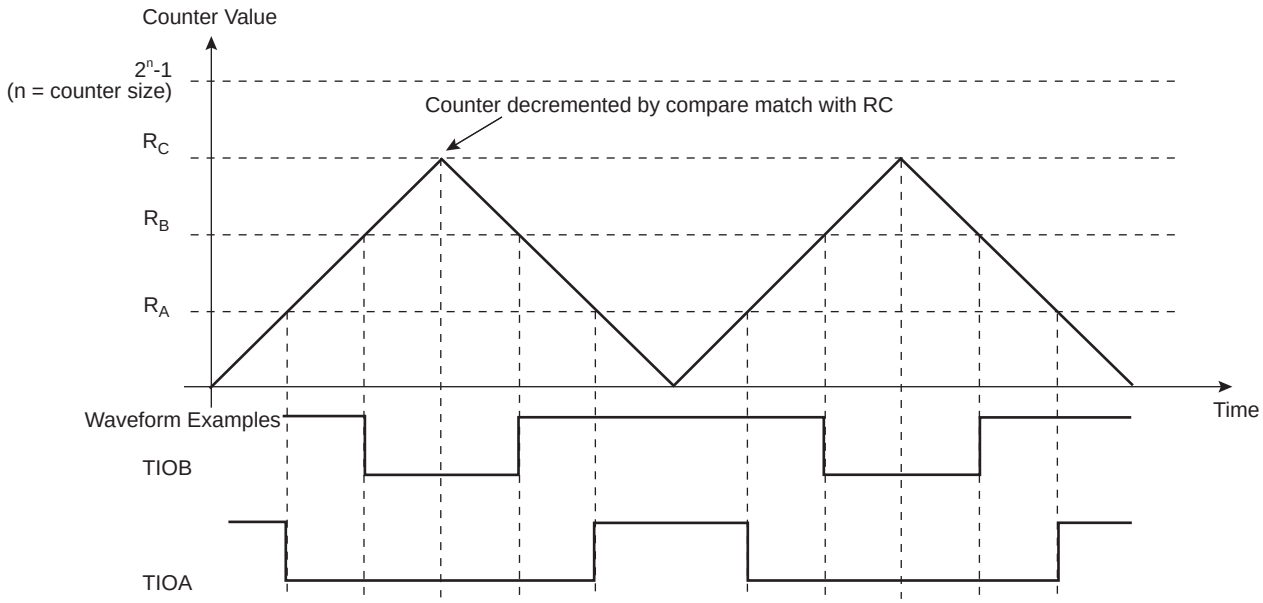
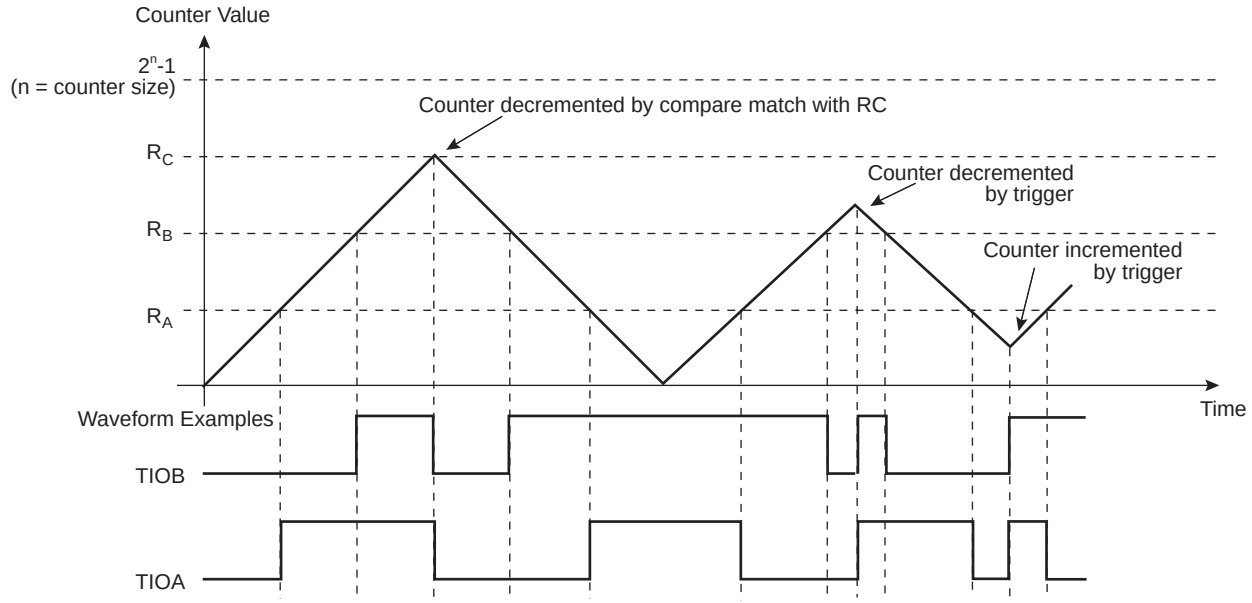


Figure 24-15. WAVSEL = 11 with Trigger



24.6.13 External Event/Trigger Conditions

An external event can be programmed to be detected on one of the clock sources (XC0, XC1, XC2) or TIOBx. The external event selected can then be used as a trigger.

The EEVT parameter in TC_CMR selects the external trigger. The EEVTEDG parameter defines the trigger edge for each of the possible external triggers (rising, falling or both). If EEVTEDG is cleared (none), no external event is defined.

If TIOBx is defined as an external event signal (EEVT = 0), TIOBx is no longer used as an output and the compare register B is not used to generate waveforms and subsequently no IRQs. In this case the TC channel can only generate a waveform on TIOAx.

When an external event is defined, it can be used as a trigger by setting bit ENETRG in the TC_CMR.

As in Capture mode, the SYNC signal and the software trigger are also available as triggers. RC Compare can also be used as a trigger depending on the parameter WAVSEL.

24.6.14 Output Controller

The output controller defines the output level changes on TIOAx and TIOBx following an event. TIOBx Control is used only if TIOBx is defined as output (not as an external event).

The following events control TIOAx and TIOBx:

- Software Trigger
- External Event
- RC Compare

RA Compare controls TIOAx, and RB Compare controls TIOBx. Each of these events can be programmed to set, clear or toggle the output as defined in the corresponding parameter in TC_CMR.

24.6.15 2-bit Gray Up/Down Counter for Stepper Motor

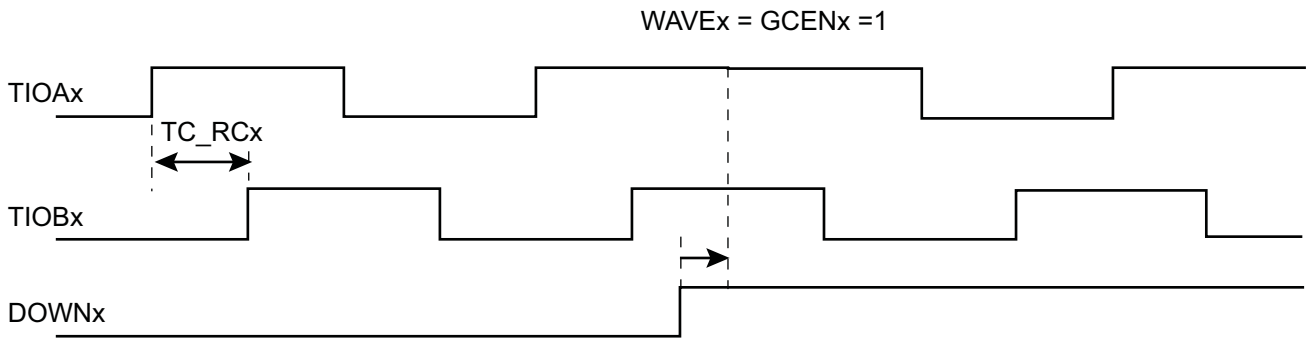
Each channel can be independently configured to generate a 2-bit Gray count waveform on corresponding TIOAx, TIOBx outputs by means of the GCEN bit in TC_SMMRx.

Up or Down count can be defined by writing bit DOWN in TC_SMMRx.

It is mandatory to configure the channel in Waveform mode in the TC_CMR.

The period of the counters can be programmed in TC_RCx.

Figure 24-16. 2-bit Gray Up/Down Counter



24.6.16 Register Write Protection

To prevent any single software error from corrupting TC behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the TC Write Protection Mode Register (TC_WPMR).

The Timer Counter clock of the first channel must be enabled to access TC_WPMR.

The following registers can be write-protected:

- TC Block Mode Register
- TC Channel Mode Register: Capture Mode
- TC Channel Mode Register: Waveform Mode
- TC Stepper Motor Mode Register
- TC Register A
- TC Register B
- TC Register C

24.7 Timer Counter (TC) User Interface

Table 24-6. Register Mapping

Offset[1]	Register	Name	Access	Reset
0x00 + channel * 0x40 + 0x00	Channel Control Register	TC_CCR	Write-only	–
0x00 + channel * 0x40 + 0x04	Channel Mode Register	TC_CMR	Read/Write	0
0x00 + channel * 0x40 + 0x08	Stepper Motor Mode Register	TC_SMMR	Read/Write	0
0x00 + channel * 0x40 + 0x0C	Register AB	TC_RAB	Read-only	0
0x00 + channel * 0x40 + 0x10	Counter Value	TC_CV	Read-only	0
0x00 + channel * 0x40 + 0x14	Register A	TC_RA	Read/Write[2]	0
0x00 + channel * 0x40 + 0x18	Register B	TC_RB	Read/Write[2]	0
0x00 + channel * 0x40 + 0x1C	Register C	TC_RC	Read/Write	0
0x00 + channel * 0x40 + 0x20	Status Register	TC_SR	Read-only	0
0x00 + channel * 0x40 + 0x24	Interrupt Enable Register	TC_IER	Write-only	–
0x00 + channel * 0x40 + 0x28	Interrupt Disable Register	TC_IDR	Write-only	–
0x00 + channel * 0x40 + 0x2C	Interrupt Mask Register	TC_IMR	Read-only	0
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 TC_CMRx.WAVE = 0

24.7.1 TC Channel Control Register

Name: TC_CCRx [x=0..2]

Address: 0x40010000 (0)[0], 0x40010040 (0)[1], 0x40010080 (0)[2], 0x40014000 (1)[0], 0x40014040 (1)[1], 0x40014080 (1)[2]

Access: Write-only

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

- **CLKEN: Counter Clock Enable Command**

0: No effect.

1: Enables the clock if CLKDIS is not 1.

- **CLKDIS: Counter Clock Disable Command**

0: No effect.

1: Disables the clock.

- **SWTRG: Software Trigger Command**

0: No effect.

1: A software trigger is performed: the counter is reset and the clock is started.

24.7.2 TC Channel Mode Register: Capture Mode

Name: TC_CMRx [x=0..2] (CAPTURE_MODE)

Address: 0x40010004 (0)[0], 0x40010044 (0)[1], 0x40010084 (0)[2], 0x40014004 (1)[0], 0x40014044 (1)[1], 0x40014084 (1)[2]

Access: Read/Write

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

23	22	21	20	19	18	17	16
–	SBSMPLR			LDRB		LDRA	

15	14	13	12	11	10	9	8
WAVE	CPCTRG	–	–	–	ABETRG	ETRGEDG	

7	6	5	4	3	2	1	0
LDBDIS	LDBSTOP	BURST		CLKI	TCCLKS		

This register can only be written if the WPEN bit is cleared in the TC Write Protection Mode Register.

- **TCCLKS: Clock Selection**

Value	Name	Description
0	TIMER_CLOCK1	Clock selected: internal MCK/2 clock signal (from PMC)
1	TIMER_CLOCK2	Clock selected: internal MCK/8 clock signal (from PMC)
2	TIMER_CLOCK3	Clock selected: internal MCK/32 clock signal (from PMC)
3	TIMER_CLOCK4	Clock selected: internal MCK/128 clock signal (from PMC)
4	TIMER_CLOCK5	Clock selected: internal PCK3 clock signal (from PMC)
5	XC0	Clock selected: XC0
6	XC1	Clock selected: XC1
7	XC2	Clock selected: XC2

- **CLKI: Clock Invert**

0: Counter is incremented on rising edge of the clock.

1: Counter is incremented on falling edge of the clock.

- **BURST: Burst Signal Selection**

Value	Name	Description
0	NONE	The clock is not gated by an external signal.
1	XC0	XC0 is ANDed with the selected clock.
2	XC1	XC1 is ANDed with the selected clock.
3	XC2	XC2 is ANDed with the selected clock.

- **LDBSTOP: Counter Clock Stopped with RB Loading**

0: Counter clock is not stopped when RB loading occurs.

1: Counter clock is stopped when RB loading occurs.

- **LDBDIS: Counter Clock Disable with RB Loading**

0: Counter clock is not disabled when RB loading occurs.

1: Counter clock is disabled when RB loading occurs.

- **ETRGEDG: External Trigger Edge Selection**

Value	Name	Description
0	NONE	The clock is not gated by an external signal.
1	RISING	Rising edge
2	FALLING	Falling edge
3	EDGE	Each edge

- **ABETRG: TIOAx or TIOBx External Trigger Selection**

0: TIOBx is used as an external trigger.

1: TIOAx is used as an external trigger.

- **CPCTRG: RC Compare Trigger Enable**

0: RC Compare has no effect on the counter and its clock.

1: RC Compare resets the counter and starts the counter clock.

- **WAVE: Waveform Mode**

0: Capture mode is enabled.

1: Capture mode is disabled (Waveform mode is enabled).

- **LDRA: RA Loading Edge Selection**

Value	Name	Description
0	NONE	None
1	RISING	Rising edge of TIOAx
2	FALLING	Falling edge of TIOAx
3	EDGE	Each edge of TIOAx

- **LDRB: RB Loading Edge Selection**

Value	Name	Description
0	NONE	None
1	RISING	Rising edge of TIOAx
2	FALLING	Falling edge of TIOAx
3	EDGE	Each edge of TIOAx

- **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

24.7.3 TC Channel Mode Register: Waveform Mode

Name: TC_CMRx [x=0..2] (WAVEFORM_MODE)

Address: 0x40010004 (0)[0], 0x40010044 (0)[1], 0x40010084 (0)[2], 0x40014004 (1)[0], 0x40014044 (1)[1], 0x40014084 (1)[2]

Access: Read/Write

31	30	29	28	27	26	25	24
BSWTRG		BEEVT		BCPC		BCPB	
23	22	21	20	19	18	17	16
ASWTRG		AEEVT		ACPC		ACPA	
15	14	13	12	11	10	9	8
WAVE	WAVSEL		ENETRG	EEVT		EEVTEDG	
7	6	5	4	3	2	1	0
CPCDIS	CPCSTOP	BURST		CLKI	TCCLKS		

This register can only be written if the WPEN bit is cleared in the TC Write Protection Mode Register.

- **TCCLKS: Clock Selection**

Value	Name	Description
0	TIMER_CLOCK1	Clock selected: internal MCK/2 clock signal (from PMC)
1	TIMER_CLOCK2	Clock selected: internal MCK/8 clock signal (from PMC)
2	TIMER_CLOCK3	Clock selected: internal MCK/32 clock signal (from PMC)
3	TIMER_CLOCK4	Clock selected: internal MCK/128 clock signal (from PMC)
4	TIMER_CLOCK5	Clock selected: internal PCK3 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 TIOAx output and TIOBx if not used as input (trigger event input or other input used).

- **WAVSEL: Waveform Selection**

Value	Name	Description
0	UP	UP mode without automatic trigger on RC Compare
1	UPDOWN	UPDOWN mode without automatic trigger on RC Compare
2	UP_RC	UP mode with automatic trigger on RC Compare
3	UPDOWN_RC	UPDOWN mode with automatic trigger on RC Compare

- **WAVE: Waveform Mode**

0: Waveform mode is disabled (Capture mode is enabled).

1: Waveform mode is enabled.

- **ACPA: RA Compare Effect on TIOAx**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **ACPC: RC Compare Effect on TIOAx**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **AEEVT: External Event Effect on TIOAx**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **ASWTRG: Software Trigger Effect on TIOAx**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **BCPB: RB Compare Effect on TIOBx**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **BCPC: RC Compare Effect on TIOBx**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **BEEVT: External Event Effect on TIOBx**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **BSWTRG: Software Trigger Effect on TIOBx**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

24.7.4 TC Stepper Motor Mode Register

Name: TC_SMMRx [x=0..2]

Address: 0x40010008 (0)[0], 0x40010048 (0)[1], 0x40010088 (0)[2], 0x40014008 (1)[0], 0x40014048 (1)[1], 0x40014088 (1)[2]

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the TC Write Protection Mode Register.

- **GCEN: Gray Count Enable**

0: TIOAx [x=0..2] and TIOBx [x=0..2] are driven by internal counter of channel x.

1: TIOAx [x=0..2] and TIOBx [x=0..2] are driven by a 2-bit Gray counter.

- **DOWN: Down Count**

0: Up counter.

1: Down counter.

24.7.5 TC Register AB

Name: TC_RABx [x=0..2]

Address: 0x4001000C (0)[0], 0x4001004C (0)[1], 0x4001008C (0)[2], 0x4001400C (1)[0], 0x4001404C (1)[1], 0x4001408C (1)[2]

Access: Read-only

31	30	29	28	27	26	25	24
RAB							

23	22	21	20	19	18	17	16
RAB							

15	14	13	12	11	10	9	8
RAB							

7	6	5	4	3	2	1	0
RAB							

- **RAB: Register A or Register B**

RAB contains the next unread capture Register A or Register B value in real time. It is usually read by the DMA after a request due to a valid load edge on TIOAx.

When DMA is used, the RAB register address must be configured as source address of the transfer.

24.7.6 TC Counter Value Register

Name: TC_CVx [x=0..2]

Address: 0x40010010 (0)[0], 0x40010050 (0)[1], 0x40010090 (0)[2], 0x40014010 (1)[0], 0x40014050 (1)[1], 0x40014090 (1)[2]

Access: Read-only

31	30	29	28	27	26	25	24
CV							

23	22	21	20	19	18	17	16
CV							

15	14	13	12	11	10	9	8
CV							

7	6	5	4	3	2	1	0
CV							

- **CV: Counter Value**

CV contains the counter value in real time.

IMPORTANT: For 16-bit channels, CV field size is limited to register bits 15:0.

24.7.7 TC Register A

Name: TC_RAx [x=0..2]

Address: 0x40010014 (0)[0], 0x40010054 (0)[1], 0x40010094 (0)[2], 0x40014014 (1)[0], 0x40014054 (1)[1], 0x40014094 (1)[2]

Access: Read-only if TC_CMRx.WAVE = 0, Read/Write if TC_CMRx.WAVE = 1

31	30	29	28	27	26	25	24
RA							

23	22	21	20	19	18	17	16
RA							

15	14	13	12	11	10	9	8
RA							

7	6	5	4	3	2	1	0
RA							

This register can only be written if the WPEN bit is cleared in the TC Write Protection Mode Register.

- **RA: Register A**

RA contains the Register A value in real time.

IMPORTANT: For 16-bit channels, RA field size is limited to register bits 15:0.

24.7.8 TC Register B

Name: TC_RBx [x=0..2]

Address: 0x40010018 (0)[0], 0x40010058 (0)[1], 0x40010098 (0)[2], 0x40014018 (1)[0], 0x40014058 (1)[1], 0x40014098 (1)[2]

Access: Read-only if TC_CMRx.WAVE = 0, Read/Write if TC_CMRx.WAVE = 1

31	30	29	28	27	26	25	24
RB							

23	22	21	20	19	18	17	16
RB							

15	14	13	12	11	10	9	8
RB							

7	6	5	4	3	2	1	0
RB							

This register can only be written if the WPEN bit is cleared in the TC Write Protection Mode Register.

- **RB: Register B**

RB contains the Register B value in real time.

IMPORTANT: For 16-bit channels, RB field size is limited to register bits 15:0.

24.7.9 TC Register C

Name: TC_RCx [x=0..2]

Address: 0x4001001C (0)[0], 0x4001005C (0)[1], 0x4001009C (0)[2], 0x4001401C (1)[0], 0x4001405C (1)[1], 0x4001409C (1)[2]

Access: Read/Write

31	30	29	28	27	26	25	24
RC							

23	22	21	20	19	18	17	16
RC							

15	14	13	12	11	10	9	8
RC							

7	6	5	4	3	2	1	0
RC							

This register can only be written if the WPEN bit is cleared in the TC Write Protection Mode Register.

- **RC: Register C**

RC contains the Register C value in real time.

IMPORTANT: For 16-bit channels, RC field size is limited to register bits 15:0.

24.7.10 TC Status Register

Name: TC_SRx [x=0..2]

Address: 0x40010020 (0)[0], 0x40010060 (0)[1], 0x400100A0 (0)[2], 0x40014020 (1)[0], 0x40014060 (1)[1], 0x400140A0 (1)[2]

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	MTIOB	MTIOA	CLKSTA
15	14	13	12	11	10	9	8
–	–	–	–	–	–	RXBUFF	ENDRX
7	6	5	4	3	2	1	0
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow Status (cleared on read)**

0: No counter overflow has occurred since the last read of the Status Register.

1: A counter overflow has occurred since the last read of the Status Register.

- **LOVRS: Load Overrun Status (cleared on read)**

0: Load overrun has not occurred since the last read of the Status Register or TC_CMRx.WAVE = 1.

1: RA or RB have been loaded at least twice without any read of the corresponding register since the last read of the Status Register, if TC_CMRx.WAVE = 0.

- **CPAS: RA Compare Status (cleared on read)**

0: RA Compare has not occurred since the last read of the Status Register or TC_CMRx.WAVE = 0.

1: RA Compare has occurred since the last read of the Status Register, if TC_CMRx.WAVE = 1.

- **CPBS: RB Compare Status (cleared on read)**

0: RB Compare has not occurred since the last read of the Status Register or TC_CMRx.WAVE = 0.

1: RB Compare has occurred since the last read of the Status Register, if TC_CMRx.WAVE = 1.

- **CPCS: RC Compare Status (cleared on read)**

0: RC Compare has not occurred since the last read of the Status Register.

1: RC Compare has occurred since the last read of the Status Register.

- **LDRAS: RA Loading Status (cleared on read)**

0: RA Load has not occurred since the last read of the Status Register or TC_CMRx.WAVE = 1.

1: RA Load has occurred since the last read of the Status Register, if TC_CMRx.WAVE = 0.

- **LDRBS: RB Loading Status (cleared on read)**

0: RB Load has not occurred since the last read of the Status Register or TC_CMRx.WAVE = 1.

1: RB Load has occurred since the last read of the Status Register, if TC_CMRx.WAVE = 0.

- **ETRGS: External Trigger Status (cleared on read)**

0: External trigger has not occurred since the last read of the Status Register.

1: External trigger has occurred since the last read of the Status Register.

- **ENDRX: End of Receiver Transfer (cleared by writing TC_RCR or TC_RNCR)**

0: The Receive Counter Register has not reached 0 since the last write in TC_RCR[(1)] or TC_RNCR[(1)].

1: The Receive Counter Register has reached 0 since the last write in TC_RCR or TC_RNCR.

- **RXBUFF: Reception Buffer Full (cleared by writing TC_RCR or TC_RNCR)**

0: TC_RCR or TC_RNCR have a value other than 0.

1: Both TC_RCR and TC_RNCR have a value of 0.

Note: 1. TC_RCR and TC_RNCR are PDC registers.

- **CLKSTA: Clock Enabling Status**

0: Clock is disabled.

1: Clock is enabled.

- **MTIOA: TIOAx Mirror**

0: TIOAx is low. If TC_CMRx.WAVE = 0, this means that TIOAx pin is low. If TC_CMRx.WAVE = 1, this means that TIOAx is driven low.

1: TIOAx is high. If TC_CMRx.WAVE = 0, this means that TIOAx pin is high. If TC_CMRx.WAVE = 1, this means that TIOAx is driven high.

- **MTIOB: TIOBx Mirror**

0: TIOBx is low. If TC_CMRx.WAVE = 0, this means that TIOBx pin is low. If TC_CMRx.WAVE = 1, this means that TIOBx is driven low.

1: TIOBx is high. If TC_CMRx.WAVE = 0, this means that TIOBx pin is high. If TC_CMRx.WAVE = 1, this means that TIOBx is driven high.

24.7.11 TC Interrupt Enable Register

Name: TC_IERx [x=0..2]

Address: 0x40010024 (0)[0], 0x40010064 (0)[1], 0x400100A4 (0)[2], 0x40014024 (1)[0], 0x40014064 (1)[1], 0x400140A4 (1)[2]

Access: Write-only

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

24.7.12 TC Interrupt Disable Register

Name: TC_IDRx [x=0..2]

Address: 0x40010028 (0)[0], 0x40010068 (0)[1], 0x400100A8 (0)[2], 0x40014028 (1)[0], 0x40014068 (1)[1], 0x400140A8 (1)[2]

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	RXBUFF	ENDRX
7	6	5	4	3	2	1	0
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow**

0: No effect.

1: Disables the Counter Overflow Interrupt.

- **LOVRS: Load Overrun**

0: No effect.

1: Disables the Load Overrun Interrupt (if TC_CMRx.WAVE = 0).

- **CPAS: RA Compare**

0: No effect.

1: Disables the RA Compare Interrupt (if TC_CMRx.WAVE = 1).

- **CPBS: RB Compare**

0: No effect.

1: Disables the RB Compare Interrupt (if TC_CMRx.WAVE = 1).

- **CPCS: RC Compare**

0: No effect.

1: Disables the RC Compare Interrupt.

- **LDRAS: RA Loading**

0: No effect.

1: Disables the RA Load Interrupt (if TC_CMRx.WAVE = 0).

- **LDRBS: RB Loading**

0: No effect.

1: Disables the RB Load Interrupt (if TC_CMRx.WAVE = 0).

- **ETRGS: External Trigger**

0: No effect.

1: Disables the External Trigger Interrupt.

- **ENDRX: End of Receiver Transfer**

0: No effect.

1: Disables the PDC Receive End of Transfer Interrupt.

- **RXBUFF: Reception Buffer Full**

0: No effect.

1: Disables the PDC Receive Buffer Full Interrupt.

24.7.13 TC Interrupt Mask Register

Name: TC_IMRx [x=0..2]

Address: 0x4001002C (0)[0], 0x4001006C (0)[1], 0x400100AC (0)[2], 0x4001402C (1)[0], 0x4001406C (1)[1], 0x400140AC (1)[2]

Access: Read-only

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

24.7.14 TC Block Control Register

Name: TC_BCR

Address: 0x400100C0 (0), 0x400140C0 (1)

Access: Write-only

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

- **SYNC: Synchro Command**

0: No effect.

1: Asserts the SYNC signal which generates a software trigger simultaneously for each of the channels.

24.7.15 TC Block Mode Register

Name: TC_BMR

Address: 0x400100C4 (0), 0x400140C4 (1)

Access: Read/Write

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

24.7.16 TC Write Protection Mode Register

Name: TC_WPMR

Address: 0x400100E4 (0), 0x400140E4 (1)

Access: Read/Write

31	30	29	28	27	26	25	24
WPKEY							

23	22	21	20	19	18	17	16
WPKEY							

15	14	13	12	11	10	9	8
WPKEY							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WPEN

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x54494D (“TIM” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x54494D (“TIM” in ASCII).

The Timer Counter clock of the first channel must be enabled to access this register.

See Section 24.6.16 “Register Write Protection” for a list of registers that can be write-protected and Timer Counter clock conditions.

- **WPKEY: Write Protection Key**

Value	Name	Description
0x54494D	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

25. Supply Controller (SUPC)

25.1 Description

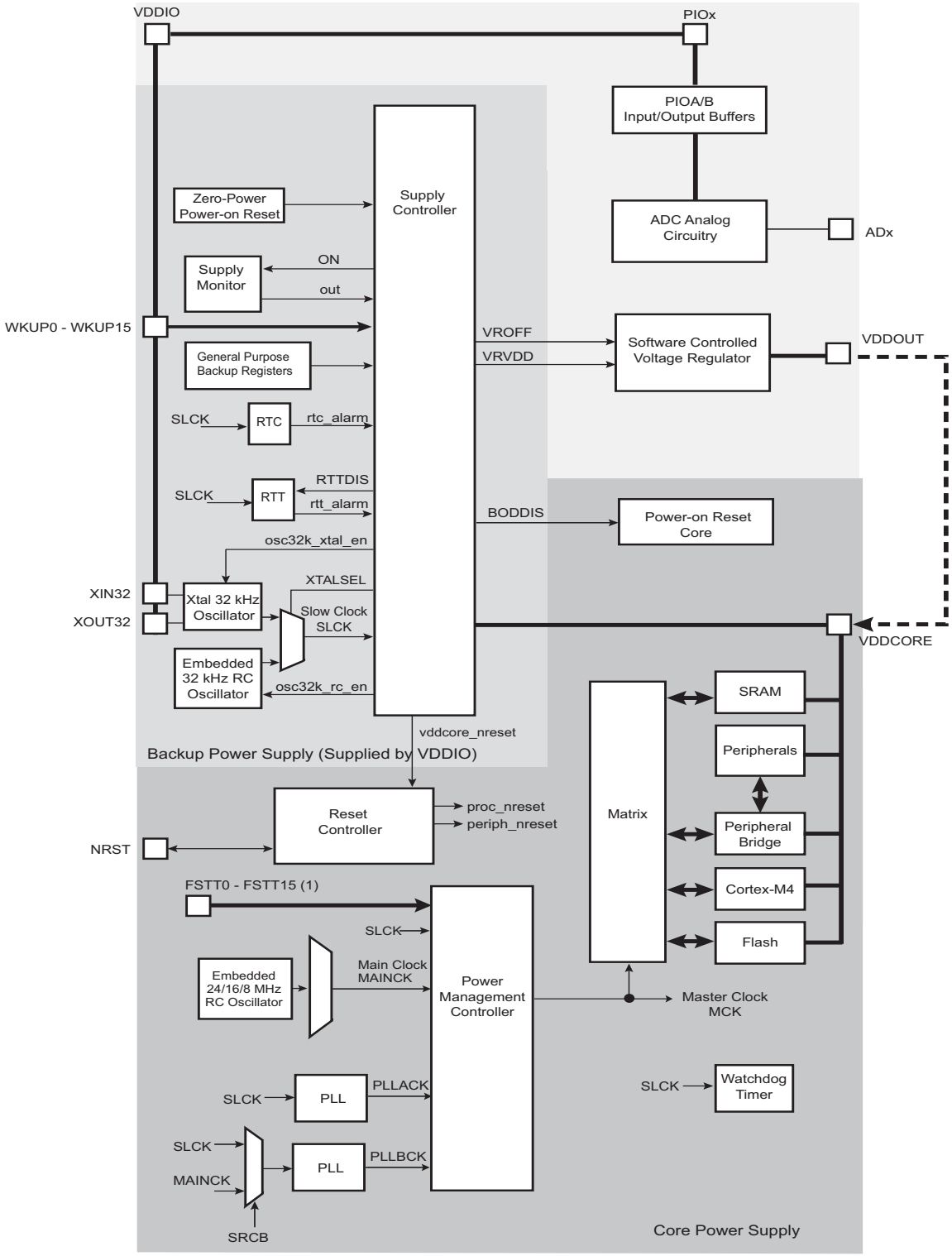
The Supply Controller (SUPC) controls the supply voltages of the system. The SUPC also generates the slow clock by selecting either the low-power RC oscillator or the low-power crystal oscillator.

25.2 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
- Supports Multiple Wakeup Sources for Exit from Backup Mode
  - 16 Wakeup Inputs with Programmable Debouncing
  - Real-Time Clock Alarm
  - Real-Time Timer Alarm
  - Supply Monitor Detection on VDDIO, with Programmable Scan Period and Voltage Threshold
- Low-power Tamper Detection on Two Inputs
- Anti-tampering by Immediate Clear of the General-purpose Backup Registers

25.3 Block Diagram

Figure 25-1. Supply Controller Block Diagram



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

25.4 Supply Controller Functional Description

25.4.1 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 startup 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.

When the Backup Low-Power mode is entered, the SUPC sequentially asserts the reset signal of the core power supply “vddcore_nreset” and disables the voltage regulator, in order to supply only the VDDIO power supply. In this mode, the current consumption is reduced to a few microamps for Backup part retention. Exit from this mode is possible on multiple wakeup sources including an event on WKUP pins, or an RTC/RTT alarm. To exit this mode, the SUPC operates in the same way as system startup.

25.4.2 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 startup time of the crystal oscillator (refer to the section “Electrical Characteristics” for details of 32 kHz crystal oscillator startup 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 to 1 prior to writing a 1 in bit XTALSEL.

25.4.3 Core Voltage Regulator Control/Backup Low-Power Mode

The SUPC can be used to control the embedded voltage regulator.

The voltage regulator automatically adapts its quiescent current depending on the required load current. More information can be found in the Electrical Characteristics.

The user can switch off the voltage regulator, and thus put the device in Backup mode, by writing a 1 to the VROFF bit in SUPC_CR.

Backup mode can also be entered by executing the WFE (Wait for Event) Cortex-M processor instruction with the SLEEPDEEP bit set to 1.

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

25.4.4 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 (refer to the “VDDIO Supply Monitor” characteristics in section “Electrical Characteristics”). 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, depending on the choice of the user. This is configured by programming the SMSMPL field in SUPC_SMMR.

By enabling the supply monitor for such reduced times, the typical supply monitor power consumption is divided, respectively, by factors of 2, 16 and 128, if the user does not require continuous monitoring of the VDDIO power supply.

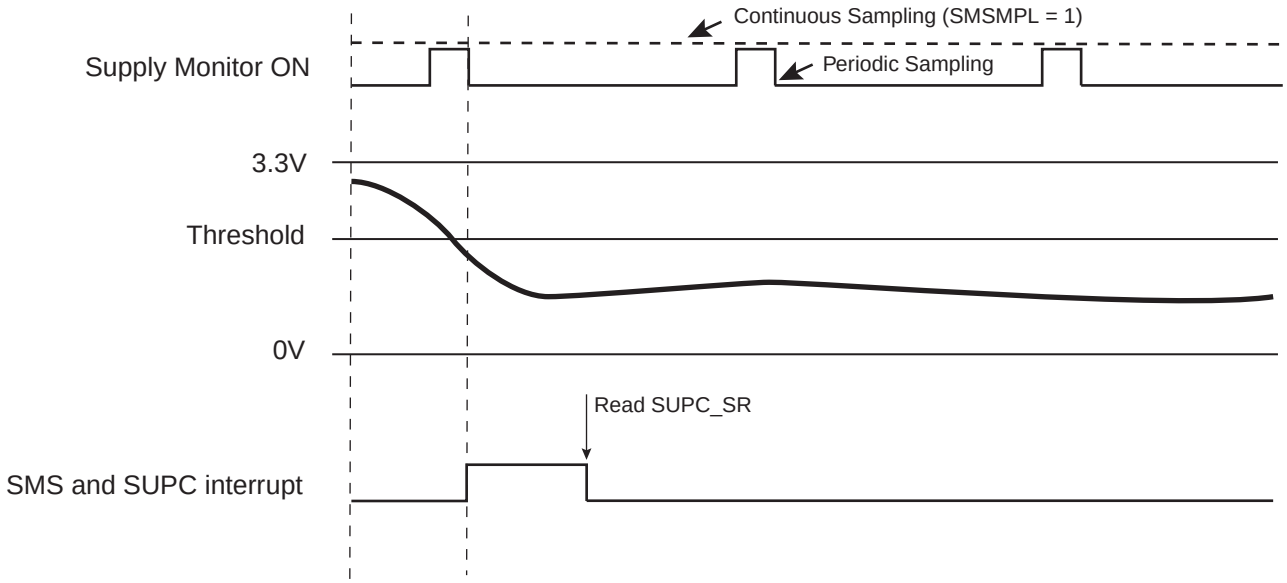
A supply monitor detection can generate a reset of the core power supply or a wakeup of the core power supply. Generating a core reset when a supply monitor detection occurs is enabled by writing the SMRSTEN bit to 1 in SUPC_SMMR.

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 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 25-2. Supply Monitor Status Bit and Associated Interrupt



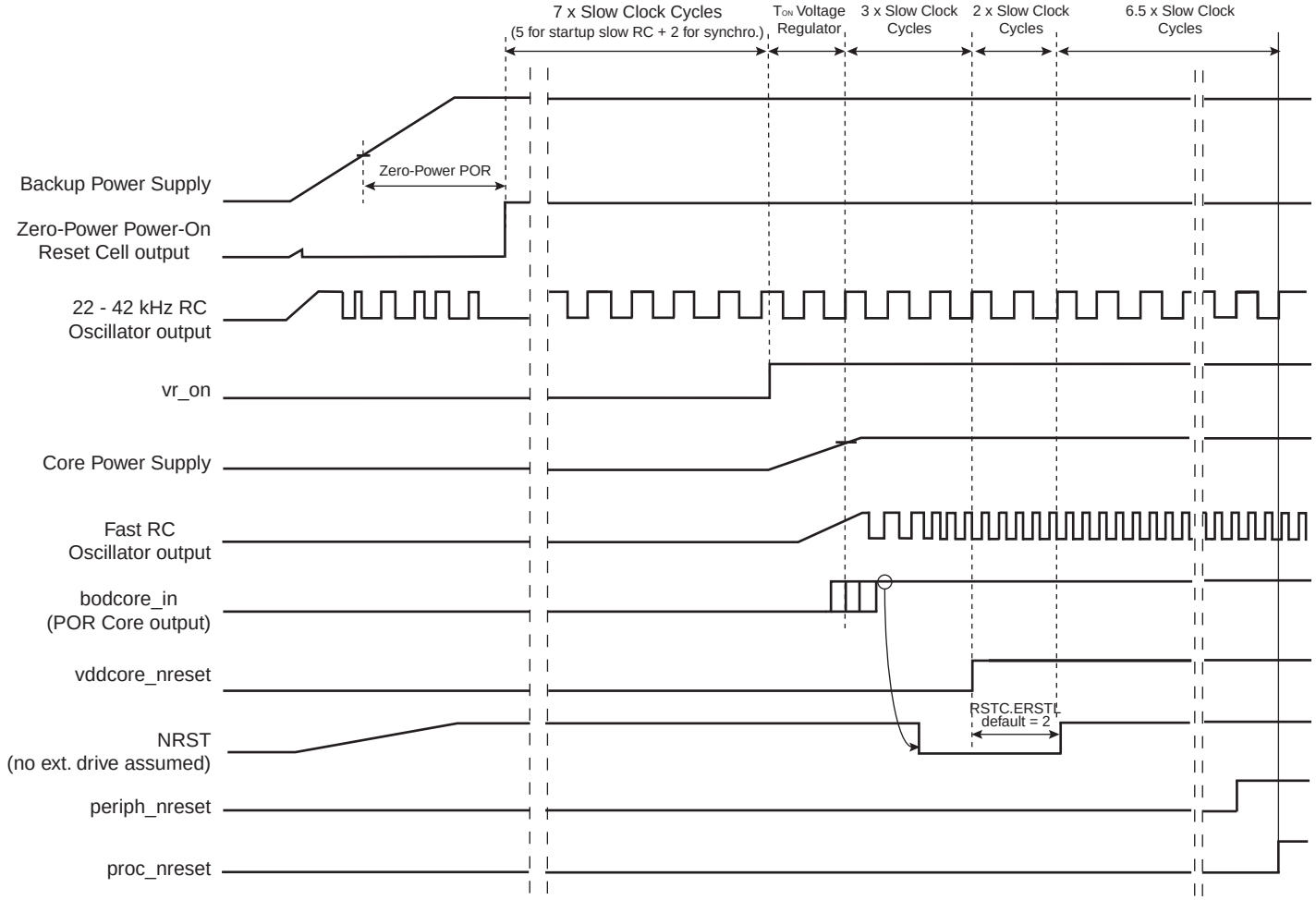
25.4.5 Power Supply Reset

25.4.5.1 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 valid for at least one slow clock cycle.

Figure 25-3. Raising the VDDIO Power Supply



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

25.4.6 Core Reset

The SUPC manages the vddcore_nreset signal to the Reset Controller, as described in Section 25.4.5 ”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

25.4.6.1 Supply Monitor Reset

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

25.4.6.2 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 one 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 Core Reset Enable) to 1 in SUPC_MR.

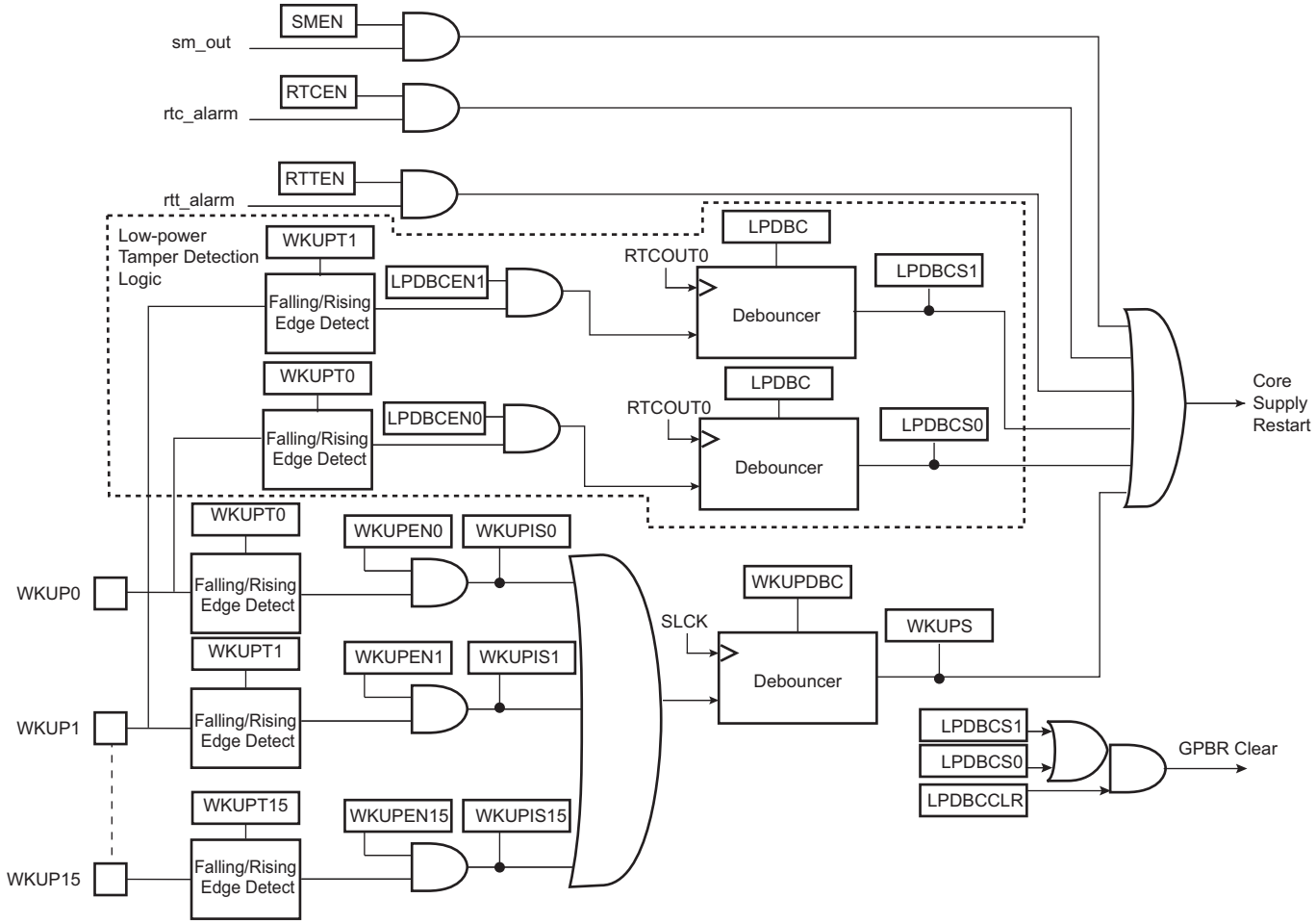
If BODRSTEN is set and the voltage regulation is lost (output voltage of the regulator too low), the vddcore_nreset signal is asserted for a minimum of one slow clock cycle and then released if bodcore_in has been reactivated. The BODRSTS bit is set in SUPC_SR so that the user can know the source of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

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

25.4.7 Wakeup Sources

The wakeup events allow the device to exit Backup mode. When a wakeup event is detected, the SUPC performs a sequence that automatically reenables the core power supply.

Figure 25-4. Wakeup Sources



25.4.7.1 Wakeup Inputs

The wakeup inputs, WKUPx, can be programmed to perform a wakeup of the core power supply. Each input can be enabled by writing a 1 to the corresponding bit, WKUPENx, in the Wakeup Inputs register (SUPC_WUIR). The wakeup level can be selected with the corresponding polarity bit, WKUPTx, also located in SUPC_WUIR.

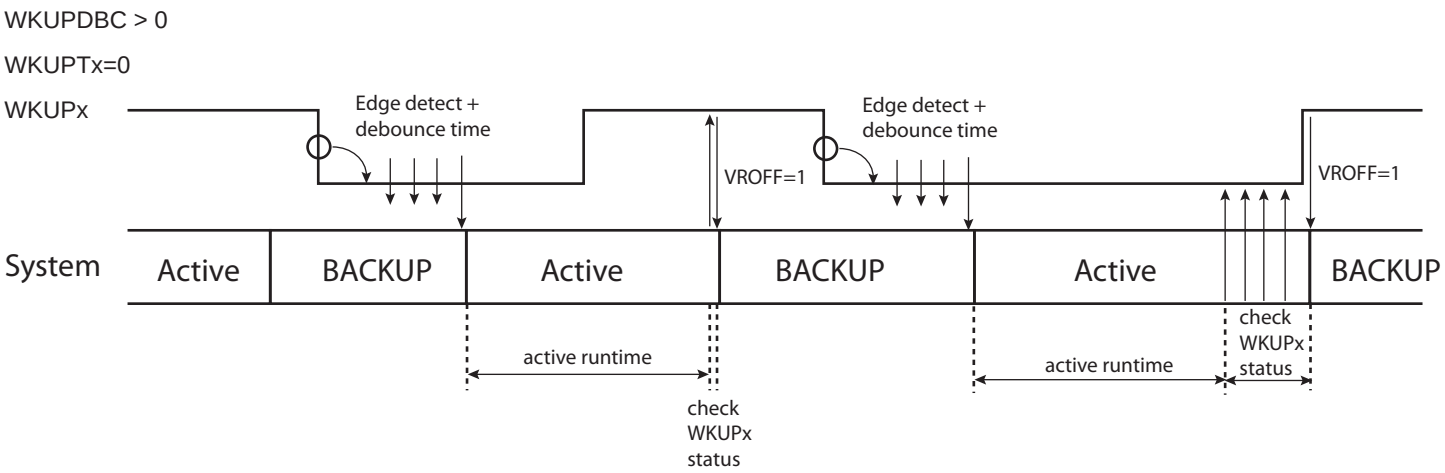
The resulting signals are wired-ORed to trigger a debounce counter, which is programmed with the WKUPDBC field in SUPC_WUMR. The WKUPDBC field selects a debouncing period of 3, 32, 512, 4,096 or 32,768 slow clock cycles. The duration of these periods corresponds, respectively, to about 100 µs, about 1 ms, about 16 ms, about 128 ms and about 1 second (for a typical slow clock frequency of 32 kHz). Programming WKUPDBC to 0x0 selects

an immediate wakeup, i.e., an enabled WKUP pin must be active according to its polarity during a minimum of one slow clock period to wake up the core power supply.

If an enabled WKUP pin is asserted for a duration longer than the debouncing period, a wakeup of the core power supply is started and the signals, WKUP0 to WKUPx as shown in Figure 25-4 "Wakeup Sources", are latched in SUPC_SR. This allows the user to identify the source of the wakeup. However, if a new wakeup condition occurs, the primary information is lost. No new wakeup can be detected since the primary wakeup condition has disappeared.

Before instructing the system to enter Backup mode, if the field WKUPDBC > 0, it must be checked that none of the WKUPx pins that are enabled for a wakeup (exit from Backup mode) holds an active polarity. This is checked by reading the pin status in the PIO Controller. If WKUPENx=1 and the pin WKUPx holds an active polarity, the system must not be instructed to enter Backup mode.

Figure 25-5. Entering and Exiting Backup Mode with a WKUP pin



25.4.7.2 Low-power Tamper Detection and Anti-Tampering

Low-power debouncer inputs (WKUP0, WKUP1) can be used for tamper detection.Separate debouncers are embedded, one for WKUP0 input, one for WKUP1 input.

The WKUP0 and/or WKUP1 inputs perform a system wakeup upon tamper detection. This is enabled by setting the LPDBCEN0/1 bit in the SUPC_WUMR.

WKUP0 and/or WKUP1 inputs can also be used when VDDCORE is powered to detect a tamper.

When the bit LPDBCENx is written to 1, WKUPx pins must not be configured to act as a debouncing source for the WKUPDBC counter (WKUPENx must be cleared in SUPC_WUIR).

Low-power tamper detection or debounce requires RTC output (RTCOUT0) to be configured to generate a duty cycle programmable pulse (i.e., OUT0 = 0x7 in RTC_MR) in order to create the sampling points of both debouncers. The sampling point is the falling edge of the RTCOUT0 waveform.

Figure 25-6 shows an example of an application where two tamper switches are used.

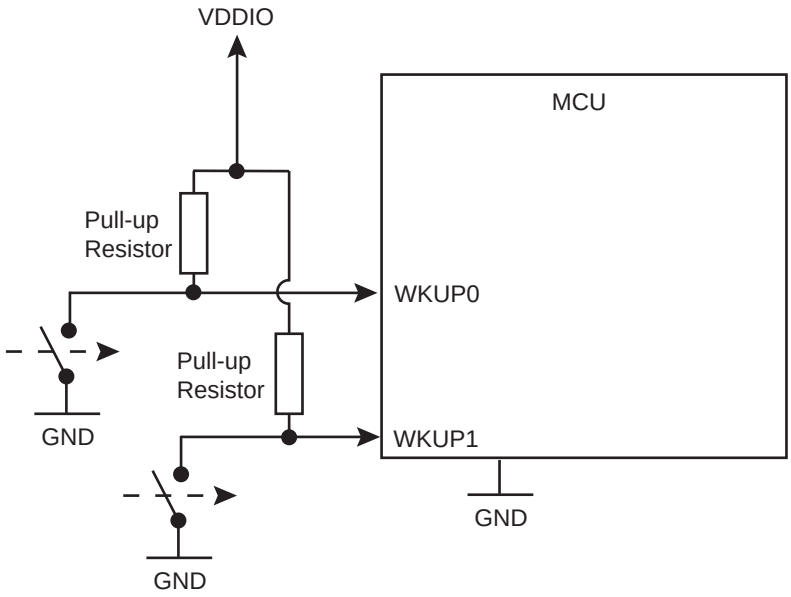
The debouncing period duration is configurable. The period is set for all debouncers (i.e., the duration cannot be adjusted for each debouncer). The number of successive identical samples to wake up the system can be configured from 2 up to 8 in the LPDBC field of SUPC_WUMR. The period of time between two samples can be configured by programming the TPERIOD field in the RTC_MR. Power parameters can be adjusted by modifying the period of time in the THIGH field in RTC_MR.

The wakeup polarity of the inputs can be independently configured by writing WKUPT0 and/ or WKUPT1 fields in SUPC_WUMR.

In order to determine which wakeup/tamper pin triggers the system wakeup, a status flag is associated for each low-power debouncer. These flags are read in SUPC_SR.

A debounce event (tamper detection) can perform an immediate clear (0 delay) on the first half the general-purpose backup registers (GPBR). The LPDBCCLR bit must be set in SUPC_WUMR.

Figure 25-6. Using WKUP Pins Without RTCOUTx Pins



25.4.8 Register Write Protection

To prevent any single software error from corrupting SUPC behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the System Controller Write Protection Mode Register (SYSC_WPMR).

The following registers can be write-protected:

- RSTC Mode Register
- RTT Mode Register
- RTT Alarm Register
- RTC Control Register
- RTC Mode Register
- RTC Time Alarm Register
- RTC Calendar Alarm Register
- General Purpose Backup Registers
- Supply Controller Control Register
- Supply Controller Supply Monitor Mode Register
- Supply Controller Mode Register

25.4.9 Register Bits in Backup Domain (VDDIO)

The following configuration registers are physically located in the product backup domain:

- RSTC Mode Register
- RTT Mode Register
- RTT Alarm Register
- RTC Control Register
- RTC Mode Register
- RTC Time Alarm Register
- RTC Calendar Alarm Register
- General Purpose Backup Registers
- Supply Controller Control Register
- Supply Controller Supply Monitor Mode Register
- Supply Controller Mode Register
- Supply Controller Wakeup Mode Register
- System Controller Wakeup Inputs Register
- Supply Controller Status Register

25.5 Supply Controller (SUPC) User Interface

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

25.5.1 System Controller (SYSC) User Interface

Table 25-1. System Controller Registers

Offset	System Controller Peripheral	Name
0x00–0x0C	Reset Controller	RSTC
0x10–0x2C	Supply Controller	SUPC
0x30–0x40	Real-Time Timer	RTT
0x50–0x5C	Watchdog Timer	WDT
0x60–0x8C	Real-Time Clock	RTC
0x90–0xDC	General-Purpose Backup Register	GPBR
0xE0	Reserved	–
0xE4	Write Protection Mode Register	SYSC_WPMR
0xE8–0xF8	Reserved	–
0x100–0x12C	Reserved	–
0x130	Real-Time Clock	RTC
0x134–0x1FC	Reserved	–

25.5.2 Supply Controller (SUPC) User Interface

Table 25-2. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Supply Controller Control Register	SUPC_CR	Write-only	–
0x04	Supply Controller Supply Monitor Mode Register	SUPC_SMMR	Read/Write	0x0000_0000
0x08	Supply Controller Mode Register	SUPC_MR	Read/Write	0x00E0_5400
0x0C	Supply Controller Wakeup Mode Register	SUPC_WUMR	Read/Write	0x0000_0000
0x10	Supply Controller Wakeup Inputs Register	SUPC_WUIR	Read/Write	0x0000_0000
0x14	Supply Controller Status Register	SUPC_SR	Read-only	0x0000_0000
0x18	Reserved	–	–	–
0x1C	Supply Controller Power Mode Register	SUPC_PWMR	Read/Write	0x00FF_180A
0x20–0x2C	Reserved	–	–	–

25.5.3 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	VROFF	–	–

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

- **VROFF: Voltage Regulator Off**

0 (NO_EFFECT): No effect.

1 (STOP_VREG): If KEY is correct, asserts the system reset signal and stops the voltage regulator.

- **XTALSEL: Crystal Oscillator Select**

0 (NO_EFFECT): No effect.

1 (CRYSTAL_SEL): If KEY is correct, switches the slow clock on the crystal oscillator output.

- **KEY: Password**

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

25.5.4 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			

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

- **SMTH: Supply Monitor Threshold**

Selects the threshold voltage of the supply monitor. Refer to the section “Electrical Characteristics” for voltage values.

- **SMSMPL: Supply Monitor Sampling Period**

Value	Name	Description
0	SMD	Supply Monitor disabled
1	CSM	Continuous Supply Monitor
2	32SLCK	Supply Monitor enables one SLCK period every 32 SLCK periods
3	256SLCK	Supply Monitor enables one SLCK period every 256 SLCK periods
4	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.

25.5.5 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
ONEA	CTPSWITCH	CDPSWITCH	OSCBYPASS	–	–	–	–

15	14	13	12	11	10	9	8
–	ONE	BODDIS	BODRSTEN	VRVDD			VDDSEL

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: Bits 23 and 14 must always be written to ‘1’.

- **VDDSEL: VRVDD Field Selection**

0 (FACTORY): If SUPC_PWMR.ECPWRS = 0, the voltage regulator output value is the factory-programmed value. If SUPC_PWMR.ECPWRS = 1, the voltage regulator value is managed by the SUPC_PWMR.ECPWRx bits.

1 (USER_VRVDD): The voltage regulator output value is defined by the value programmed in the field VRVDD.

- **VRVDD: Voltage Regulator Output Voltage Selection**

Refer to the section “Electrical Characteristics” for details.

- **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.

- **ONE: This bit must always be set to 1.**

This bit must always be set to 1.

- **OSCBYPASS: Oscillator Bypass**

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

1 (BYPASS): The 32 kHz crystal oscillator is bypassed if XTALSEL = 1. OSCBYPASS must be set prior to write XTALSEL = 1.

- **CDPSWITCH: Cache Data SRAM Power Switch**

0 (OFF): The cache data SRAM is not powered.

1 (ON): The cache data SRAM is powered.

Refer to “Internal SRAM” in section “Memories”.

- **CTPSWITCH: Cache Tag SRAM Power Switch**

0 (OFF): The cache tag SRAM is not powered.

1 (ON): The cache tag SRAM is powered.

Refer to “Internal SRAM” in section “Memories”.

- **ONEA: This bit must always be set to 1.**

This bit must always be set to 1.

- **KEY: Password Key**

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

25.5.6 Supply Controller Wakeup Mode Register

Name: SUPC_WUMR

Address: 0x400E141C

Access: Read/Write

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

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

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

7	6	5	4	3	2	1	0
LPDBCCLR	LPDBCEN1	LPDBCEN0	–	RTCEN	RTTEN	SMEN	–

- **SMEN: Supply Monitor Wakeup Enable**

0 (NOT_ENABLE): The supply monitor detection has no wakeup effect.

1 (ENABLE): The supply monitor detection forces the wakeup of the core power supply.

- **RTTEN: Real-time Timer Wakeup Enable**

0 (NOT_ENABLE); The RTT alarm signal has no wakeup effect.

1 (ENABLE): The RTT alarm signal forces the wakeup of the core power supply.

- **RTCEN: Real-time Clock Wakeup Enable**

0 (NOT_ENABLE); The RTC alarm signal has no wakeup effect.

1 (ENABLE): The RTC alarm signal forces the wakeup of the core power supply.

- **LPDBCEN0: Low-power Debouncer Enable WKUP0**

0 (NOT_ENABLE): The WKUP0 input pin is not connected to the low-power debouncer.

1 (ENABLE): The WKUP0 input pin is connected to the low-power debouncer and forces a system wakeup.

- **LPDBCEN1: Low-power Debouncer Enable WKUP1**

0 (NOT_ENABLE): The WKUP1 input pin is not connected to the low-power debouncer.

1 (ENABLE): The WKUP1 input pin is connected to the low-power debouncer and forces a system wakeup.

- **LPDBCCLR: Low-power Debouncer Clear**

0 (NOT_ENABLE): A low-power debounce event does not create an immediate clear on the first half of GPBR registers.

1 (ENABLE): A low-power debounce event on WKUP0 or WKUP1 generates an immediate clear on the first half of GPBR registers.

- **WKUPDBC: Wakeup Inputs Debouncer Period**

Value	Name	Description
0	IMMEDIATE	Immediate, no debouncing, detected active at least on one Slow Clock edge.
1	3_SCLK	WKUPx shall be in its active state for at least 3 SLCK periods
2	32_SCLK	WKUPx shall be in its active state for at least 32 SLCK periods
3	512_SCLK	WKUPx shall be in its active state for at least 512 SLCK periods
4	4096_SCLK	WKUPx shall be in its active state for at least 4,096 SLCK periods
5	32768_SCLK	WKUPx shall be in its active state for at least 32,768 SLCK periods

- **LPDBC: Low-power Debouncer Period**

Value	Name	Description
0	DISABLE	Disable the low-power debouncers.
1	2_RTCOUT0	WKUP0/1 in active state for at least 2 RTCOUTx periods
2	3_RTCOUT0	WKUP0/1 in active state for at least 3 RTCOUTx periods
3	4_RTCOUT0	WKUP0/1 in active state for at least 4 RTCOUTx periods
4	5_RTCOUT0	WKUP0/1 in active state for at least 5 RTCOUTx periods
5	6_RTCOUT0	WKUP0/1 in active state for at least 6 RTCOUTx periods
6	7_RTCOUT0	WKUP0/1 in active state for at least 7 RTCOUTx periods
7	8_RTCOUT0	WKUP0/1 in active state for at least 8 RTCOUTx periods

25.5.7 System Controller Wakeup Inputs Register

Name: SUPC_WUIR

Address: 0x400E1420

Access: Read/Write

31	30	29	28	27	26	25	24
WKUPT15	WKUPT14	WKUPT13	WKUPT12	WKUPT11	WKUPT10	WKUPT9	WKUPT8
23	22	21	20	19	18	17	16
WKUPT7	WKUPT6	WKUPT5	WKUPT4	WKUPT3	WKUPT2	WKUPT1	WKUPT0
15	14	13	12	11	10	9	8
WKUPEN15	WKUPEN14	WKUPEN13	WKUPEN12	WKUPEN11	WKUPEN10	WKUPEN9	WKUPEN8
7	6	5	4	3	2	1	0
WKUPEN7	WKUPEN6	WKUPEN5	WKUPEN4	WKUPEN3	WKUPEN2	WKUPEN1	WKUPEN0

- **WKUPEN0 - WKUPEN15: Wakeup Input Enable 0 to 15**

0 (DISABLE): The corresponding wakeup input has no wakeup effect.

1 (ENABLE): The corresponding wakeup input forces the wakeup of the core power supply.

- **WKUPT0 - WKUPT15: Wakeup Input Type 0 to 15**

0 (LOW): A falling edge followed by a low level for a period defined by WKUPDBC on the corresponding wakeup input forces the wakeup of the core power supply.

1 (HIGH): A rising edge followed by a high level for a period defined by WKUPDBC on the corresponding wakeup input forces the wakeup of the core power supply.

25.5.8 Supply Controller Status Register

Name: SUPC_SR

Address: 0x400E1424

Access: Read-only

31	30	29	28	27	26	25	24
WKUPIS15	WKUPIS14	WKUPIS13	WKUPIS12	WKUPIS11	WKUPIS10	WKUPIS9	WKUPIS8
23	22	21	20	19	18	17	16
WKUPIS7	WKUPIS6	WKUPIS5	WKUPIS4	WKUPIS3	WKUPIS2	WKUPIS1	WKUPIS0
15	14	13	12	11	10	9	8
–	LPDBCS1	LPDBCS0	–	–	–	–	–
7	6	5	4	3	2	1	0
OSCSEL	SMOS	SMS	SMRSTS	BODRSTS	–	WKUPS	–

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

- **WKUPS: WKUP Wakeup Status (cleared on read)**

0 (NO): No wakeup due to the assertion of the WKUP pins has occurred since the last read of SUPC_SR.

1 (PRESENT): At least one wakeup due to the assertion of the WKUP pins has occurred since the last read of SUPC_SR.

- **BODRSTS: Brownout Detector Reset Status (cleared on read)**

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

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

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

- **SMRSTS: Supply Monitor Reset Status (cleared on read)**

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

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

- **SMS: Supply Monitor Status (cleared on read)**

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

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

- **SMOS: Supply Monitor Output Status**

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

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

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

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

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

- **LPDBCS0: Low-power Debouncer Wakeup Status on WKUP0 (cleared on read)**

0 (NO): No wakeup due to the assertion of the WKUP0 pin has occurred since the last read of SUPC_SR.

1 (PRESENT): At least one wakeup due to the assertion of the WKUP0 pin has occurred since the last read of SUPC_SR.

- **LPDBCS1: Low-power Debouncer Wakeup Status on WKUP1 (cleared on read)**

0 (NO): No wakeup due to the assertion of the WKUP1 pin has occurred since the last read of SUPC_SR.

1 (PRESENT): At least one wakeup due to the assertion of the WKUP1 pin has occurred since the last read of SUPC_SR.

- **WKUPIS0-WKUPIS15: WKUP Input Status 0 to 15 (cleared on read)**

0 (DISABLED): The corresponding wakeup input is disabled, or was inactive at the time the debouncer triggered a wakeup event.

1 (ENABLED): The corresponding wakeup input was active at the time the debouncer triggered a wakeup event since the last read of SUPC_SR.

25.5.9 Supply Controller Power Mode Register

Name: SUPC_PWMR

Address: 0x400E142C

Access: Read/Write

31	30	29	28	27	26	25	24
KEY							

23	22	21	20	19	18	17	16
DPRAMON	SRAM6ON	SRAM5ON	SRAM4ON	SRAM3ON	SRAM2ON	SRAM1ON	SRAM0ON

15	14	13	12	11	10	9	8
–	–	–	ECPWR3	ECPWR2	ECPWR1	ECPWR0	ECPWRS

7	6	5	4	3	2	1	0
STUPTIME	–	–	LPOWER3	LPOWER2	LPOWER1	LPOWER0	LPOWERS

- **LPOWERS: Low Power Value Selection**

0 (FACTORY): The trimming value applied to the regulator when the device is in Wait mode. This value is factory-defined.

1 (USER): The trimming value applied to the regulator is defined by the value programmed in the LPOWERx bits.

- **LPOWER0 - LPOWER3: Low Power Value**

The regulator trimming value that allows the device to run in Wait mode. Must be read by customer in the Flash unique identifier page. Refer to “Calibration Bits” in section “Memories”.

- **STUPTIME: Startup Time when Resuming from Wait Mode**

0 (FAST): Fast startup.

1 (SLOW): Slow startup.

Refer to section “Electrical Characteristics”.

- **ECPWRS: Enhanced Custom Power Value Selection**

0 (FACTORY): The trimming value applied to the regulator when the device is in Active mode is factory-defined when SUPC_MR.VDDSEL = 0. If SUPC_MR.VDDSEL = 1, the SUPC_MR.VRVDD manages the regulator output value.

1 (USER): If SUPC_MR.VDDSEL = 1, the trimming value applied to the regulator is defined by the value programmed in ECPWRx bits.

- **ECPWRx: Enhanced Custom Power Value**

The regulator trimming value that allows the device to run in Active mode. Must be read by customer in the Flash unique identifier page. Refer to “Unique Identifier in section “Memories”.

- **SRAMxON: SRAM Power Control**

0 (OFF): SRAMx is not powered.

1 (ON): SRAMx is powered.

Refer to “Internal SRAM” in section “Memories”.

- **DPRAMON: Dual-port RAM Power Control**

0 (OFF): USB dual-port RAM is not powered.

1 (ON): USB dual-port RAM is powered.

- **KEY: Password Key**

Value	Name	Description
0x5A	PASSWD	Writing any other value in this field aborts the write operation. Always reads as 0.

25.5.10 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 25.4.8 ”Register Write Protection” for the list of registers that can be write-protected.

- **WPKEY: Write Protection Key**

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

26. Real-time Clock (RTC)

26.1 Description

The Real-time Clock (RTC) peripheral is designed for very low power consumption. For optimal functionality, the RTC requires an accurate external 32.768 kHz clock, which can be provided by a crystal oscillator.

It combines a complete time-of-day clock with alarm and a Gregorian or Persian calendar, complemented by a programmable periodic interrupt. The alarm and calendar registers are accessed by a 32-bit data bus.

The time and calendar values are coded in binary-coded decimal (BCD) format. The time format can be 24-hour mode or 12-hour mode with an AM/PM indicator. The number of periods of 1/1024 seconds within 1 second is reported. This information allows a timestamping with an accuracy higher than 1 millisecond.

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

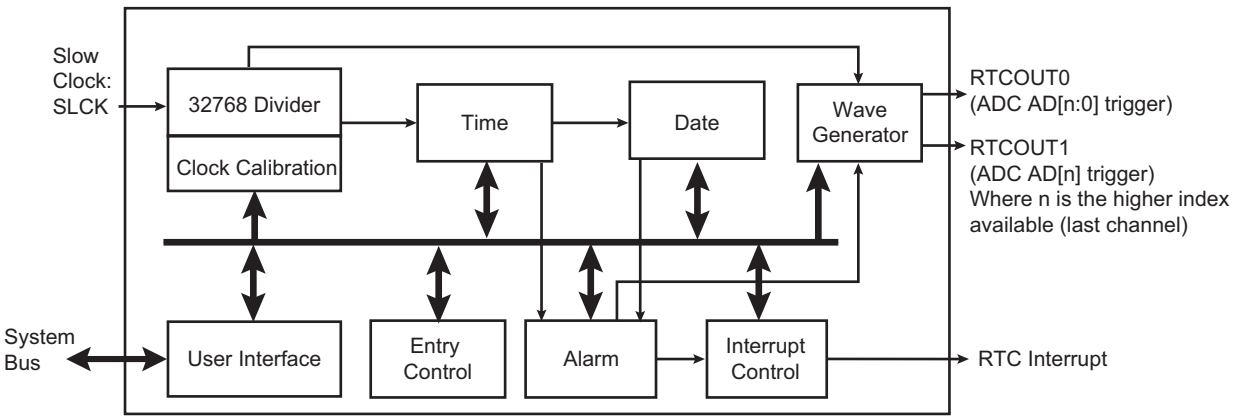
A clock divider calibration circuitry can be used to compensate for crystal oscillator frequency variations.

26.2 Embedded Characteristics

- Full Asynchronous Design for Ultra Low Power Consumption
- Gregorian and Persian Modes Supported
- Milliseconds Image Report
- Programmable Periodic Interrupt
- Safety/security Features:
  - Valid Time and Date Programming Check
  - On-The-Fly Time and Date Validity Check
- Counters Calibration Circuitry to Compensate for Crystal Oscillator Variations
- Waveform Generation for Trigger Event
- Register Write Protection

26.3 Block Diagram

Figure 26-1. Real-time Clock Block Diagram



26.4 Product Dependencies

26.4.1 Power Management

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

26.4.2 Interrupt

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

Table 26-1. Peripheral IDs

Instance	ID
RTC	2

26.5 Functional Description

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

The MS field in the RTC Milliseconds Register (RTC_MSR) reports the number of periods of 1/1024 seconds elapsed within 1 second. The MS field is cleared at the beginning of each 1-second period.

The MS field can be used for timestamping with an accuracy higher than 1 ms. It can also be post-processed by software to provide the millisecond value. The data in milliseconds can be obtained by multiplying the value read in the MS field by 1000/1024 (for an optimal accuracy, the result must be rounded and not truncated).

The valid year range is up to 2099 in Gregorian mode (or 1300 to 1499 in Persian mode).

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

Corrections for leap years are included (all years divisible by 4 being leap years except 1900). This is correct up to the year 2099.

The RTC can generate events to trigger ADC measurements.

26.5.1 Reference Clock

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

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

26.5.2 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.

26.5.3 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.

26.5.4 Error Checking when Programming

Verification on user interface data is performed when accessing the century, year, month, date, day, hours, minutes, seconds and alarms. A check is performed on illegal BCD entries such as illegal date of the month with regard to the year and century configured.

If one of the time fields is not correct, the data is not loaded into the register/counter and a flag is set in the validity register. The user can not reset this flag. It is reset as soon as an acceptable value is programmed. This avoids any further side effects in the hardware. The same procedure is followed for the alarm.

The following checks are performed:

1. Century (check if it is in range 19–20 or 13–14 in Persian mode)
2. Year (BCD entry check)
3. Date (check range 01–31)
4. Month (check if it is in BCD range 01–12, check validity regarding “date”)
5. Day (check range 1–7)
6. Hour (BCD checks: in 24-hour mode, check range 00–23 and check that AM/PM flag is not set if RTC is set in 24-hour mode; in 12-hour mode check range 01–12)
7. Minute (check BCD and range 00–59)
8. Second (check BCD and range 00–59)

Note: If the 12-hour mode is selected by means of the RTC Mode Register (RTC_MR), a 12-hour value can be programmed and the returned value on RTC_TIMR will be the corresponding 24-hour value. The entry control checks the value of the AM/PM indicator (bit 22 of RTC_TIMR) to determine the range to be checked.

26.5.5 RTC Internal Free Running Counter Error Checking

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

An error is reported by TDERR bit in the status register (RTC_SR) if an incorrect value has been detected. The flag can be cleared by setting the TDERRCLR bit in the Status Clear Command Register (RTC_SCCR).

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

The RTC internal free running counters may automatically clear the source of TDERR due to their roll-over (i.e., every 10 seconds for SECONDS[3:0] field in RTC_TIMR). In this case the TDERR is held high until a clear command is asserted by TDERRCLR bit in RTC_SCCR.

26.5.6 Updating Time/Calendar

The update of the time/calendar must be synchronized on a second periodic event by either polling the RTC_SR.SEC status bit or by enabling the SECEN interrupt in the RTC_IER register.

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

The ACKUPD bit must then be read to 1 by either polling the RTC_SR or by enabling the ACKUPD interrupt in the RTC_IER. Once ACKUPD is read to 1, it is mandatory to clear this flag by writing the corresponding bit in the RTC_SCCR, after which the user can write to the Time Register, the Calendar Register, or both.

Once the update is finished, the user must write UPDTIM and/or UPDCAL to 0 in the RTC_CR.

The timing sequence of the time/calendar update is described in Figure 26-2.

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

Figure 26-2. Time/Calendar Update Timing Diagram

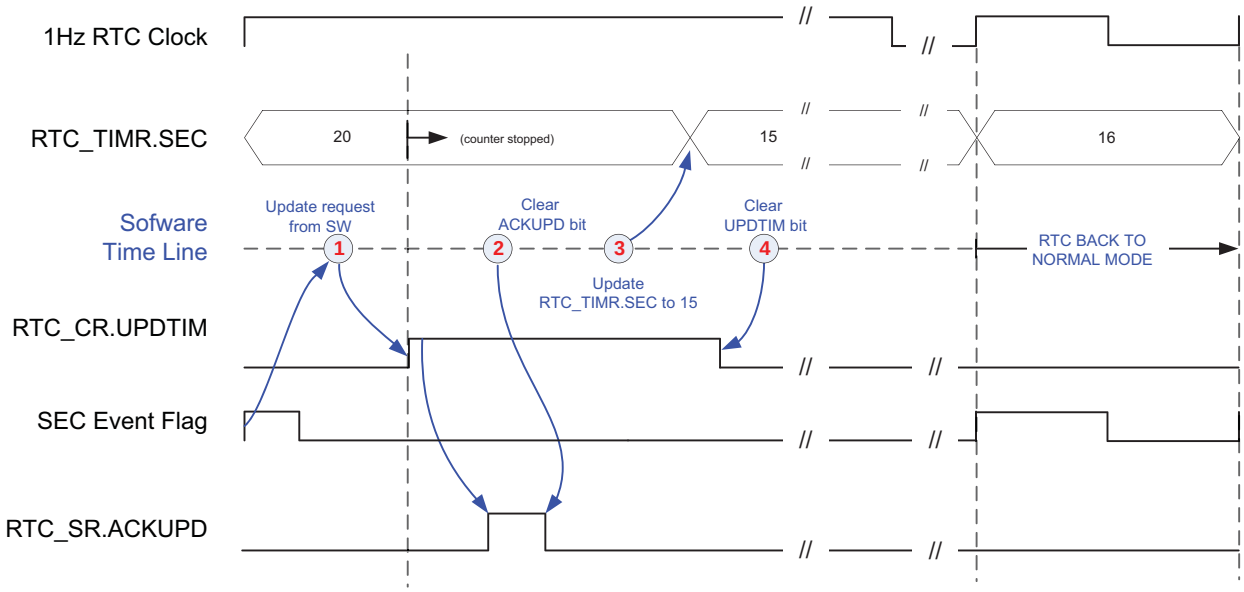
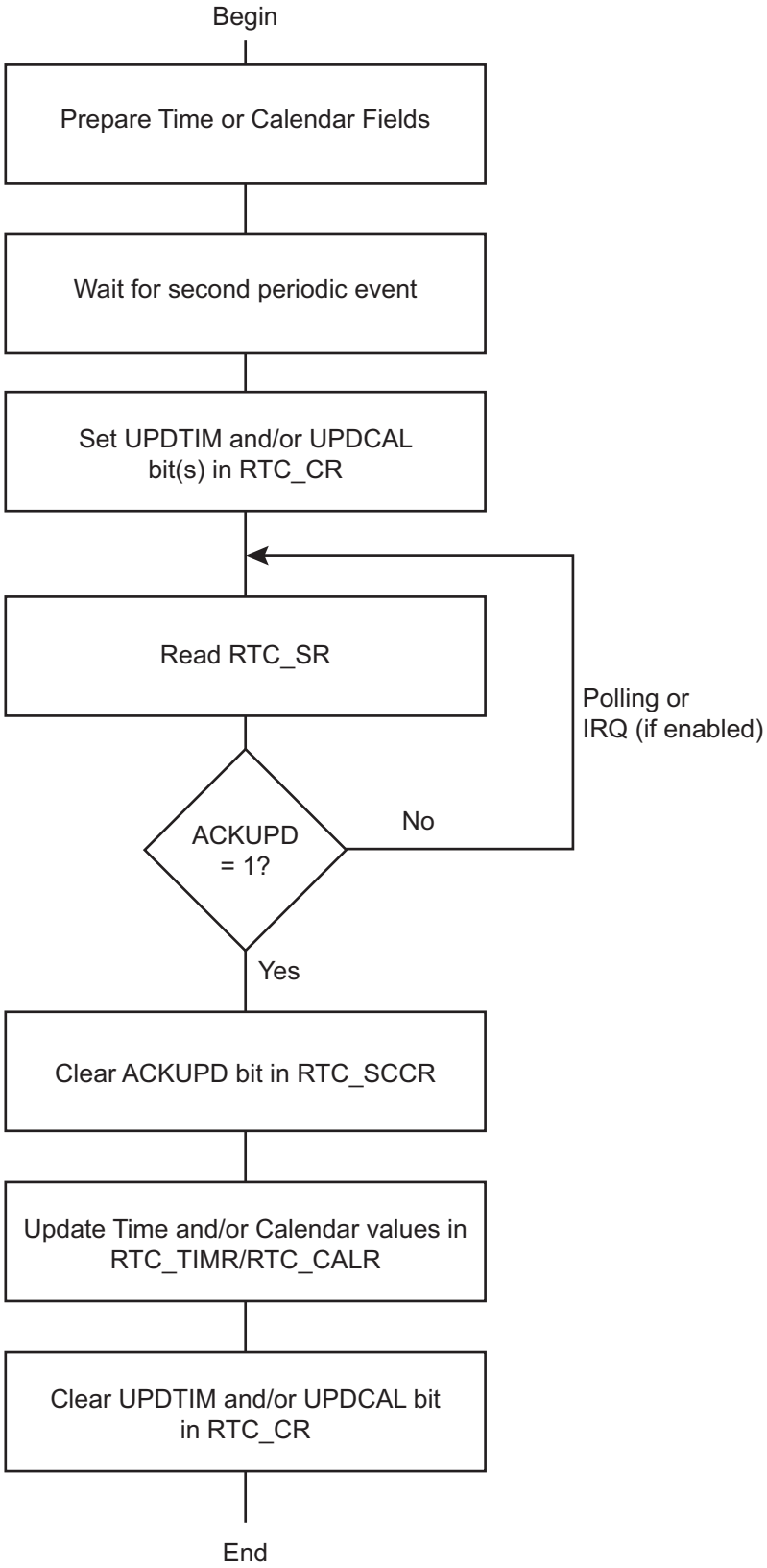


Figure 26-3. Gregorian and Persian Modes Update Sequence



26.5.7 RTC Accurate Clock Calibration

The crystal oscillator that drives the RTC may not be as accurate as expected mainly due to temperature variation. The RTC is equipped with circuitry able to correct slow clock crystal drift.

To compensate for possible temperature variations over time, this accurate clock calibration circuitry can be programmed on-the-fly and also programmed during application manufacturing, in order to correct the crystal frequency accuracy at room temperature (20–25°C). The typical clock drift range at room temperature is ±20 ppm.

In the device operating temperature range, the 32.768 kHz crystal oscillator clock inaccuracy can be up to -200 ppm.

The RTC clock calibration circuitry allows positive or negative correction in a range of 1.5 ppm to 1950 ppm.

The calibration circuitry is fully digital. Thus, the configured correction is independent of temperature, voltage, process, etc., and no additional measurement is required to check that the correction is effective.

If the correction value configured in the calibration circuitry results from an accurate crystal frequency measure, the remaining accuracy is bounded by the values listed below:

- Below 1 ppm, for an initial crystal drift between 1.5 ppm up to 20 ppm, and from 30 ppm to 90 ppm
- Below 2 ppm, for an initial crystal drift between 20 ppm up to 30 ppm, and from 90 ppm to 130 ppm
- Below 5 ppm, for an initial crystal drift between 130 ppm up to 200 ppm

The calibration circuitry does not modify the 32.768 kHz crystal oscillator clock frequency but it acts by slightly modifying the 1 Hz clock period from time to time. The correction event occurs every 1 + [(20 - (19 x HIGHPPM)) x CORRECTION] seconds. When the period is modified, depending on the sign of the correction, the 1 Hz clock period increases or reduces by around 4 ms. Depending on the CORRECTION, NEGPPM and HIGHPPM values configured in RTC_MR, the period interval between two correction events differs.

Figure 26-4. Calibration Circuitry

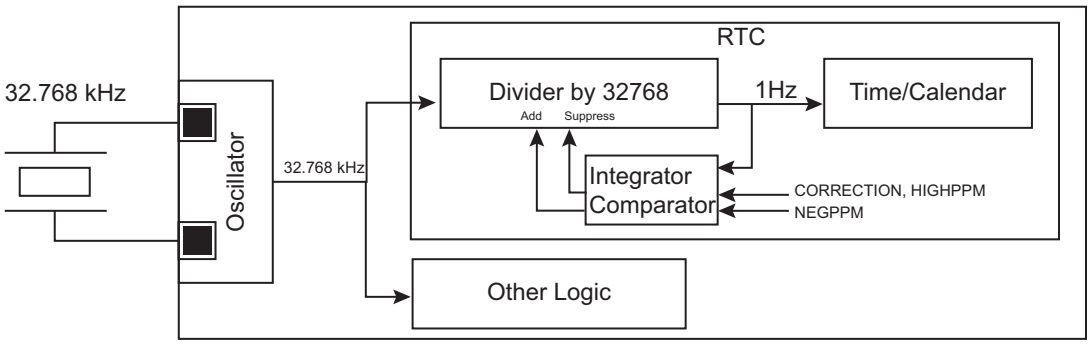
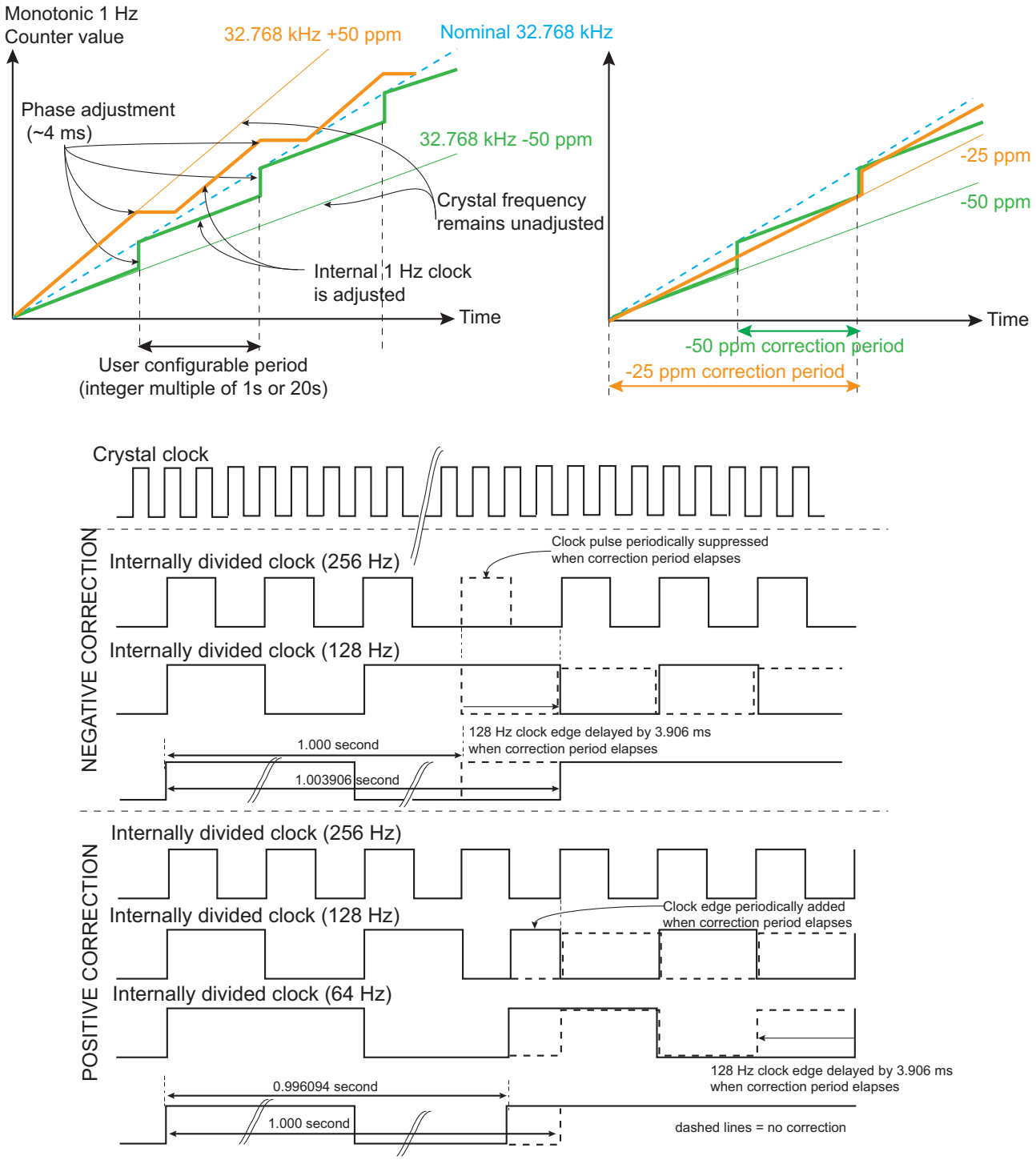


Figure 26-5. Calibration Circuitry Waveforms



The inaccuracy of a crystal oscillator at typical room temperature (±20 ppm at 20–25 °C) can be compensated if a reference clock/signal is used to measure such inaccuracy. This kind of calibration operation can be set up during the final product manufacturing by means of measurement equipment embedding such a reference clock. The correction of value must be programmed into the (RTC_MR), and this value is kept as long as the circuitry is powered (backup area). Removing the backup power supply cancels this calibration. This room temperature calibration can be further processed by means of the networking capability of the target application.

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.

26.5.8 Waveform Generation

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

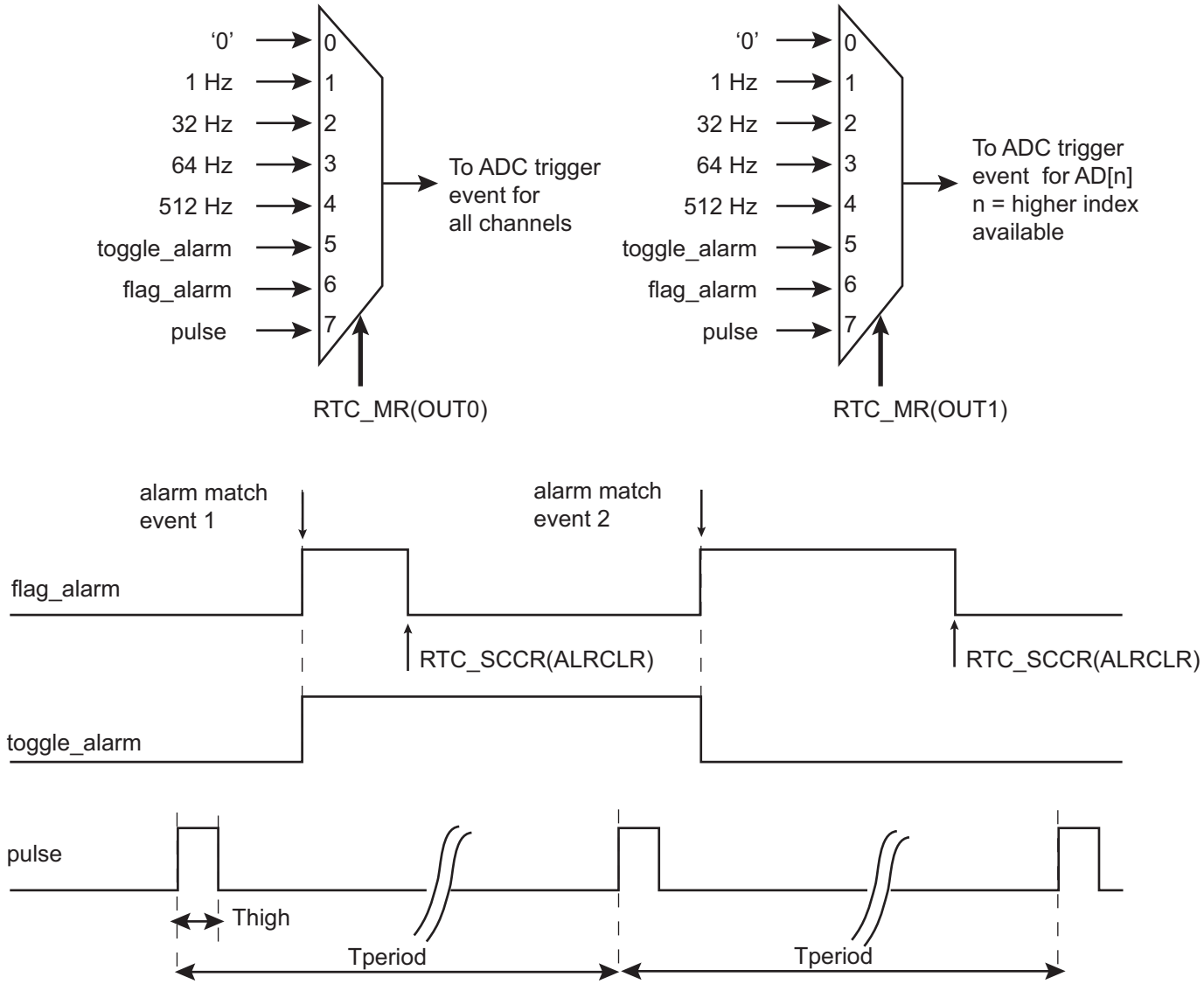
The RTC waveforms are internally routed to ADC trigger events and those events have a source driver selected among five possibilities. Two different triggers can be generated at a time, the first one is configurable through field OUT0 in RTC_MR while the second trigger is configurable through field OUT1 in RTC_MR. OUT0 field manages the trigger for channel AD[n:0] (where n is the higher index available (last channel)), while OUT1 manages the channel AD[n] 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 26-6. Waveform Generation for ADC Trigger Event



26.6 Real-time Clock (RTC) User Interface

Table 26-2. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	RTC_CR	Read/Write	0x00000000
0x04	Mode Register	RTC_MR	Read/Write	0x00000000
0x08	Time Register	RTC_TIMR	Read/Write	0x00000000
0x0C	Calendar Register	RTC_CALR	Read/Write	0x01A11020
0x10	Time Alarm Register	RTC_TIMALR	Read/Write	0x00000000
0x14	Calendar Alarm Register	RTC_CALALR	Read/Write	0x01010000
0x18	Status Register	RTC_SR	Read-only	0x00000000
0x1C	Status Clear Command Register	RTC_SCCR	Write-only	–
0x20	Interrupt Enable Register	RTC_IER	Write-only	–
0x24	Interrupt Disable Register	RTC_IDR	Write-only	–
0x28	Interrupt Mask Register	RTC_IMR	Read-only	0x00000000
0x2C	Valid Entry Register	RTC_VER	Read-only	0x00000000
0x30–0xC8	Reserved	–	–	–
0xCC	Reserved	–	–	–
0xD0	Milliseconds Register	RTC_MSR	Read-only	0x00000000
0xD4–0xE0	Reserved	–	–	–
0xE4	Write Protection Mode Register	RTC_WPMR	Read/Write	0x00000000
0xE8–0xF8	Reserved	–	–	–
0xFC	Reserved	–	–	–

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

26.6.1 RTC Control Register

Name: RTC_CR

Address: 0x400E1460

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the RTC Write Protection Mode Register.

- **UPDTIM: Update Request Time Register**

0: No effect or, if UPDTIM has been previously written to 1, stops the update procedure.

1: Stops the RTC time counting.

Time counting consists of second, minute and hour counters. Time counters can be programmed once this bit is set and acknowledged by the bit ACKUPD of the RTC_SR.

- **UPDCAL: Update Request Calendar Register**

0: No effect or, if UPDCAL has been previously written to 1, stops the update procedure.

1: Stops the RTC calendar counting.

Calendar counting consists of day, date, month, year and century counters. Calendar counters can be programmed once this bit is set and acknowledged by the bit ACKUPD of the RTC_SR.

- **TIMEVSEL: Time Event Selection**

The event that generates the flag TIMEV in RTC_SR depends on the value of TIMEVSEL.

Value	Name	Description
0	MINUTE	Minute change
1	HOUR	Hour change
2	MIDNIGHT	Every day at midnight
3	NOON	Every day at noon

- **CALEVSEL: Calendar Event Selection**

The event that generates the flag CALEV in RTC_SR depends on the value of CALEVSEL

Value	Name	Description
0	WEEK	Week change (every Monday at time 00:00:00)
1	MONTH	Month change (every 01 of each month at time 00:00:00)
2	YEAR	Year change (every January 1 at time 00:00:00)
3	–	Reserved

26.6.2 RTC Mode Register

Name: RTC_MR

Address: 0x400E1464

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	TPERIOD		–	THIGH		
23	22	21	20	19	18	17	16
–	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 RTC Write Protection Mode Register.

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

0: 24-hour mode is selected.

1: 12-hour mode is selected.

- **PERSIAN: PERSIAN Calendar**

0: Gregorian calendar.

1: Persian calendar.

- **NEGPPM: NEGative PPM Correction**

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

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

Refer to CORRECTION and HIGHPPM field descriptions.

Note: NEGPPM must be cleared to correct a crystal slower than 32.768 kHz.

- **CORRECTION: Slow Clock Correction**

0: No correction

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

- **HIGHPPM: HIGH PPM Correction**

0: Lower range ppm correction with accurate correction.

1: Higher range ppm correction with accurate correction.

If the absolute value of the correction to be applied is lower than 30 ppm, it is recommended to clear HIGHPPM. HIGHPPM set to 1 is recommended for 30 ppm correction and above.

Formula:

If HIGHPPM = 0, then the clock frequency correction range is from 1.5 ppm up to 98 ppm. The RTC accuracy is less than 1 ppm for a range correction from 1.5 ppm up to 30 ppm.

The correction field must be programmed according to the required correction in ppm; the formula is as follows:

$$CORRECTION = \frac{3906}{20 \times ppm} - 1$$

The value obtained must be rounded to the nearest integer prior to being programmed into CORRECTION field.

If HIGHPPM = 1, then the clock frequency correction range is from 30.5 ppm up to 1950 ppm. The RTC accuracy is less than 1 ppm for a range correction from 30.5 ppm up to 90 ppm.

The correction field must be programmed according to the required correction in ppm; the formula is as follows:

$$CORRECTION = \frac{3906}{ppm} - 1$$

The value obtained must be rounded to the nearest integer prior to be programmed into CORRECTION field.

If NEGPPM is set to 1, the ppm correction is negative (used to correct crystals that are faster than the nominal 32.768 kHz).

- **OUT0: 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
5	ALARM_TOGGLE	Output toggles when alarm flag rises
6	ALARM_FLAG	Output is a copy of the alarm flag
7	PROG_PULSE	Duty cycle programmable pulse

- **OUT1: 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
5	ALARM_TOGGLE	Output toggles when alarm flag rises
6	ALARM_FLAG	Output is a copy of the alarm flag
7	PROG_PULSE	Duty cycle programmable pulse

- **THIGH: High Duration of the Output Pulse**

Value	Name	Description
0	H_31MS	31.2 ms
1	H_16MS	15.6 ms
2	H_4MS	3.91 ms
3	H_976US	976 µs
4	H_488US	488 µs
5	H_122US	122 µs
6	H_30US	30.5 µs
7	H_15US	15.2 µs

- **TPERIOD: Period of the Output Pulse**

Value	Name	Description
0	P_1S	1 second
1	P_500MS	500 ms
2	P_250MS	250 ms
3	P_125MS	125 ms

26.6.3 RTC Time Register

Name: RTC_TIMR

Address: 0x400E1468

Access: Read/Write

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

23	22	21	20	19	18	17	16
–	AMPM	HOUR					

15	14	13	12	11	10	9	8
–	MIN						

7	6	5	4	3	2	1	0
–	SEC						

- **SEC: Current Second**

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

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

- **MIN: Current Minute**

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

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

- **HOUR: Current Hour**

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

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

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

0: AM.

1: PM.

26.6.4 RTC Calendar Register

Name: RTC_CALR

Address: 0x400E146C

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	DATE					

23	22	21	20	19	18	17	16
DAY			MONTH				

15	14	13	12	11	10	9	8
YEAR							

7	6	5	4	3	2	1	0
–	CENT						

- **CENT: Current Century**

The range that can be set is 19–20 (Gregorian) or 13–14 (Persian) (BCD).

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

- **YEAR: Current Year**

The range that can be set is 00–99 (BCD).

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

- **MONTH: Current Month**

The range that can be set is 01–12 (BCD).

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

- **DAY: Current Day in Current Week**

The range that can be set is 1–7 (BCD).

The coding of the number (which number represents which day) is user-defined as it has no effect on the date counter.

- **DATE: Current Day in Current Month**

The range that can be set is 01–31 (BCD).

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

26.6.5 RTC Time Alarm Register

Name: RTC_TIMALR

Address: 0x400E1470

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the RTC Write Protection Mode Register.

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

- **SEC: Second Alarm**

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

- **SECEN: Second Alarm Enable**

0: The second-matching alarm is disabled.

1: The second-matching alarm is enabled.

- **MIN: Minute Alarm**

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

- **MINEN: Minute Alarm Enable**

0: The minute-matching alarm is disabled.

1: The minute-matching alarm is enabled.

- **HOUR: Hour Alarm**

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

- **AMPM: AM/PM Indicator**

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

- **HOUREN: Hour Alarm Enable**

0: The hour-matching alarm is disabled.

1: The hour-matching alarm is enabled.

26.6.6 RTC Calendar Alarm Register

Name: RTC_CALALR

Address: 0x400E1474

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the RTC Write Protection Mode Register.

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

- **MONTH: Month Alarm**

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

- **MTHEN: Month Alarm Enable**

0: The month-matching alarm is disabled.

1: The month-matching alarm is enabled.

- **DATE: Date Alarm**

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

- **DATEEN: Date Alarm Enable**

0: The date-matching alarm is disabled.

1: The date-matching alarm is enabled.

26.6.7 RTC Status Register

Name: RTC_SR

Address: 0x400E1478

Access: Read-only

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

- **ACKUPD: Acknowledge for Update**

Value	Name	Description
0	FREERUN	Time and calendar registers cannot be updated.
1	UPDATE	Time and calendar registers can be updated.

- **ALARM: Alarm Flag**

Value	Name	Description
0	NO_ALARMEVENT	No alarm matching condition occurred.
1	ALARMEVENT	An alarm matching condition has occurred.

- **SEC: Second Event**

Value	Name	Description
0	NO_SECEVENT	No second event has occurred since the last clear.
1	SECEVENT	At least one second event has occurred since the last clear.

- **TIMEV: Time Event**

Value	Name	Description
0	NO_TIMEVENT	No time event has occurred since the last clear.
1	TIMEVENT	At least one time event has occurred since the last clear.

Note: The time event is selected in the TIMEVSEL field in the Control Register (RTC_CR) and can be any one of the following events: minute change, hour change, noon, midnight (day change).

- **CALEV: Calendar Event**

Value	Name	Description
0	NO_CALEVENT	No calendar event has occurred since the last clear.
1	CALEVENT	At least one calendar event has occurred since the last clear.

Note: The calendar event is selected in the CALEVSEL field in the Control Register (RTC_CR) and can be any one of the following events: week change, month change and year change.

- **TDERR: Time and/or Date Free Running Error**

Value	Name	Description
0	CORRECT	The internal free running counters are carrying valid values since the last read of the Status Register (RTC_SR).
1	ERR_TIMEDATE	The internal free running counters have been corrupted (invalid date or time, non-BCD values) since the last read and/or they are still invalid.

26.6.8 RTC Status Clear Command Register

Name: RTC_SCCR

Address: 0x400E147C

Access: Write-only

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

- **ACKCLR: Acknowledge Clear**

0: No effect.

1: Clears corresponding status flag in the Status Register (RTC_SR).

- **ALRCLR: Alarm Clear**

0: No effect.

1: Clears corresponding status flag in the Status Register (RTC_SR).

- **SECCLR: Second Clear**

0: No effect.

1: Clears corresponding status flag in the Status Register (RTC_SR).

- **TIMCLR: Time Clear**

0: No effect.

1: Clears corresponding status flag in the Status Register (RTC_SR).

- **CALCLR: Calendar Clear**

0: No effect.

1: Clears corresponding status flag in the Status Register (RTC_SR).

- **TDERRCLR: Time and/or Date Free Running Error Clear**

0: No effect.

1: Clears corresponding status flag in the Status Register (RTC_SR).

26.6.9 RTC Interrupt Enable Register

Name: RTC_IER

Address: 0x400E1480

Access: Write-only

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

- **ACKEN: Acknowledge Update Interrupt Enable**

0: No effect.

1: The acknowledge for update interrupt is enabled.

- **ALREN: Alarm Interrupt Enable**

0: No effect.

1: The alarm interrupt is enabled.

- **SECEN: Second Event Interrupt Enable**

0: No effect.

1: The second periodic interrupt is enabled.

- **TIMEN: Time Event Interrupt Enable**

0: No effect.

1: The selected time event interrupt is enabled.

- **CALEN: Calendar Event Interrupt Enable**

0: No effect.

1: The selected calendar event interrupt is enabled.

- **TDERREN: Time and/or Date Error Interrupt Enable**

0: No effect.

1: The time and date error interrupt is enabled.

26.6.10 RTC Interrupt Disable Register

Name: RTC_IDR

Address: 0x400E1484

Access: Write-only

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

- **ACKDIS: Acknowledge Update Interrupt Disable**

0: No effect.

1: The acknowledge for update interrupt is disabled.

- **ALRDIS: Alarm Interrupt Disable**

0: No effect.

1: The alarm interrupt is disabled.

- **SECDIS: Second Event Interrupt Disable**

0: No effect.

1: The second periodic interrupt is disabled.

- **TIMDIS: Time Event Interrupt Disable**

0: No effect.

1: The selected time event interrupt is disabled.

- **CALDIS: Calendar Event Interrupt Disable**

0: No effect.

1: The selected calendar event interrupt is disabled.

- **TDERRDIS: Time and/or Date Error Interrupt Disable**

0: No effect.

1: The time and date error interrupt is disabled.

26.6.11 RTC Interrupt Mask Register

Name: RTC_IMR

Address: 0x400E1488

Access: Read-only

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

- **ACK: Acknowledge Update Interrupt Mask**

0: The acknowledge for update interrupt is disabled.

1: The acknowledge for update interrupt is enabled.

- **ALR: Alarm Interrupt Mask**

0: The alarm interrupt is disabled.

1: The alarm interrupt is enabled.

- **SEC: Second Event Interrupt Mask**

0: The second periodic interrupt is disabled.

1: The second periodic interrupt is enabled.

- **TIM: Time Event Interrupt Mask**

0: The selected time event interrupt is disabled.

1: The selected time event interrupt is enabled.

- **CAL: Calendar Event Interrupt Mask**

0: The selected calendar event interrupt is disabled.

1: The selected calendar event interrupt is enabled.

- **TDERR: Time and/or Date Error Mask**

0: The time and/or date error event is disabled.

1: The time and/or date error event is enabled.

26.6.12 RTC Valid Entry Register

Name: RTC_VER

Address: 0x400E148C

Access: Read-only

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

- **NVTIM: Non-valid Time**

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

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

- **NVCAL: Non-valid Calendar**

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

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

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

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

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

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

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

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

26.6.13 RTC Milliseconds Register

Name: RTC_MSR

Address: 0x400E1530

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
–	–	–	–	–	–	MS	
7	6	5	4	3	2	1	0
MS							

- **MS: Number of 1/1024 seconds elapsed within 1 second**

This field reports a multiple of 1/1024 seconds. The field is synchronized with the fields reported in RTC_TIMR. Thus the MS field is cleared after each elapsed second. This field is incremented monotonically over a period of 1 second and can be used for timestamping with an accuracy better than 1 millisecond.

26.6.14 RTC Write Protection Mode Register

Name: RTC_WPMR

Address: 0x400E1544

Access: Read/Write

31	30	29	28	27	26	25	24
WPKEY							

23	22	21	20	19	18	17	16
WPKEY							

15	14	13	12	11	10	9	8
WPKEY							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WPEN

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x525443 (“RTC” in ASCII).

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

The following registers can be write-protected:

- RTC Mode Register
- RTC Time Alarm Register
- RTC Calendar Alarm Register

- **WPKEY: Write Protection Key**

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

27. Real-time Timer (RTT)

27.1 Description

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

The RTT can also be configured to be driven by the 1Hz RTC signal, thus taking advantage of a calibrated 1Hz clock.

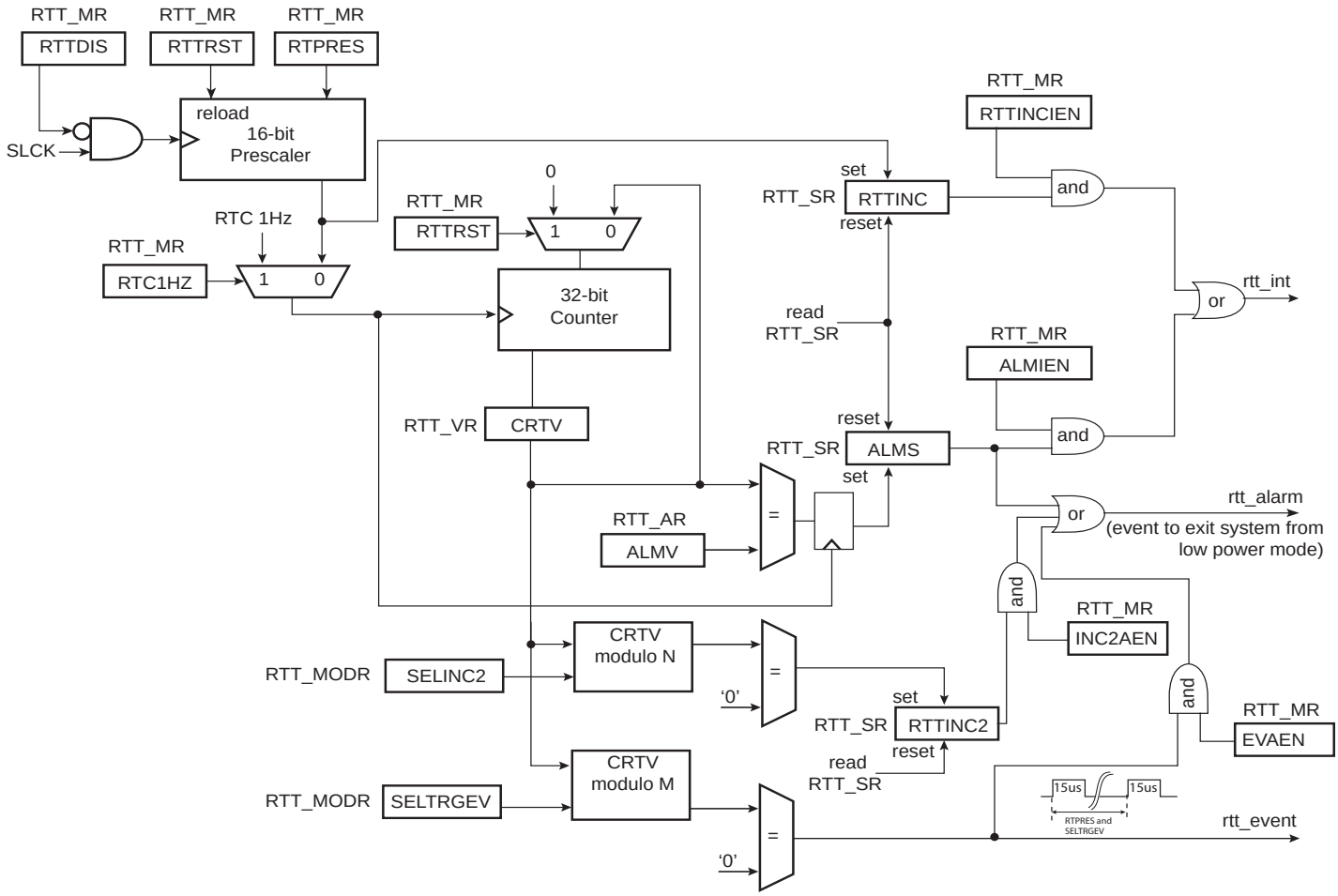
The slow clock source can be fully disabled to reduce power consumption when only an elapsed seconds count is required.

27.2 Embedded Characteristics

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

27.3 Block Diagram

Figure 27-1. Real-time Timer



27.4 Functional Description

The programmable 16-bit prescaler value can be configured through the RTPRES field in the "Real-time Timer Mode Register" (RTT_MR).

Configuring the RTPRES field value to 0x8000 (default value) corresponds to feeding the real-time counter with a 1Hz signal (if the slow clock is 32.768 kHz). The 32-bit counter can count up to 2^{32} seconds, corresponding to more than 136 years, then roll over to 0. Bit RTTINC in the "Real-time Timer Status Register" (RTT_SR) is set each time there is a prescaler roll-over (see Figure 27-2)

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

Setting the RTC1HZ bit in the RTT_MR drives the 32-bit RTT counter from the 1Hz RTC clock. In this mode, the RTPRES field has no effect on the 32-bit counter.

The prescaler roll-over generates an increment of the real-time timer counter if RTC1HZ = 0. Otherwise, if RTC1HZ = 1, the real-time timer counter is incremented every second. The RTTINC bit is set independently from the 32-bit counter increment.

The real-time timer can also be used as a free-running timer with a lower time-base. The best accuracy is achieved by writing RTPRES to 3 in RTT_MR.

Programming RTPRES to 1 or 2 is forbidden.

If the RTT is configured to trigger an interrupt, the interrupt occurs two slow clock cycles after reading the RTT_SR. To prevent several executions of the interrupt handler, the interrupt must be disabled in the interrupt handler and re-enabled when the RTT_SR is cleared.

The CRTV field can be read at any time in the "Real-time Timer Value Register" (RTT_VR). As this value can be updated asynchronously with the Master Clock, the CRTV field must be read twice at the same value to read a correct value.

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

The ALMS flag is always a source of the RTT alarm signal that may be used to exit the system from low power modes (see Figure 27-1).

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

The RTTINC bit can be used to start a periodic interrupt, the period being one second when the RTPRES field value = 0x8000 and the slow clock = 32.768 kHz.

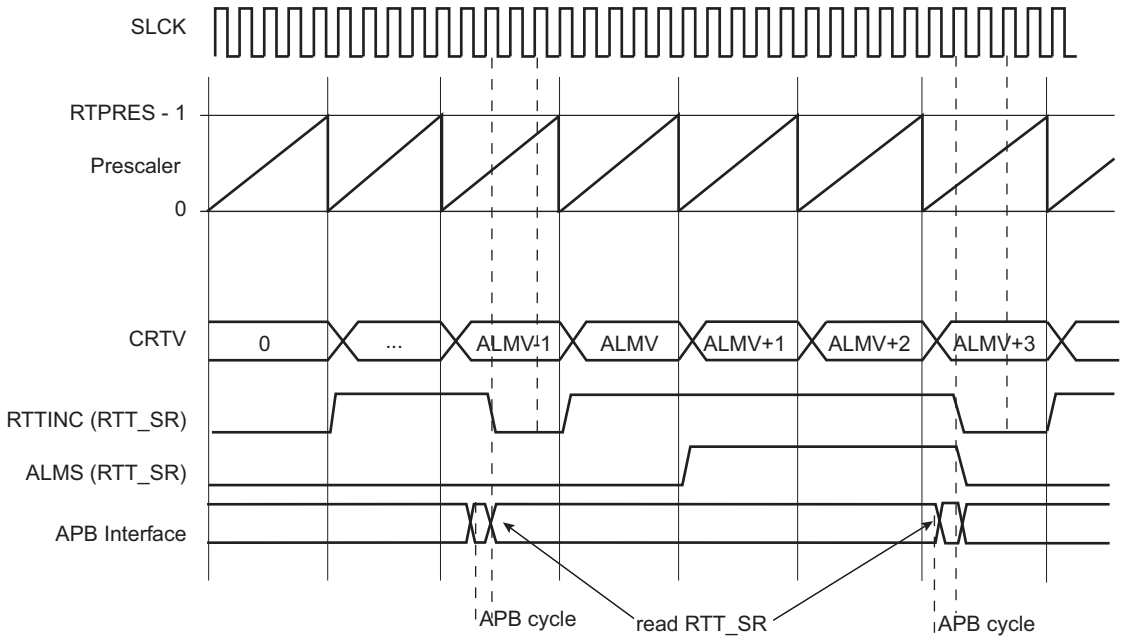
The RTTINCIEN bit must be cleared prior to writing a new RTPRES value in the RTT_MR.

Reading the RTT_SR automatically clears the RTTINC and ALMS bits.

Writing the RTTRST bit in the RTT_MR immediately reloads and restarts the clock divider with the new programmed value. This also resets the 32-bit counter.

When not used, the Real-time Timer can be disabled in order to suppress dynamic power consumption in this module. This can be achieved by setting the RTTDIS bit in the RTT_MR.

Figure 27-2. RTT Counting



The RTTINC2 flag is set when the number of prescaler roll-overs programmed through the SELINC2 field in the "Real-time Timer Modulo Selection Register" (RTT_MODR) has been reached since the last read of the RTT_SR.

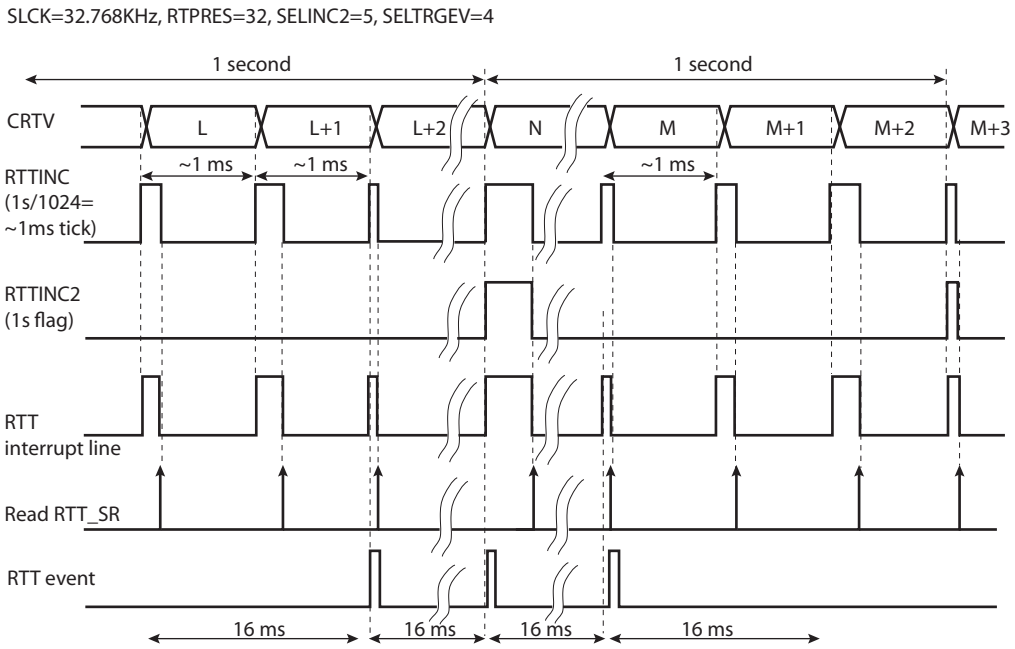
For example, if the SLCK frequency is 32.768 kHz and RTPRES=32, the RTTINC flag rises 1024 times per second (less than 1ms period). If the field SELINC2=5, the RTTINC2 flag rises once per second.

Consequently, if RTTINC is defined as the unique source of interrupt (RTTINCEN=1 and ALMIEN=0 in RTT_MR), the value read in RTT_SR by the interrupt handler determines if the current interrupt event corresponds to a 1-second event (RTT_SR[2:1]=6) or a 1-millisecond event (RTT_SR[2:1]=2). See Figure 27-3.

It is possible to define the RTTINC2 flag as a source of the RTT alarm signal by setting the INC2AEN bit to 1 in RTT_MR (see Figure 27-1).

An event can also be generated and its period of occurrence can be programmed through the SELTRGEV field in RTT_MODR. This event can be enabled as a source of alarm if the bit EVAEN=1.

Figure 27-3. RTTINC2 behavior



27.5 Real-time Timer (RTT) User Interface

Table 27-1. 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
0x10	Modulo Selection Register	RTT_MODR	Read/Write	0x0000_0000

27.5.1 Real-time Timer Mode Register

Name: RTT_MR

Address: 0x400E1430

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	RTC1HZ

23	22	21	20	19	18	17	16
–	EVAEN	INC2AEN	RTTDIS	–	RTTRST	RTTINCIEN	ALMIEN

15	14	13	12	11	10	9	8
RTPRES							

7	6	5	4	3	2	1	0
RTPRES							

- **RTPRES: Real-time Timer Prescaler Value**

Defines the number of SLCK periods required to increment the Real-time timer. RTPRES is defined as follows:

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

RTPRES = 1 or 2: forbidden.

RTPRES ≠ 0,1 or 2: The prescaler period is equal to RTPRES * SLCK periods.

Note: The RTTINCIEN bit must be cleared prior to writing a new RTPRES value.

- **ALMIEN: Alarm Interrupt Enable**

0: The bit ALMS in RTT_SR has no effect on interrupt.

1: The bit ALMS in RTT_SR asserts interrupt.

- **RTTINCIEN: Real-time Timer Increment Interrupt Enable**

0: The bit RTTINC in RTT_SR has no effect on interrupt.

1: The bit RTTINC in RTT_SR asserts interrupt.

- **RTTRST: Real-time Timer Restart**

0: No effect.

1: Reloads and restarts the clock divider with the new programmed value. This also resets the 32-bit counter.

- **RTTDIS: Real-time Timer Disable**

0: The real-time timer is enabled.

1: The real-time timer is disabled (no dynamic power consumption).

- **INC2AEN: RTTINC2 Alarm Enable**

0: The RTTINC2 flag is not a source of the RTT alarm signal.

1: The RTTINC2 flag is a source of the RTT alarm signal.

- **EVAEN: Trigger Event Alarm Enable**

0: The rtt_event signal is not a source of the RTT alarm signal.

1: The rtt_event signal is a source of the RTT alarm signal.

- **RTC1HZ: Real-Time Clock 1Hz Clock Selection**

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

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

27.5.2 Real-time Timer Alarm Register

Name: RTT_AR

Address: 0x400E1434

Access: Read/Write

31	30	29	28	27	26	25	24
ALMV							

23	22	21	20	19	18	17	16
ALMV							

15	14	13	12	11	10	9	8
ALMV							

7	6	5	4	3	2	1	0
ALMV							

- **ALMV: Alarm Value**

When the CRTV value in RTT_VR equals the ALMV field, the ALMS flag is set in RTT_SR. As soon as the ALMS flag rises, the CRTV value equals ALMV+1 (refer to Figure 27-2).

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

27.5.3 Real-time Timer Value Register

Name: RTT_VR

Address: 0x400E1438

Access: Read-only

31	30	29	28	27	26	25	24
CRTV							

23	22	21	20	19	18	17	16
CRTV							

15	14	13	12	11	10	9	8
CRTV							

7	6	5	4	3	2	1	0
CRTV							

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

Returns the current value of the Real-time Timer.

Note: As CRTV can be updated asynchronously, it must be read twice at the same value.

27.5.4 Real-time Timer Status Register

Name: RTT_SR

Address: 0x400E143C

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	–	RTTINC2	RTTINC	ALMS

- **ALMS: Real-time Alarm Status (cleared on read)**

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

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

- **RTTINC: Prescaler Roll-over Status (cleared on read)**

0: No prescaler roll-over occurred since the last read of the RTT_SR.

1: Prescaler roll-over occurred since the last read of the RTT_SR.

- **RTTINC2: Predefined Number of Prescaler Roll-over Status (cleared on read)**

0: SELINC2=0 or the number of prescaler roll-overs programmed through the SELINC2 field in RTT_MODR has not been reached since the last read of the RTT_SR.

1: The number of prescaler roll-overs programmed through the SELINC2 field has been reached since the last read of the RTT_SR.

27.5.5 Real-time Timer Modulo Selection Register

Name: RTT_MODR

Address: 0x400E1440

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
–	–	–	–	–	SELTRGEV		
7	6	5	4	3	2	1	0
–	–	–	–	–	SELINC2		

- **SELINC2: Selection of the 32-bit Counter Modulo to generate RTTINC2 flag**

Value	Name	Description
0	NO_RTTINC2	The RTTINC2 flag never rises
1	MOD64	The RTTINC2 flag is set when CRTV modulo 64 equals 0
2	MOD128	The RTTINC2 flag is set when CRTV modulo 128 equals 0
3	MOD256	The RTTINC2 flag is set when CRTV modulo 256 equals 0
4	MOD512	The RTTINC2 flag is set when CRTV modulo 512 equals 0
5	MOD1024	The RTTINC2 flag is set when CRTV modulo 1024 equals 0. Example: If RTPRES=32 then RTTINC2 flag rises once per second if the slow clock is 32.768 kHz.
6	MOD2048	The RTTINC2 flag is set when CRTV modulo 2048 equals 0
7	MOD4096	The RTTINC2 flag is set when CRTV modulo 4096 equals 0

- **SELTRGEV: Selection of the 32-bit Counter Modulo to generate the trigger event**

Value	Name	Description
0	NO_EVENT	No event generated
1	MOD2	Event occurs when CRTV modulo 2 equals 0
2	MOD4	Event occurs when CRTV modulo 4 equals 0
3	MOD8	Event occurs when CRTV modulo 8 equals 0
4	MOD16	Event occurs when CRTV modulo 16 equals 0
5	MOD32	Event occurs when CRTV modulo 32 equals 0
6	MOD64	Event occurs when CRTV modulo 64 equals 0
7	MOD128	Event occurs when CRTV modulo 128 equals 0

28. General Purpose Backup Registers (GPBR)

28.1 Description

The System Controller embeds 256 bits of General Purpose Backup registers organized as Eight 32-bit registers.

It is possible to generate an immediate clear of the content of General Purpose Backup registers 0 to GPBR_NUMBER_DIV2-1 (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).

28.2 Embedded Characteristics

- 256 bits of General Purpose Backup Registers
- Immediate Clear on Tamper Event

28.3 General Purpose Backup Registers (GPBR) User Interface

Table 28-1. Register Mapping

Offset	Register	Name	Access	Reset
0x0	General Purpose Backup Register 0	SYS_GPBR0	Read/Write	0x00000000
...	...	...	...	...
0xAC	General Purpose Backup Register 7	SYS_GPBR7	Read/Write	0x00000000

28.3.1 General Purpose Backup Register x

Name: SYS_GPBRx

Address: 0x400E1490

Access: Read/Write

31	30	29	28	27	26	25	24
GPBR_VALUE							

23	22	21	20	19	18	17	16
GPBR_VALUE							

15	14	13	12	11	10	9	8
GPBR_VALUE							

7	6	5	4	3	2	1	0
GPBR_VALUE							

These registers are reset at first power-up and on each loss of VDDIO.

- **GPBR_VALUE: Value of GPBR x**

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

29. Flexible Serial Communication Controller (FLEXCOM)

29.1 Description

The Flexible Serial Communication Controller (FLEXCOM) offers several serial communication protocols that are managed by the three submodules USART, SPI and TWI.

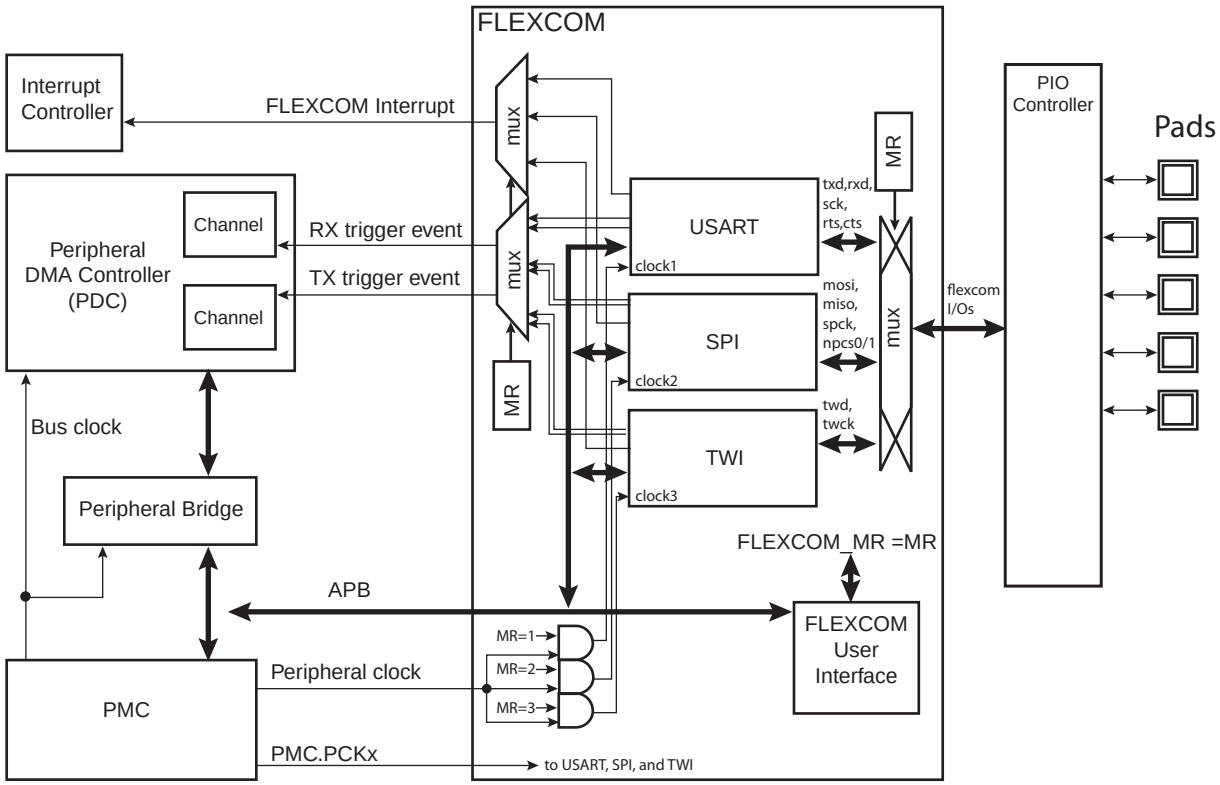
A single submodule at a time can be enabled by the FLEXCOM Mode Register (FLEXCOM_MR). The I/O lines, interrupt lines, and event lines of the three submodules are multiplexed by the FLEXCOM. All details related to these submodules are provided in independent sections.

29.2 Embedded Characteristics

- Selection of USART, SPI or TWI submodules
- Multiplexing of interrupt lines into a single interrupt line
- Multiplexing of the trigger events to PDC

29.3 Block Diagram

Figure 29-1. FLEXCOM Block Diagram



29.4 I/O Lines Description

Table 29-1. I/O Line Description

Name	Description			Type
	USART/UART	SPI	TWI	
TXD_IO1	TXD	MOSI	TWD	I/O
RXD_IO2	RXD	MISO	TWCK	I/O
SCK_IO3	SCK	SPCK	SMBALERT(1)	I/O
CTS_IO4	CTS	NPCS0/NSS	–	I/O
RTS_IO5	RTS	NPCS1	–	O

Note: 1. FLEXCOM3 only

29.5 Product Dependencies

29.5.1 I/O Lines

The pins used for interfacing the FLEXCOM are multiplexed with the PIO lines. The programmer must first program the PIO controller to assign the desired FLEXCOM pins to their peripheral function. If I/O lines of the FLEXCOM are not used by the application, they can be used for other purposes by the PIO Controller.

29.5.2 Power Management

The peripheral clock is not continuously provided to the FLEXCOM. The programmer must first enable the FLEXCOM Clock in the Power Management Controller (PMC) before using the FLEXCOM. However, if the application does not require FLEXCOM operations, the FLEXCOM clock can be stopped and restarted later.

In SleepWalking mode (asynchronous partial wakeup), the PMC must be configured to enable SleepWalking for the FLEXCOM in the Sleepwalking Enable Register (PMC_SLPWK_ER). The peripheral clock can be automatically provided to the FLEXCOM, depending on the instructions (requests) provided by the FLEXCOM to the PMC.

29.5.3 Interrupt Sources

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

29.6 FLEXCOM Functional Description

The selection of USART, SPI or TWI is defined in FLEXCOM_MR.

When a submodule is selected, the others are disabled and cannot be configured.

A unique interrupt line and the PDC channels are shared among the three submodules.

Table 29-2. FLEXCOM Configuration

Function									
Main Function	**Subfunction**	**FLEX0**	**FLEX1**	**FLEX2**	**FLEX3**	**FLEX4**	**FLEX5**	**FLEX6**	**FLEX7**
TWI		✓	✓	✓	✓	✓	✓	✓	✓
TWI	High-speed SMBUS	–	–	–	✓	–	–	–	–
	High-speed Alternate Function	✓	✓	✓	✓	✓	✓	✓	✓
	High-speed Sleepwalking	✓	✓	✓	✓	✓	✓	✓	✓
UART		✓	✓	✓	✓	✓	✓	✓	✓
USART		✓	✓	✓	✓	✓	✓	✓	✓
USART	Sleepwalking	✓	✓	✓	✓	✓	✓	✓	✓
	ISO7816	–	–	–	–	–	✓	✓	✓
	IrDA	–	–	–	–	–	✓	✓	✓
	LIN Mode	–	–	–	–	✓	–	–	–
	SPI Mode	–	–	✓	–	–	–	–	–
SPI	2 Chip Selects	✓	✓	✓	✓	✓	✓	✓	✓

29.7 Flexible Serial Communication Controller (FLEXCOM) User Interface

Table 29-3. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	FLEXCOM Mode Register	FLEXCOM_MR	Read/Write	0x0
0x0004–0x000C	Reserved	–	–	–
0x0010	FLEXCOM Receive Holding Register	FLEXCOM_RHR	Read-only	0x0
0x0014–0x001C	Reserved	–	–	–
0x0020	FLEXCOM Transmit Holding Register	FLEXCOM_THR	Read/Write	0x0
0x0024–0x01FC	Reserved	–	–	–
0x0200–0x02FC	Reserved for USART	–	–	–
0x0300–0x0328	Reserved for USART PDC Registers	–	–	–
0x032C–0x03FC	Reserved	–	–	–
0x0400–0x04FC	Reserved for Fast SPI	–	–	–
0x0500–0x0528	Reserved for Fast SPI PDC Registers	–	–	–
0x052C–0x05FC	Reserved	–	–	–
0x0600–0x06FC	Reserved for TWI	–	–	–
0x0700–0x0728	Reserved for TWI PDC Registers	–	–	–
0x072C–0x07FC	Reserved	–	–	–

29.7.1 FLEXCOM Mode Register

Name: FLEXCOM_MR

Address: 0x4000C000 (0), 0x40020000 (1), 0x40024000 (2), 0x40018000 (3), 0x4001C000 (4), 0x40008000 (5), 0x40040000 (6), 0x40034000 (7)

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
–	–	–	–	–	–	OPMODE	

- **OPMODE: FLEXCOM Operating Mode**

Value	Name	Description
0	NO_COM	No communication
1	USART	All related USART related protocols are selected (RS232, IrDA, ISO7816, LIN) All SPI/TWI related registers are not accessible and have no impact on IOs.
2	SPI	SPI operating mode is selected. All USART/TWI related registers are not accessible and have no impact on IOs.
3	TWI	All related TWI protocols are selected (TWI, SMBUS). All USART/SPI related registers are not accessible and have no impact on IOs.

29.7.2 FLEXCOM Transmit Holding Register

Name: FLEXCOM_THR

Address: 0x4000C020 (0), 0x40020020 (1), 0x40024020 (2), 0x40018020 (3), 0x4001C020 (4), 0x40008020 (5), 0x40040020 (6), 0x40034020 (7)

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
TXDATA							

7	6	5	4	3	2	1	0
TXDATA							

- **TXDATA: Transmit Data**

This register is an image of:

- USART Transmit Holding Register (US_THR) if FLEXCOM_MR.OPMODE = 1
- SPI Transmit Data Register (SPI_TDR) if FLEXCOM_MR.OPMODE = 2
- TWI Transmit Holding Register (TWI_THR) if FLEXCOM_MR.OPMODE = 3

29.7.3 FLEXCOM Receive Holding Register

Name: FLEXCOM_RHR

Address: 0x4000C010 (0), 0x40020010 (1), 0x40024010 (2), 0x40018010 (3), 0x4001C010 (4), 0x40008010 (5), 0x40040010 (6), 0x40034010 (7)

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
RXDATA							

7	6	5	4	3	2	1	0
RXDATA							

- **RXDATA: Receive Data**

This register is an image of:

- USART Receive Holding Register (US_RHR) if FLEXCOM_MR.OPMODE = 1
- SPI Receive Data Register (SPI_RDR) if FLEXCOM_MR.OPMODE = 2
- TWI Transmit Holding Register (TWI_RHR) if FLEXCOM_MR.OPMODE = 3

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 timeout 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 LIN, and SPI buses, with ISO7816 T = 0 or T = 1 smart card slots, and infrared transceivers. The hardware handshaking feature enables an out-of-band flow control by automatic management of 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
- Baud Rate can be Independent of the Processor/Peripheral Clock
- Asynchronous Partial wakeup on Receive Line Activity (SleepWalking)
- Comparison Function on Received Character
- 5- to 9-bit Full-duplex Synchronous or Asynchronous Serial Communications
  - 1, 1.5 or 2 Stop Bits in Asynchronous Mode or 1 or 2 Stop Bits in Synchronous Mode
  - Parity Generation and Error Detection
  - Framing Error Detection, Overrun Error Detection
  - Digital Filter on Receive Line
  - MSB- or LSB-first
  - Optional Break Generation and Detection
  - By 8 or by 16 Over-sampling Receiver Frequency
  - Optional Hardware Handshaking RTS-CTS
  - Receiver Timeout and Transmitter Timeguard
  - Optional Multidrop Mode with Address Generation and Detection
- 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 $f_{peripheral\ clock}/6$
- LIN Mode
  - Compliant with LIN 1.3 and LIN 2.0 Specifications
  - Master or Slave
  - Processing of Frames with Up to 256 Data Bytes
  - Response Data Length can be Configurable or Defined Automatically by the Identifier
  - Self-synchronization in Slave Node Configuration
  - Automatic Processing and Verification of the “Synch Break” and the “Synch Field”
  - “Synch Break” Detection Even When Partially Superimposed with a Data Byte
  - Automatic Identifier Parity Calculation/Sending and Verification
  - Parity Sending and Verification Can be Disabled
  - Automatic Checksum Calculation/sending and Verification
  - Checksum Sending and Verification Can be Disabled
  - Support Both “Classic” and “Enhanced” Checksum Types
  - Full LIN Error Checking and Reporting
  - Frame Slot Mode: Master Allocates Slots to the Scheduled Frames Automatically
  - Generation of the Wakeup Signal
- Test Modes
  - Remote Loopback, Local Loopback, Automatic Echo
- Supports Connection of two Peripheral DMA Controller Channels (PDC)
- Offers Buffer Transfer without Processor Intervention

30.3 Block Diagram

Figure 30-1. USART Block Diagram

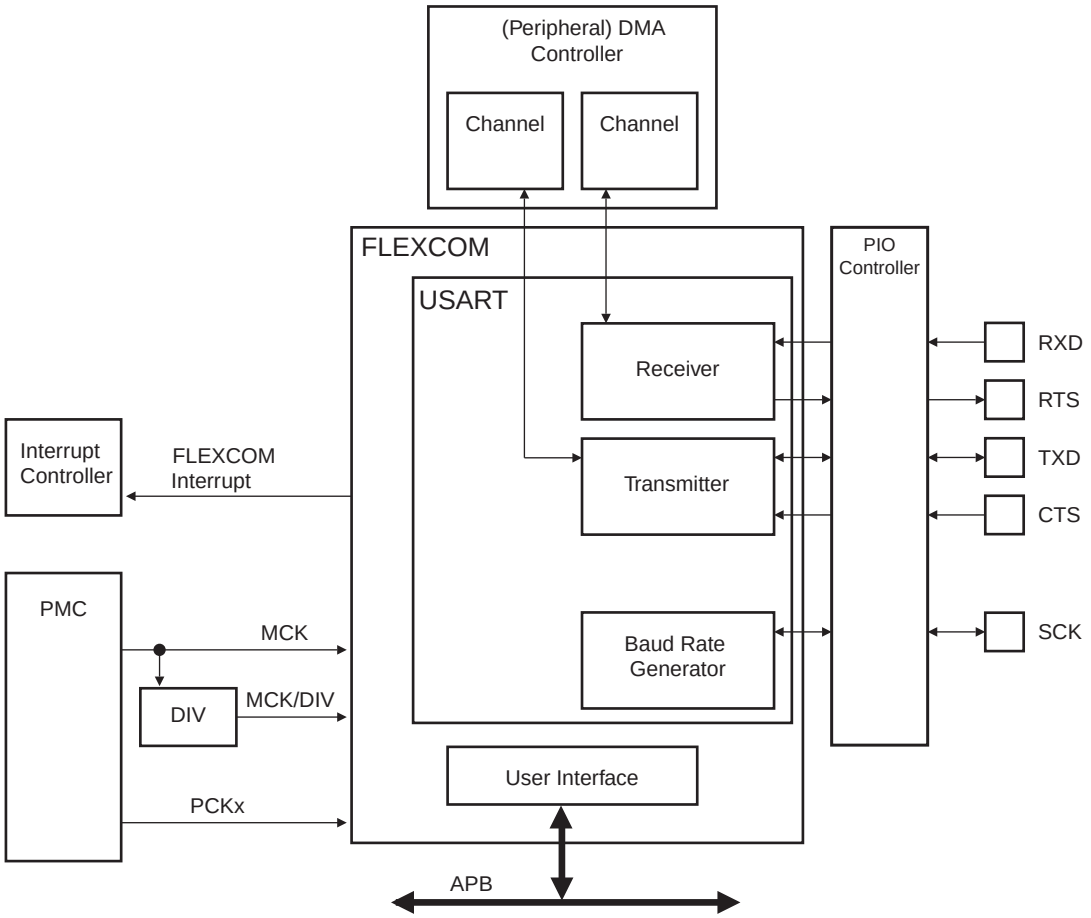


Table 30-1. SPI Operating Mode

Pin	USART	Basic SPI Slave	Basic SPI Master
RXD	RXD	MOSI	MISO
TXD	TXD	MISO	MOSI
RTS	RTS	–	CS
CTS	CTS	CS	–

30.4 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

30.5 Product Dependencies

30.5.1 I/O Lines

The pins used for interfacing the USART are multiplexed with the SPI and TWI lines within the FLEXCOM module. The programmer must first program the PIO controller to assign the desired FLEXCOM pins to their peripheral function. If I/O lines of the FLEXCOM are not used by the application, they can be used for other purposes by the PIO Controller.

Table 30-3. I/O Lines

Instance	Signal	I/O Line	Peripheral
USART0	RXD0/SPI0_MISO/TWCK0	PA9	A
USART0	SCK0/SPI0_SPCK	PB0	A
USART0	SPI0_NPCS0/CTS0	PA25	A
USART0	SPI0_NPCS1/RTS0	PA26	A
USART0	TXD0/SPI0_MOSI/TWD0	PA10	A
USART1	RXD1/SPI1_MISO/TWCK1	PB2	A
USART1	SCK1/SPI1_SPCK	PA27	A
USART1	SPI1_NPCS0/CTS1	PA28	A
USART1	SPI1_NPCS1/RTS1	PA29	A
USART1	TXD1/SPI1_MOSI/TWD1	PB3	A
USART2	RXD2/SPI2_MISO/TWCK2	PA5	A
USART2	SCK2/SPI2_SPCK	PA15	B
USART2	SCK2/SPI2_SPCK	PA24	B
USART2	SPI2_NPCS0/CTS2	PA16	A
USART2	SPI2_NPCS1/RTS2	PA15	A
USART2	TXD2/SPI2_MOSI/TWD2	PA6	A

Table 30-3. I/O Lines (Continued)

Instance	Signal	I/O Line	Peripheral
USART3	RXD3/SPI3_MISO/TWCK3	PA4	A
USART3	SCK3/SPI3_SPCK	PB13	A
USART3	SPI3_NPCS0/CTS3	PB14	A
USART3	SPI3_NPCS1/RTS3	PB15	A
USART3	TXD3/SPI3_MOSI/TWD3	PA3	A
USART4	RXD4/SPI4_MISO/TWCK4	PB9	A
USART4	RXD4/SPI4_MISO/TWCK4	PB11	A
USART4	SCK4/SPI4_SPCK	PB1	A
USART4	SPI4_NPCS0/CTS4	PB8	B
USART4	SPI4_NPCS1/RTS4	PB9	B
USART4	TXD4/SPI4_MOSI/TWD4	PB8	A
USART4	TXD4/SPI4_MOSI/TWD4	PB10	A
USART5	RXD5/SPI5_MISO/TWCK5	PA12	A
USART5	SCK5/SPI5_SPCK	PA14	A
USART5	SPI5_NPCS0/CTS5	PA11	A
USART5	SPI5_NPCS1/RTS5	PA5	B
USART5	SPI5_NPCS1/RTS5	PB2	B
USART5	TXD5/SPI5_MOSI/TWD5	PA13	A
USART6	RXD6/SPI6_MISO/TWCK6	PB1	B
USART6	RXD6/SPI6_MISO/TWCK6	PB11	B
USART6	SCK6/SPI6_SPCK	PB13	B
USART6	SPI6_NPCS0/CTS6	PB14	B
USART6	SPI6_NPCS1/RTS6	PB15	B
USART6	TXD6/SPI6_MOSI/TWD6	PB0	B
USART6	TXD6/SPI6_MOSI/TWD6	PB10	B
USART7	RXD7/SPI7_MISO/TWCK7	PA27	B
USART7	SCK7/SPI7_SPCK	PA29	B
USART7	SPI7_NPCS0/CTS7	PA30	B
USART7	SPI7_NPCS1/RTS7	PA31	B
USART7	TXD7/SPI7_MOSI/TWD7	PA28	B

30.5.2 Power Management

The peripheral clock is not continuously provided to the USART. The programmer must first enable the FLEXCOM Clock in the Power Management Controller (PMC) and set the OPMODE field to value ‘1’ in the FLEXCOM Mode Register (FLEX_MR) before using the USART.

If the OPMODE field differs from ‘1’, the USART clock is stopped.

In SleepWalking mode (asynchronous partial wakeup), the PMC must be configured to enable SleepWalking for the FLEXCOM in the Sleepwalking Enable Register (PMC_SLPWK_ER). The peripheral clock can be automatically provided to the FLEXCOM, depending on the instructions (requests) provided by the FLEXCOM to the PMC.

30.5.3 Interrupt Sources

The USART interrupt line is the FLEXCOM interrupt line if the field OPMODE = 1 in FLEX_MR. The FLEXCOM interrupt line is connected on one of the internal sources of the Interrupt Controller. Using the FLEXCOM interrupt requires the Interrupt Controller to be programmed first.

Table 30-4. Peripheral IDs

Instance	ID
USART0	8
USART1	9
USART2	14
USART3	19
USART4	20
USART5	21
USART6	22
USART7	7

30.6 USART Functional Description

30.6.1 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 is selected by setting the USCLKS field in the USART Mode register (US_MR) to use one of the following:

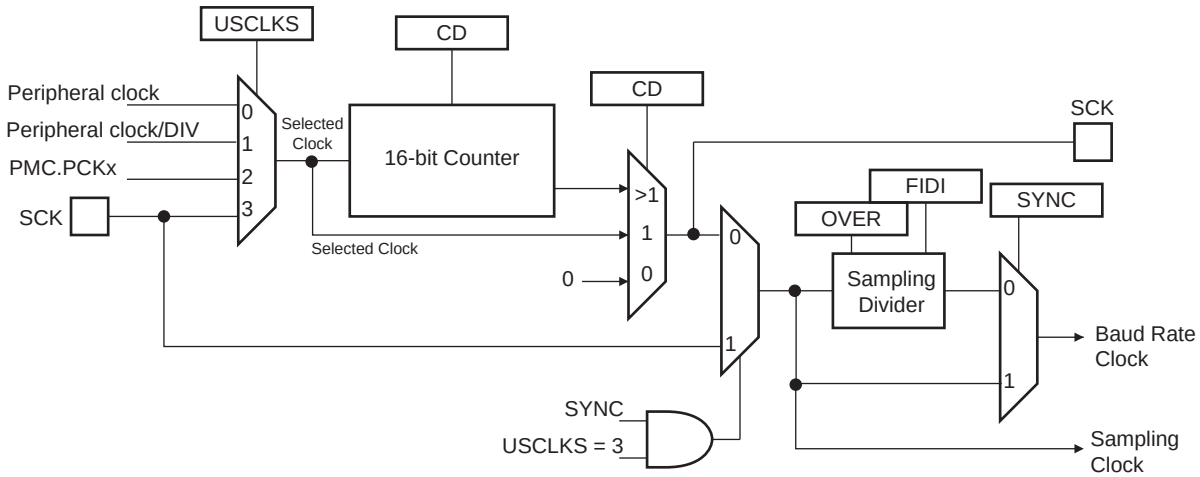
- The peripheral clock
- A division of the peripheral clock, where the divider is product-dependent, but generally set to 8
- A fully programmable processor/peripheral independent clock source provided by PMC (PCK)
- The external clock, available on the SCK pin

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 peripheral clock period. The frequency of the signal provided on SCK must be at least three times lower than the frequency provided on the peripheral clock in USART modes (field USART_MODE equals 0x0 to 0xB), or six times lower in SPI modes (field USART_MODE equals 0xE or 0xF).

If PMC PCK is selected, the baud rate is independent of the processor/peripheral clock and thus processor/peripheral clock frequency can be changed without affecting the USART transfer. The PMC PCKx frequency must always be three times lower than peripheral clock frequency. If PMC PCK is selected (USCLKS=2) and the SCK pin is driven (CLKO=1), the CD field must be greater than 1.

Figure 30-2. Baud Rate Generator



30.6.1.1 Baud Rate in Asynchronous Mode

If the USART is programmed to operate in Asynchronous mode, the selected clock is first divided by CD, which is field programmed in the US_BRGR. The resulting clock is provided to the receiver as a sampling clock and then divided by 16 or 8, depending on how the OVER bit

in the US_MR is programmed.

If OVER is set, the receiver sampling is 8 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 peripheral clock divided by 8, assuming that peripheral clock 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%.

$$Error = 1 - \left(\frac{ExpectedBaudRate}{ActualBaudRate}\right)$$

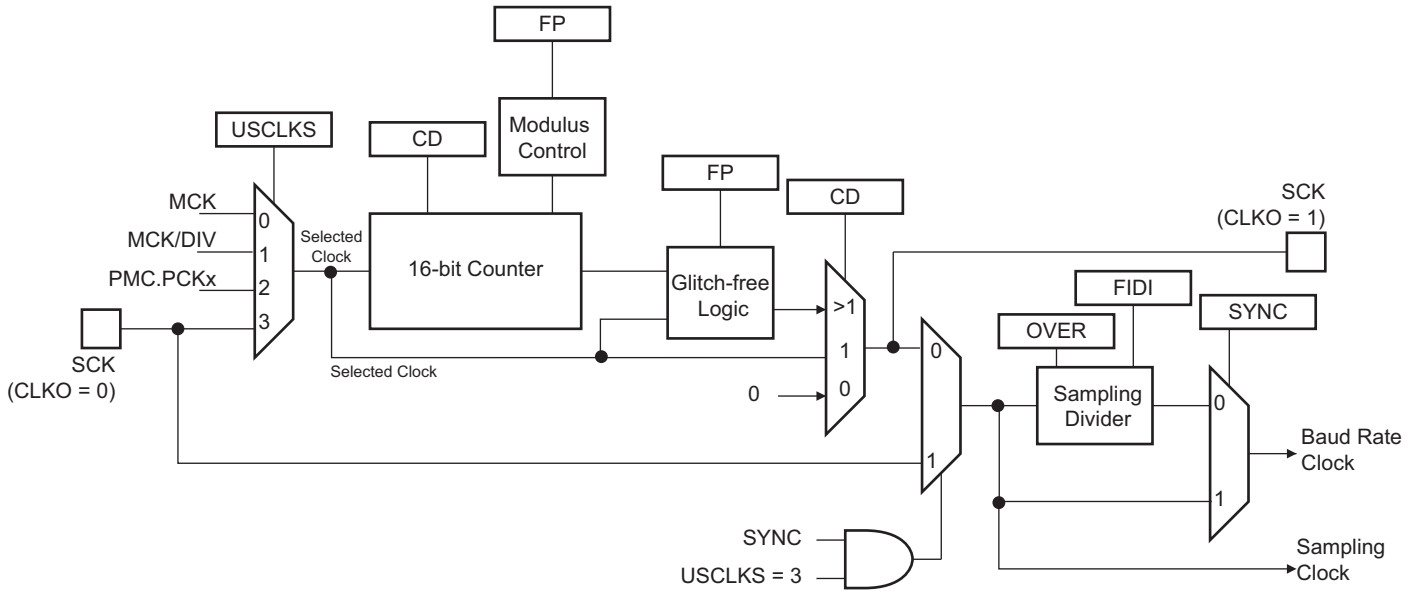
30.6.1.2 Fractional Baud Rate in Asynchronous Mode

The baud rate generator 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. The fractional baud rate is calculated using the following formula:

$$Baudrate = \frac{SelectedClock}{\left(8(2-Over)\left(CD + \frac{FP}{8}\right)\right)}$$

The modified architecture is presented in the following Figure 30-3.

Figure 30-3. Fractional Baud Rate Generator



Warning: When the value of field FP is greater than 0, the SCK (oversampling clock) gener-ates nonconstant duty cycles. The SCK high duration is increased by “selected clock” period from time to time. The duty cycle depends on the value of the CD field.

30.6.1.3 Baud Rate in Synchronous Mode or SPI Mode

If the USART is programmed to operate in Synchronous mode, the selected clock is simply divided by the field CD in the US_BRGR.

$$BaudRate = \frac{SelectedClock}{CD}$$

In Synchronous mode, if the external clock is selected (USCLKS = 3) and CLKO=0 (Slave mode), 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, CLKO set to ‘1’), the receive part limits the SCK maximum frequency to $f_{peripheral\ clock}/3$ in USART mode, or $f_{peripheral\ clock}/6$ in SPI mode.

When either the external clock SCK or the internal clock divided (peripheral clock/DIV or PMC PCK) 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 peripheral clock is selected, the baud rate generator ensures a 50:50 duty cycle on the SCK pin, even if the value programmed in CD is odd.

30.6.1.4 Baud Rate in ISO 7816 Mode

The ISO7816 specification defines the bit rate with the following formula:

$$B = \frac{Di}{Fi} \times f$$

where:

- B is the bit rate
- Di is the bit-rate adjustment factor
- Fi is the clock frequency division factor
- f is the ISO7816 clock frequency (Hz)

Di is a binary value encoded on a 4-bit field, named DI, as represented in Table 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	744	1116	1488	1806	512	768	1024	1536	2048
1	372	558	744	1116	1488	1860	512	768	1024	1536	2048
2	186	279	372	558	744	930	256	384	512	768	1024
4	93	139.5	186	279	372	465	128	192	256	384	512
8	46.5	69.75	93	139.5	186	232.5	64	96	128	192	256
16	23.25	34.87	46.5	69.75	93	116.2	32	48	64	96	128
32	11.62	17.43	23.25	34.87	46.5	58.13	16	24	32	48	64
12	31	46.5	62	93	124	155	42.66	64	85.33	128	170.6
20	18.6	27.9	37.2	55.8	74.4	93	25.6	38.4	51.2	76.8	102.4

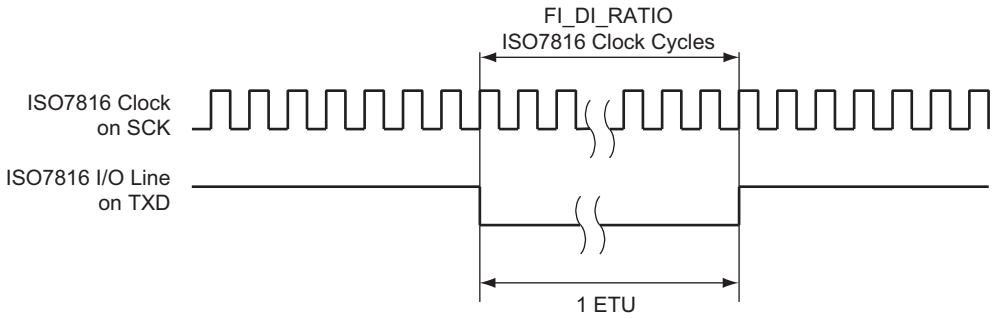
If the USART is configured in ISO7816 mode, the clock selected by the USCLKS field in the US_MR is first divided by the value programmed in US_BRGR.CD. The resulting clock can be provided to the SCK pin to feed the smart card clock inputs. This means that US_MR.CLKO bit can be written to ‘1’.

This clock is then divided by the value programmed in the FI_DI_RATIO field in the FI_DI_Ratio register (US_FIDI). This is performed by the Sampling Divider, which performs a division by up to 2047 in ISO7816 mode. The noninteger values of the Fi/Di Ratio are not supported and the user must program 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 30-4 shows the relation between the Elementary Time Unit, corresponding to a bit time, and the ISO 7816 clock.

Figure 30-4. Elementary Time Unit (ETU)



30.6.2 Receiver and Transmitter Control

After reset, the receiver is disabled. The user must enable the receiver by setting the RXEN bit in the Control register (US_CR). However, the receiver registers can be programmed before the receiver clock is enabled.

After reset, the transmitter is disabled. The user must enable it by setting the TXEN bit in the US_CR. However, the transmitter registers can be programmed before being enabled.

The receiver and the transmitter can be enabled together or independently.

At any time, the software can perform a reset on the receiver or the transmitter of the USART by setting the corresponding bit, RSTRX and RSTTX respectively, in the US_CR. The software resets clear the status flag and reset internal state machines but the user interface configuration registers hold the value configured prior to software reset. Regardless of what the receiver or the transmitter is performing, the communication is immediately stopped.

The user can also independently disable the receiver or the transmitter by setting RXDIS and TXDIS respectively in the US_CR. If the receiver is disabled during a character reception, the USART waits until the end of reception of the current character, then the reception is stopped. If the transmitter is disabled while it is operating, the USART waits the end of transmission of both the current character and character being stored in the USART Transmit Holding register (US_THR). If a timeguard is programmed, it is handled normally.

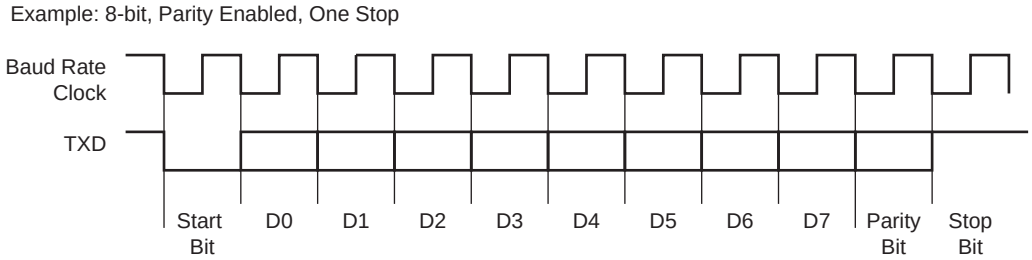
30.6.3 Synchronous and Asynchronous Modes

30.6.3.1 Transmitter Operations

The transmitter performs the same in both Synchronous and Asynchronous operating modes (SYNC = 0 or SYNC = 1). One start bit, up to 9 data bits, one optional parity bit and up to two stop bits are successively shifted out on the TXD pin at each falling edge of the programmed serial clock.

The number of data bits is selected by the CHRL field and the MODE 9 bit in US_MR. Nine bits are selected by setting the MODE 9 bit regardless of the CHRL field. The parity bit is set depending on 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.

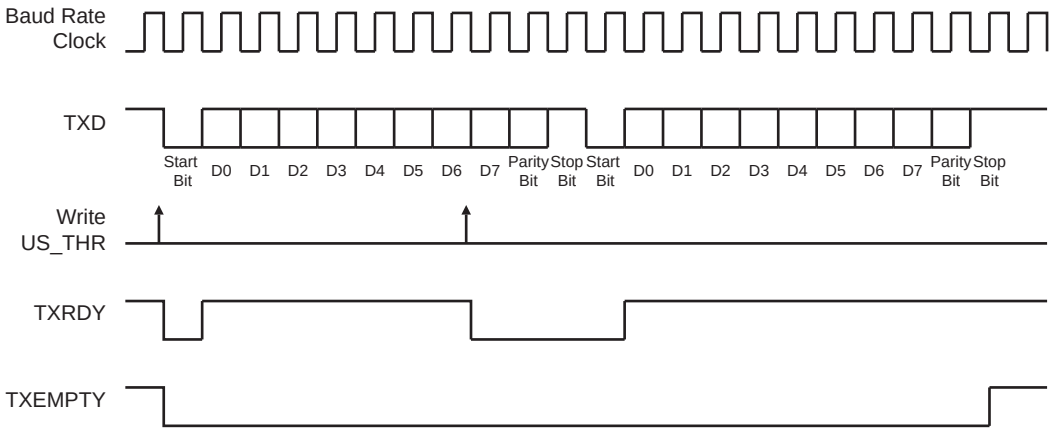
Figure 30-5. Character Transmit



The characters are sent by writing in the US_THR. The transmitter reports two status bits in the Channel Status register (US_CSR): TXRDY (Transmitter Ready), which indicates that US_THR is empty and TXEMPTY, which indicates that all the characters written in US_THR have been processed. When the current character processing is completed, the last character written in US_THR is transferred into the Shift register of the transmitter and US_THR becomes empty, thus TXRDY rises.

Both TXRDY and TXEMPTY 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 30-6. Transmitter Status



30.6.3.2 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 = 0), a start is detected at the eighth sample to 0. Data bits, parity bit and stop bit are assumed to have a duration corresponding to 16 oversampling clock cycles. If the oversampling is 8 (OVER = 1), a start bit is detected at the fourth sample to 0. Data bits, parity bit and stop bit are assumed to have a duration corresponding to 8 oversampling clock cycles.

The number of data bits, first bit sent and Parity mode are selected by the same fields and bits as the transmitter, i.e., respectively CHRL, MODE9, MSBF and PAR. For the synchronization mechanism **only**, the number of stop bits has no effect on the receiver as it considers only one stop bit, regardless of the field NBSTOP, so that resynchronization between the receiver and the transmitter can occur. Moreover, as soon as the stop bit is sampled, the receiver starts looking for a new start bit so that resynchronization can also be accomplished when the transmitter is operating with one stop bit.

Figure 30-7 and Figure 30-8 illustrate start detection and character reception when USART operates in Asynchronous mode.

Figure 30-7. Asynchronous Start Detection

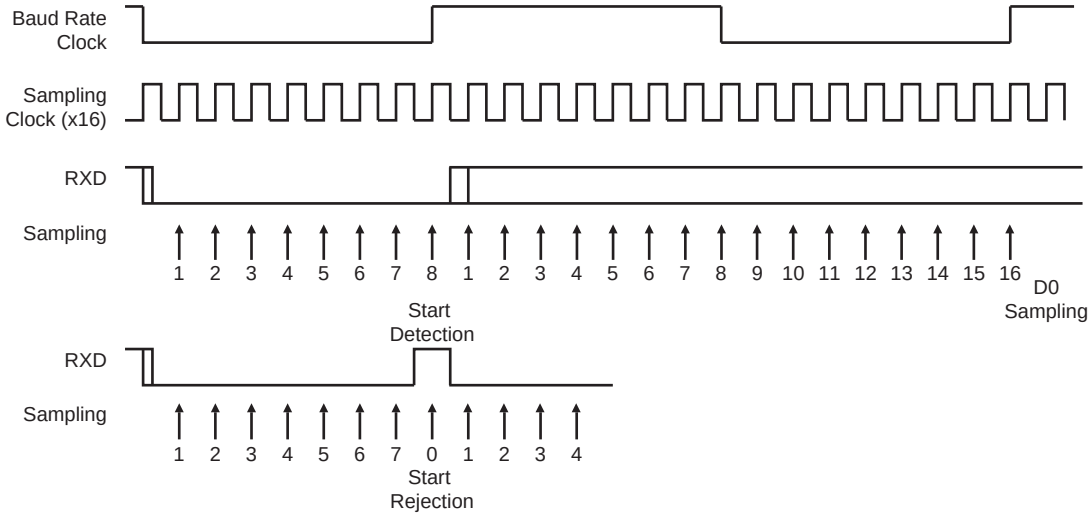
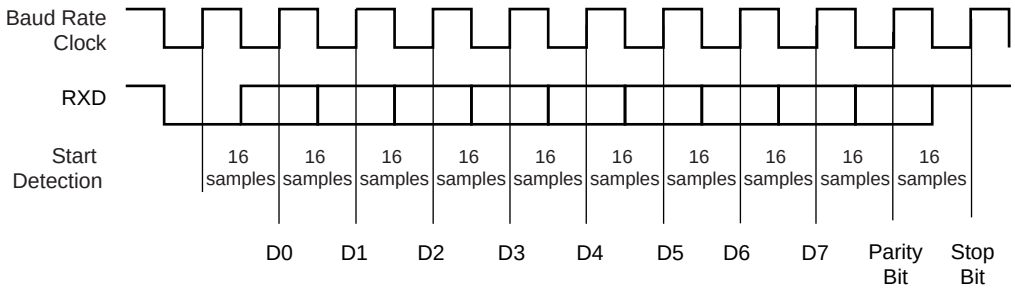


Figure 30-8. Asynchronous Character Reception

Example: 8-bit, Parity Enabled



30.6.3.3 Synchronous Receiver

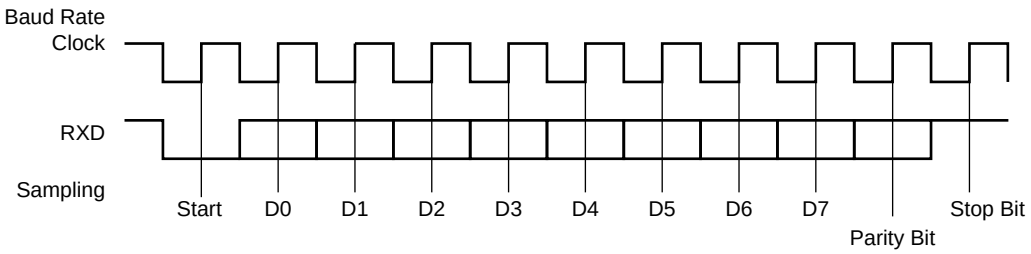
In Synchronous mode (SYNC = 1), the receiver samples the RXD signal on each rising edge of the baud rate clock. If a low level is detected, it is considered as a start. All data bits, the parity bit and the stop bits are sampled and the receiver waits for the next start bit. Synchronous mode operations provide a high-speed transfer capability.

Configuration fields and bits are the same as in Asynchronous mode.

Figure 30-9 illustrates a character reception in Synchronous mode.

Figure 30-9. Synchronous Mode Character Reception

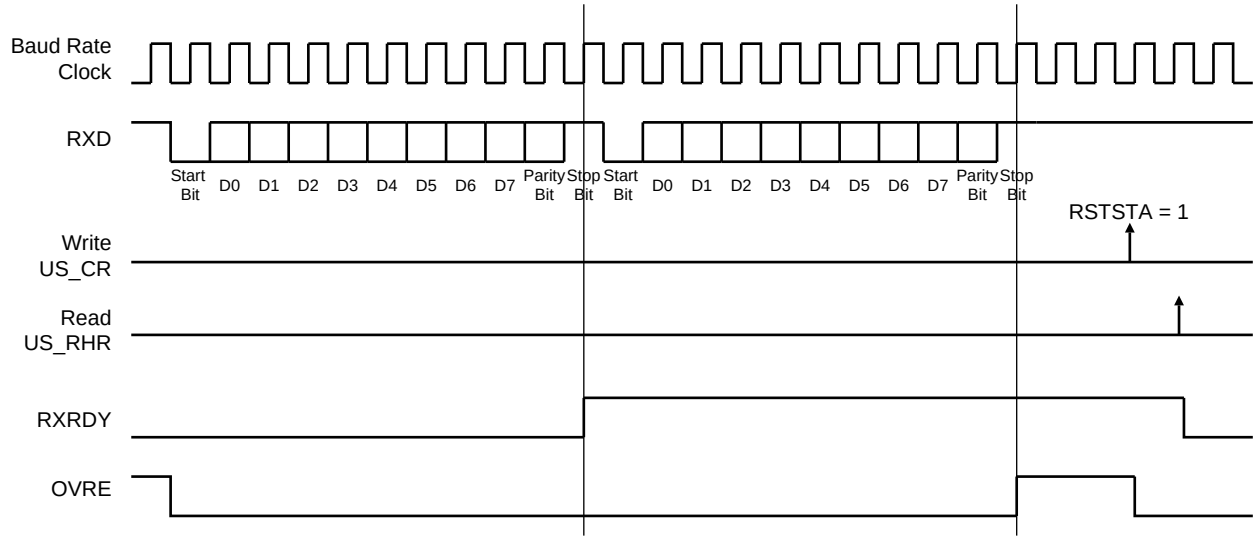
Example: 8-bit, Parity Enabled 1 Stop



30.6.3.4 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 30-10. Receiver Status



30.6.3.5 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 Section 30.6.3.6 "Multidrop Mode". Even and odd parity bit generation and error detection are supported.

If even parity is selected, the parity generator of the transmitter drives the parity bit to 0 if a number of 1s in the character data bit is even, and to 1 if the number of 1s is odd. Accordingly, the receiver parity checker counts the number of received 1s and reports a parity error if the sampled parity bit does not correspond. If odd parity is selected, the parity generator of the transmitter drives the parity bit to 1 if a number of 1s in the character data bit is even, and to 0 if the number of 1s is odd. Accordingly, the receiver parity checker counts the number of received 1s and reports a parity error if the sampled parity bit does not correspond. If the mark parity is used, the parity generator of the transmitter drives the parity bit to 1 for all characters. The receiver parity checker reports an error if the parity bit is sampled to 0. If the space parity is used, the parity generator of the transmitter drives the parity bit to 0 for all characters. The receiver parity checker reports an error if the parity bit is sampled to 1. If parity is disabled, the transmitter does not generate any parity bit and the receiver does not report any parity error.

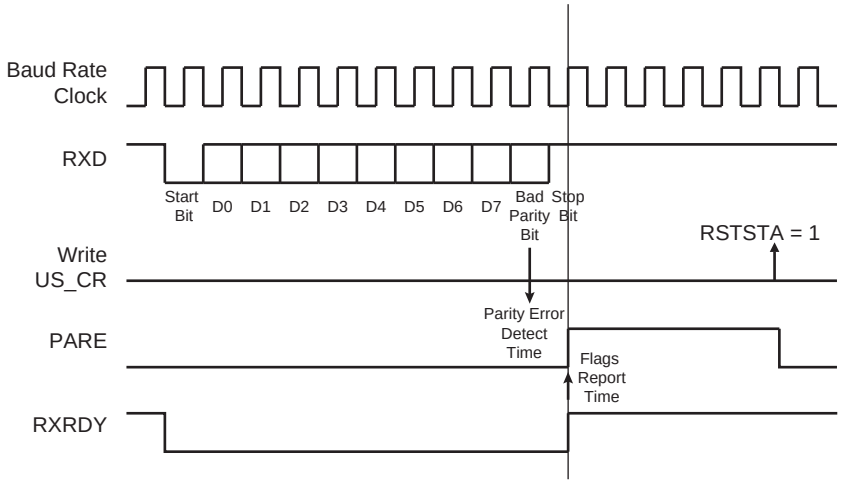
Table 30-9 shows an example of the parity bit for the character 0x41 (character ASCII "A") depending on the configuration of the USART. Because there are two bits set to 1 in the character value, the parity bit is set to 1 when the parity is odd, or configured to 0 when the parity is even.

Table 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 30-11 illustrates the parity bit status setting and clearing.

Figure 30-11. Parity Error



30.6.3.6 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.

30.6.3.7 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 30-12, 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 30-12. Timeguard Operations

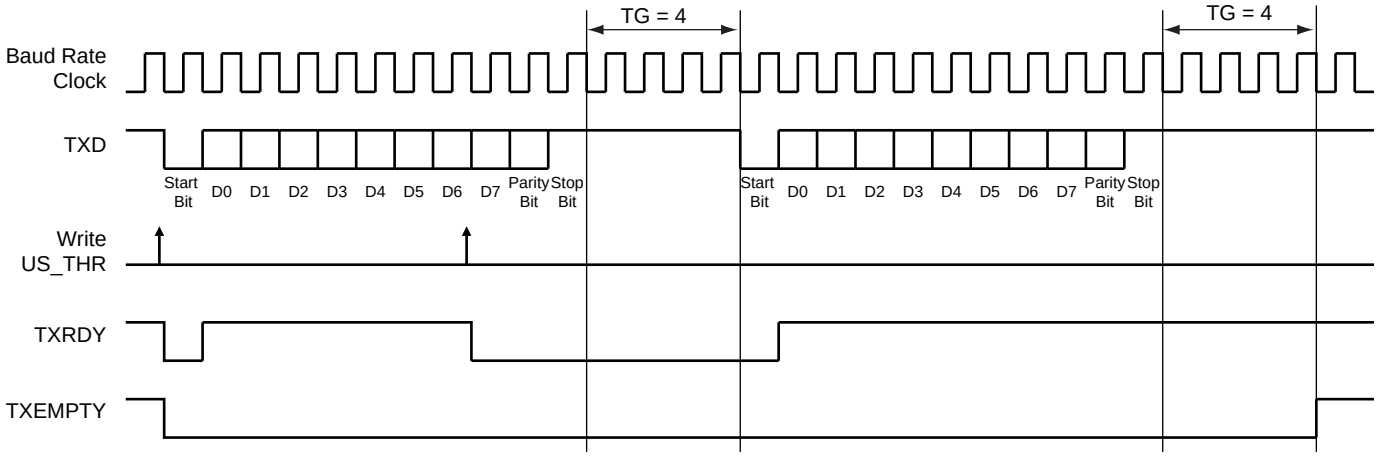


Table 30-10 indicates the maximum length of a timeguard period that the transmitter can handle in relation to the function of the baud rate.

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

30.6.3.8 Receiver Timeout

The Receiver Timeout provides support in handling variable-length frames. This feature detects an idle condition on the RXD line. When a timeout 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 timeout delay period (during which the receiver waits for a new character) is programmed in the TO field of the Receiver Timeout register (US_RTOR). If the TO field is written to 0, the Receiver Timeout is disabled and no timeout is detected. The TIMEOUT bit in the US_CSR remains at 0. Otherwise, the receiver loads a 16-bit counter with the value programmed in TO. This counter is decremented at each bit period and reloaded each time a new character is received. If the counter reaches 0, the TIMEOUT bit in US_CSR rises.

Then, the user can either:

- Stop the counter clock until a new character is received. This is performed by writing a ‘1’ to the STTTO (Start Timeout) bit in the US_CR. In this case, the idle state on RXD before a new character is received does not provide a timeout. This prevents having to handle an interrupt before a character is received, and allows waiting for the next idle state on RXD after a frame is received.
- Obtain an interrupt while no character is received. This is performed by writing a ‘1’ to the RETTO (Reload and Start Timeout) 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 timeout can be handled, for example when no key is pressed on a keyboard.

Figure 30-13 shows the block diagram of the Receiver Timeout feature.

Figure 30-13. Receiver Timeout Block Diagram

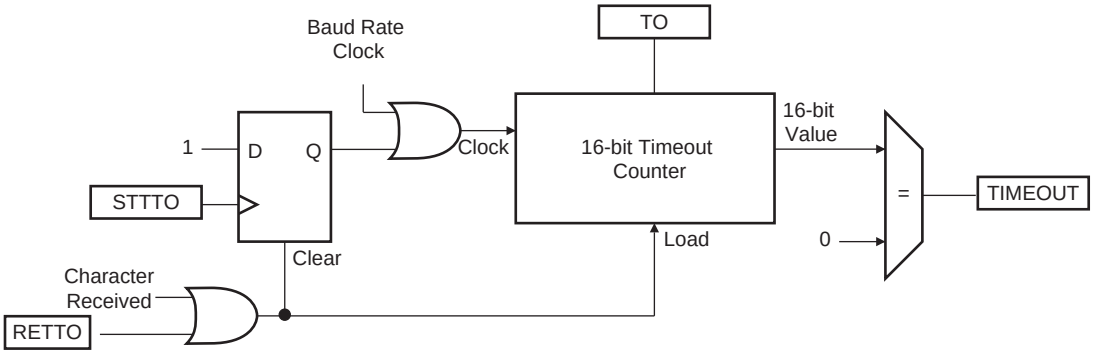


Table 30-11 gives the maximum timeout period for some standard baud rates.

Table 30-11. Maximum Timeout Period

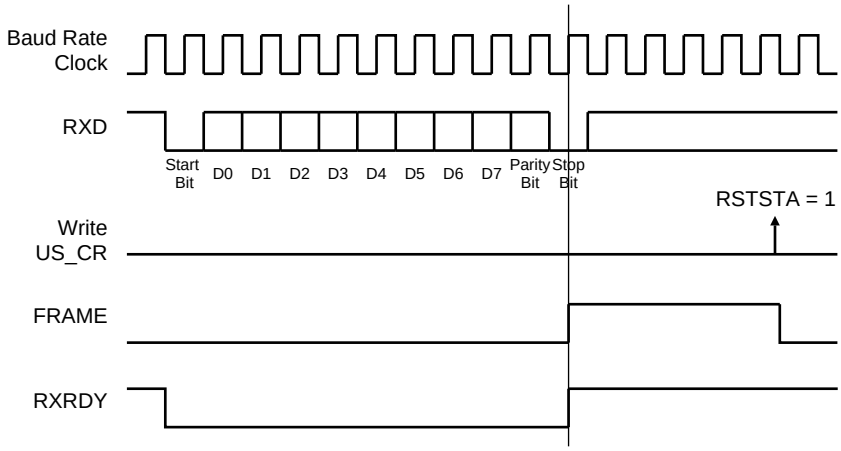
Baud Rate (Bit/s)	Bit Time (µs)	Timeout (ms)
600	1,667	109,225
1,200	833	54,613
2,400	417	27,306
4,800	208	13,653
9,600	104	6,827
14,400	69	4,551
19,200	52	3,413
28,800	35	2,276
38,400	26	1,704
56,000	18	1,170
57,600	17	1,138
200,000	5	328

30.6.3.9 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 30-14. Framing Error Status



30.6.3.10 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.

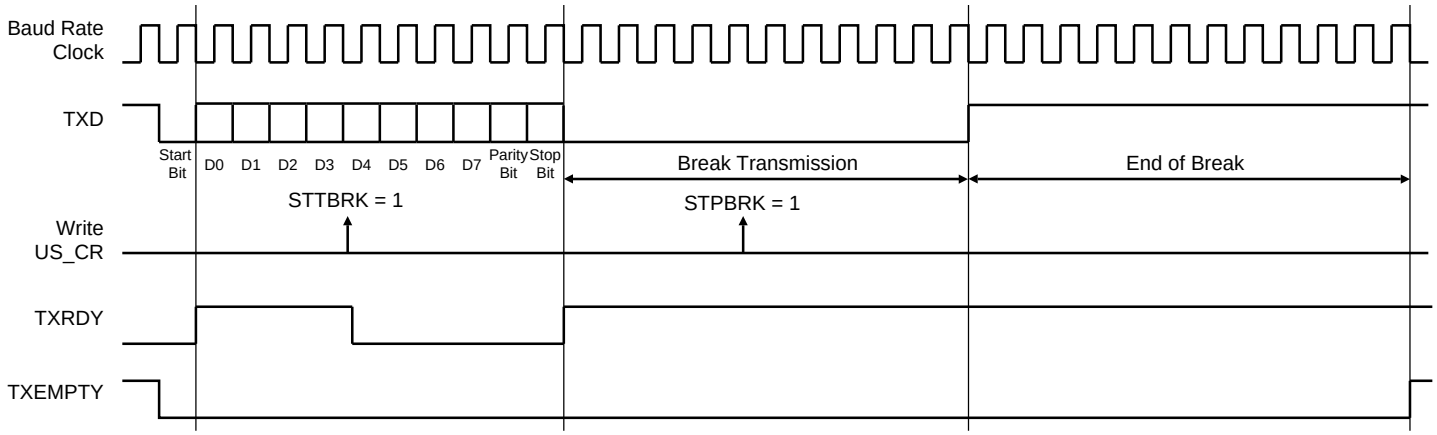
Writing US_CR with both STTBRK and STPBRK bits to ’1’ can lead to an unpredictable result. All STPBRK commands requested without a previous STTBRK command are ignored. A byte written into the Transmit Holding register while a break is pending, but not started, is ignored.

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 30-15 illustrates the effect of both the Start Break (STTBRK) and Stop Break (STPBRK) commands on the TXD line.

Figure 30-15. Break Transmission



30.6.3.11 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.

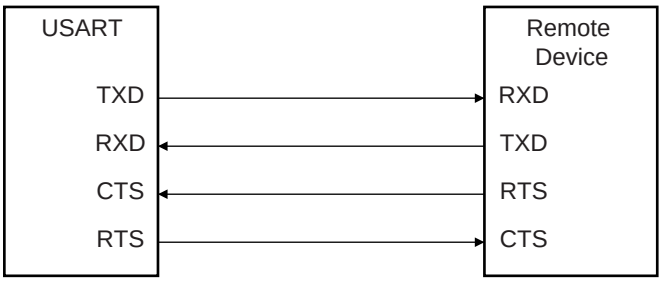
When the low stop bit is detected, the receiver asserts the RXBRK bit in US_CSR. This bit may be cleared by writing US_CR with the bit RSTSTA to ’1’.

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.

30.6.3.12 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 30-16.

Figure 30-16. 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 30-17. RTS Line Software Control when US_MR.USART_MODE = 2

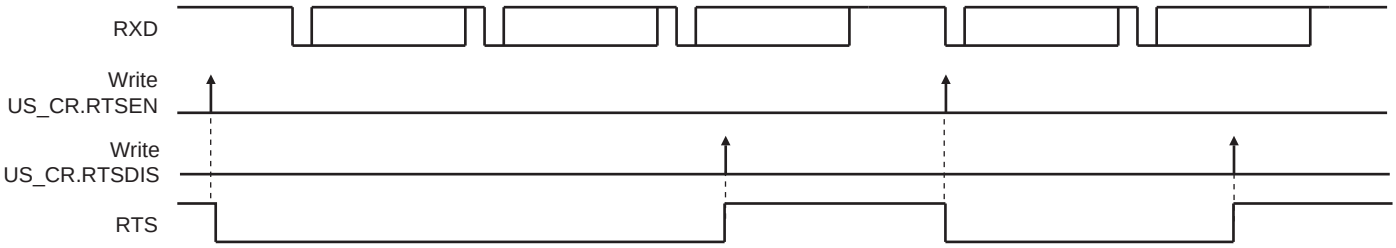


Figure 30-18 shows how the receiver operates if hardware handshaking is enabled. The RTS pin is driven high if the receiver is disabled and if the status RXBUFF (Receive Buffer Full) coming from the PDC channel is high. Normally, the remote device does not start transmitting while its CTS pin (driven 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

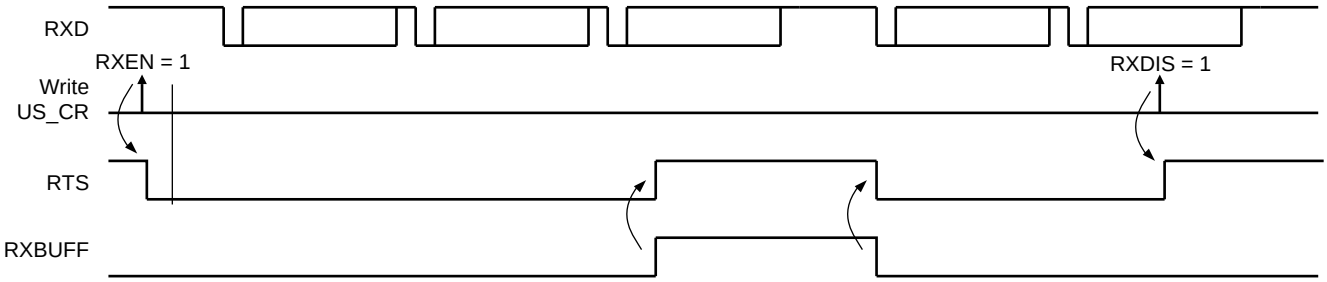
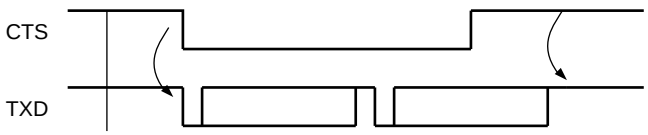


Figure 30-19 shows how the transmitter operates if hardware handshaking is enabled. The CTS pin disables the transmitter. If a character is being processed, the transmitter is disabled only after the completion of the current character and transmission of the next character happens as soon as the pin CTS falls.

Figure 30-19. Transmitter Behavior when Operating with Hardware Handshaking



30.6.4 ISO7816 Mode

The USART features an ISO7816-compatible operating mode. This mode permits interfacing with smart cards and Security Access Modules (SAM) communicating through an ISO7816 link. Both T = 0 and T = 1 protocols defined by the ISO7816 specification are supported.

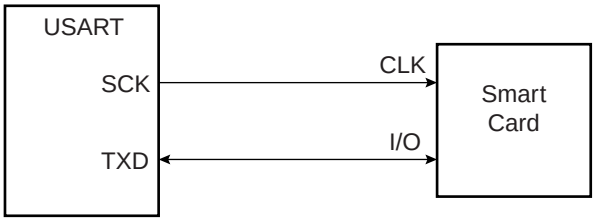
Setting the USART in ISO7816 mode is performed by writing US_MR.USART_MODE to the value 0x4 for protocol T = 0 and to the value 0x6 for protocol T = 1.

30.6.4.1 ISO7816 Mode Overview

The ISO7816 is a half duplex communication on only one bidirectional line. The baud rate is determined by a division of the clock provided to the remote device (see Section 30-2 ”Baud Rate Generator”).

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. US_MR.MSBF can be used to transmit LSB or MSB first. US_MR.PAR can be used to transmit in Normal or Inverse mode. See Section 30.7.3 ”USART Mode Register” and “PAR: Parity Type” .

The USART cannot operate concurrently in both Receiver and Transmitter modes as the communication is unidirectional at a time. It has to be configured according to the required mode by enabling or disabling either the receiver or the transmitter as desired. Enabling both the receiver and the transmitter at the same time in ISO7816 mode may lead to unpredictable results.

The ISO7816 specification defines an inverse transmission format. Data bits of the character must be transmitted on the I/O line at their negative value.

30.6.4.2 Protocol T = 0

In T = 0 protocol, a character is made up of one start bit, eight data bits, one parity bit and one guard time, which lasts two bit times. The transmitter shifts out the bits and does not drive the I/O line during the guard time.

If no parity error is detected, the I/O line remains 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 US_RHR. It sets US_SR.PARE so that the software can handle the error.

Figure 30-21. T = 0 Protocol without Parity Error

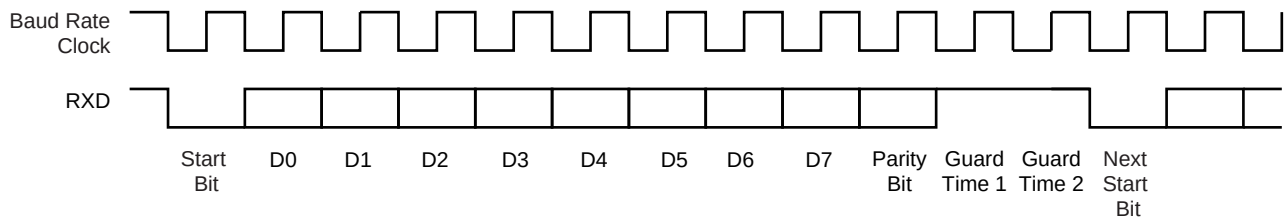
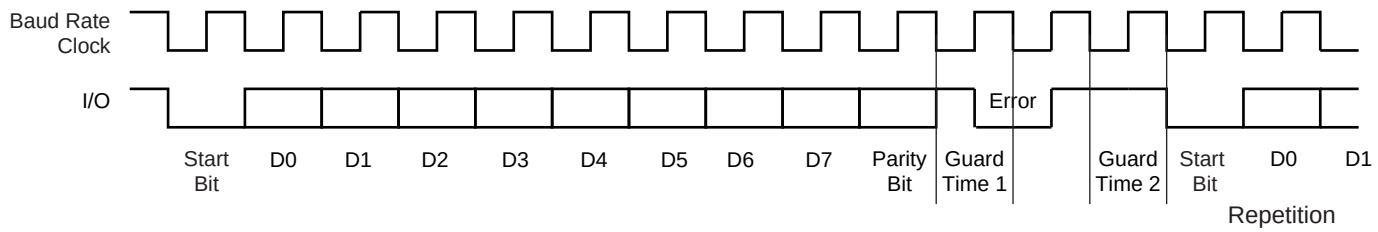


Figure 30-22. T = 0 Protocol with Parity Error



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 writing a '1' to the USART_MR.INACK. In this case, no error signal is driven on the I/O line even if a parity bit is detected.

Moreover, if INACK = 1, the erroneous received character is stored in the US_RHR, as if no error occurred and the RXRDY bit rises.

Transmit Character Repetition

When the USART is transmitting a character and gets a NACK, it can automatically repeat the character before moving on to the next one. Repetition is enabled by writing the US_MR.MAX_ITERATION to a value higher than 0. Each character can be transmitted up to eight times; the first transmission plus seven repetitions.

If MAX_ITERATION does not equal zero, the USART repeats the character as many times as the value loaded in MAX_ITERATION.

When the USART repetition number reaches MAX_ITERATION, and the last repeated character is not acknowledged, US_CSR.ITER is set. If the repetition of the character is acknowledged by the receiver, the repetitions are stopped and the iteration counter is cleared.

US_CSR.ITER can be cleared by writing a '1' to US_CR.RSTIT.

Disable Successive Receive NACK

The receiver can limit the number of successive NACKs sent back to the remote transmitter. This is programmed by setting US_MR.DSNACK. The maximum number of NACKs transmitted is configured in US_MR.MAX_ITERATION. As soon as MAX_ITERATION is reached, no error signal is driven on the I/O line and US_CSR.ITER is set.

30.6.4.3 Protocol T = 1

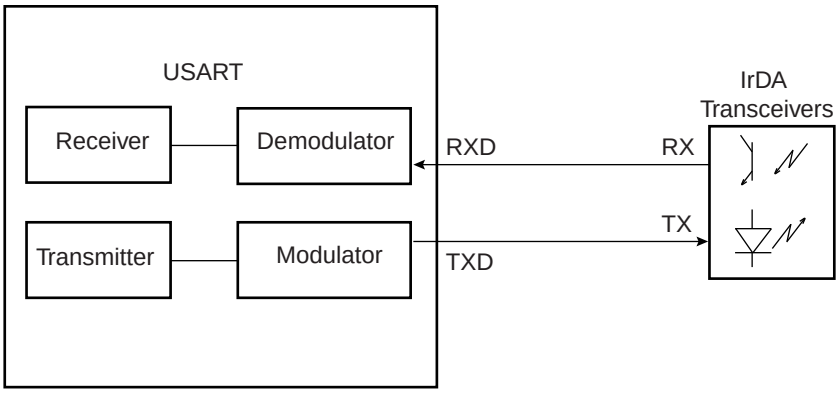
When operating in ISO7816 protocol T = 1, the transmission is similar to an asynchronous format with only one stop bit. The parity is generated when transmitting and checked when receiving. Parity error detection sets US_CSR.PARE.

30.6.5 IrDA Mode

The USART features an IrDA mode supplying half-duplex point-to-point wireless communication. It embeds the modulator and demodulator which allows a glueless connection to the infrared transceivers, as shown in Figure 30-23. The modulator and demodulator are compliant with the IrDA specification version 1.1 and support data transfer speeds ranging from 2.4 kbit/s to 115.2 kbit/s.

The USART IrDA mode is enabled by setting US_MR.USART_MODE to the value 0x8. The IrDA Filter register (US_IF) is used to configure the demodulator filter. The USART transmitter and receiver operate in a normal Asynchronous mode and all parameters are accessible. Note that the modulator and the demodulator are activated.

Figure 30-23. Connection to IrDA Transceivers



The receiver and the transmitter must be enabled or disabled depending on the direction of the transmission to be managed.

To receive IrDA signals, the following must be done:

- Disable TX and enable RX.
- Configure the TXD pin as PIO and set it as an output to 0 (to avoid LED transmission). Disable the internal pull-up (better for power consumption).
- Receive data.

30.6.5.1 IrDA Modulation

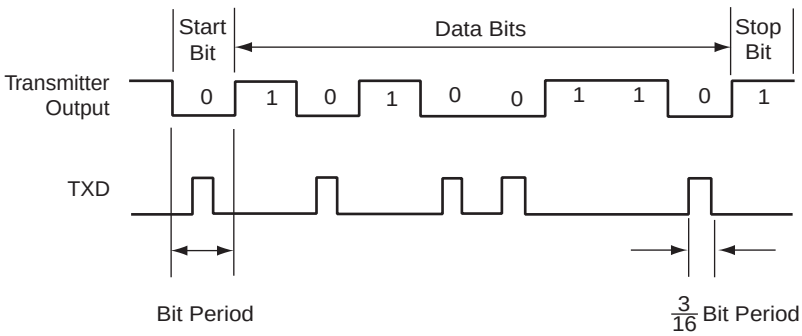
For baud rates up to and including 115.2 kbit/s, the RZI modulation scheme is used. “0” is represented by a light pulse of 3/16th of a bit time. Some examples of signal pulse duration are shown in Table 30-12.

Table 30-12. IrDA Pulse Duration

Baud Rate	Pulse Duration (3/16)
2.4 kbit/s	78.13 µs
9.6 kbit/s	19.53 µs
19.2 kbit/s	9.77 µs
38.4 kbit/s	4.88 µs
57.6 kbit/s	3.26 µs
115.2 kbit/s	1.63 µs

Figure 30-24 shows an example of character transmission.

Figure 30-24. IrDA Modulation



30.6.5.2 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

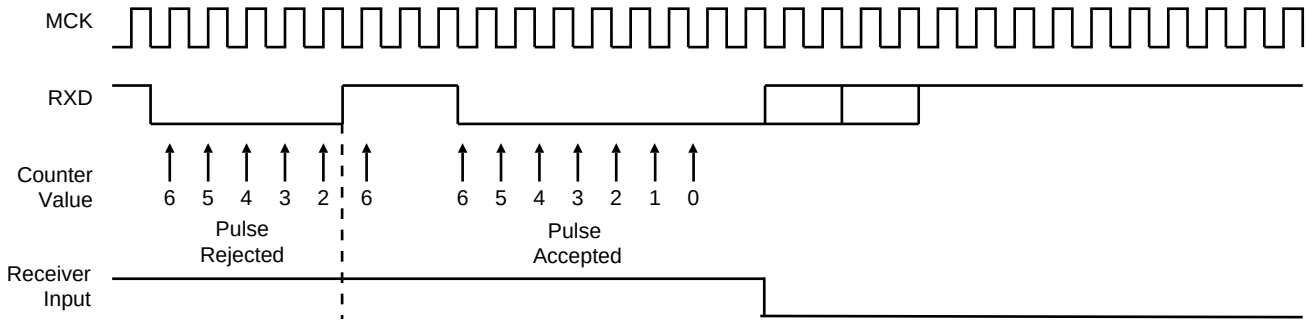
Peripheral Clock	Baud Rate (Bit/s)	CD	Baud Rate Error	Pulse Time (µs)
3,686,400	115,200	2	0.00%	1.63
20,000,000	115,200	11	1.38%	1.63
32,768,000	115,200	18	1.25%	1.63
40,000,000	115,200	22	1.38%	1.63
3,686,400	57,600	4	0.00%	3.26
20,000,000	57,600	22	1.38%	3.26
32,768,000	57,600	36	1.25%	3.26
40,000,000	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
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

30.6.5.3 IrDA Demodulator

The demodulator is based on the IrDA Receive filter comprised of an 8-bit down counter which is loaded with the value programmed in US_IF. When a falling edge is detected on the RXD pin, the Filter Counter starts counting down at the peripheral clock speed. If a rising edge is detected on the RXD pin, the counter stops and is reloaded with US_IF. If no rising edge is detected when the counter reaches 0, the input of the receiver is driven low during one bit time.

Figure 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:

$t_{peripheral\ clock}$* (IRDA_FILTER + 3) < 1.41 µs

As the IrDA mode uses the same logic as the ISO7816, note that US_FIDI.FI_DI_RATIO must be set to a value higher than 0 in order to make sure that IrDA communications operate correctly.

30.6.6 USART Comparison Function on Received Character

The effect of a comparison match changes if the system is in Wait or Active mode.

In Wait mode, if asynchronous partial wakeup is enabled, a system wakeup is performed (see Section 30.6.7 ”USART Asynchronous and Partial Wakeup (SleepWalking)”).

In Active mode, the CMP flag in US_CSR is raised. It is set when the received character matches the conditions programmed in the USART Comparison Register (US_CMPR). The CMP flag is set as soon as US_RHR is loaded with the new received character. The CMP flag is cleared by writing a one to the RSTSTA bit in US_CR.

US_CMPR (see Section 30.7.28 ”USART Comparison Register”) can be programmed to provide different comparison methods, as described below:

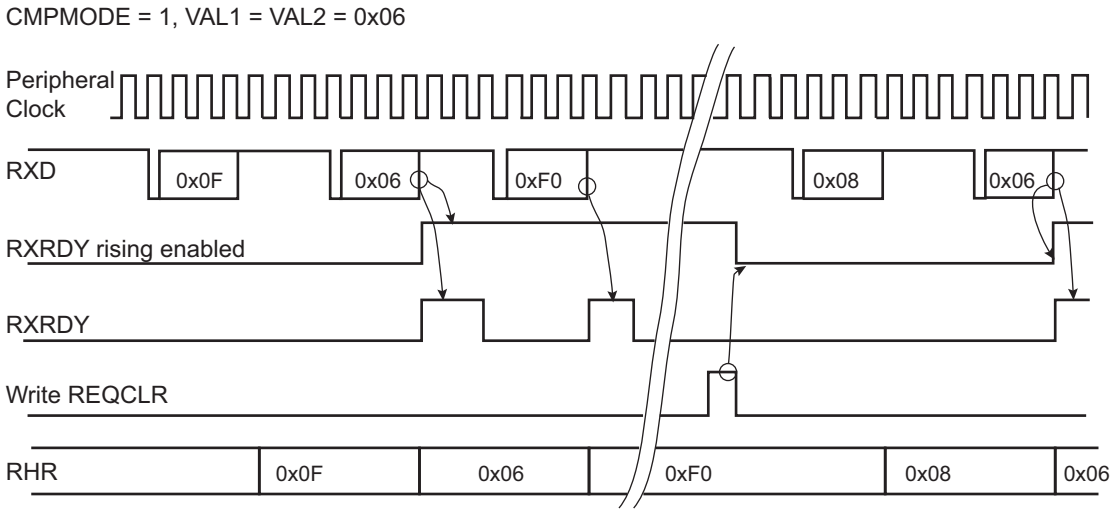
- If VAL1 equals VAL2, then the comparison is performed on a single value and the flag is set to ’1’ if the received character equals VAL1.
- If VAL1 is strictly lower than VAL2, then any value between VAL1 and VAL2 sets the CMP flag.
- If VAL1 is strictly higher than VAL2, then the CMP flag is set to ’1’ if any received character equals VAL1 or VAL2.

When the CMPMODE bit is cleared in US_CMPR register, all received data is loaded in US_RHR and the CMP flag is providing the status of the comparison result.

By setting the CMPMODE bit, the comparison result triggers the start of the US_RHR loading (see Figure 30-26). The trigger condition exists as soon as the received character value matches the conditions defined by VAL1, VAL2 and CMPPAR in US_CMPR. The comparison trigger event is restarted by writing a ’1’ to the REQCLR bit in US_CR.

The value programmed in VAL1 and VAL2 fields must not exceed the maximum value of the received character (see CHRL field in US_MR).

Figure 30-26. Receive Holding Register Management



30.6.7 USART Asynchronous and Partial Wakeup (SleepWalking)

This operating mode is a means of data preprocessing that qualifies an incoming event, thus allowing the USART to decide whether or not to wake up the system. Asynchronous and partial wakeup is mainly used when the system is in Wait mode (refer to the section “Power Management Controller (PMC)” for details). It can also be enabled when the system is fully running.

Asynchronous and partial wakeup (SleepWalking) requires the USART module to be programmed in UART mode (SYNC=0 in US_MR).

The maximum USART bit rate that can be achieved when asynchronous and partial wakeup is enabled is 19200.

The US_RHR must be read before enabling the asynchronous and partial wakeup.

When asynchronous and partial wakeup is enabled for the USART (refer to the section “Power Management Controller (PMC)”), the PMC decodes a clock request from the USART. The request is generated as soon as there is a falling edge on the RXD line as this may indicate the beginning of a start bit. If the system is in Wait mode (processor and peripheral clocks switched off), the PMC restarts the fast RC oscillator and provides the clock only to the USART.

As soon as the clock is provided by the PMC, the USART processes the received frame and compares the received character with VAL1 and VAL2 in US_CMPR (Section 30.7.28 ”USART Comparison Register”).

The USART instructs the PMC to disable the peripheral clock if the received character value does not meet the conditions defined by the VAL1 and VAL2 fields in US_CMPR (see Figure 30-28).

If the received character value meets the conditions, the USART instructs the PMC to exit the system from Wait mode (see Figure 30-27).

The VAL1 and VAL2 fields can be programmed to provide different comparison methods and thus matching conditions.

- If VAL1 equals VAL2, then the comparison is performed on a single value and the wakeup is triggered if the received character equals VAL1.
- If VAL1 is strictly lower than VAL2, then any value between VAL1 and VAL2 wakes up the system.
- If VAL1 is strictly higher than VAL2, then the wakeup is triggered if any received character equals VAL1 or VAL2.
- If VAL1 = 0 and VAL2 = 511, the wakeup is triggered as soon as a character is received.

The matching condition can be configured to include the parity bit (US_CMPR.CMPPAR). Thus, if the received data matches the comparison condition defined by VAL1 and VAL2 but a parity error is encountered, the matching condition is cancelled and the USART instructs the PMC to disable the clock (see Figure 30-28).

If the processor and peripherals are running, the USART can be configured in Asynchronous and Partial Wakeup mode by enabling the PMC_SLPWK_ER (refer to the section “Power Management Controller (PMC)”). When some activity is detected on the receive line, the USART requests the clock from the PMC and the comparison is performed. If there is a comparison match, the USART continues to request the clock. If there is no match, the clock is switched off for the USART only, until a new activity is detected.

The CMPMODE configuration has no effect when Asynchronous and Partial Wakeup mode is enabled for the USART (refer to register PMC_SLPWK_ER in the section “Power Management Controller (PMC)”).

When the system is in Active mode and the USART enters Asynchronous and Partial Wakeup mode, the RXRDY flag must be programmed as the unique source of the USART interrupt.

When the system exits Wait mode as the result of a matching condition, the RXRDY flag is used to determine if the USART is the source for the exit from Wait mode.

Figure 30-27. Asynchronous Wakeup Use Case Examples

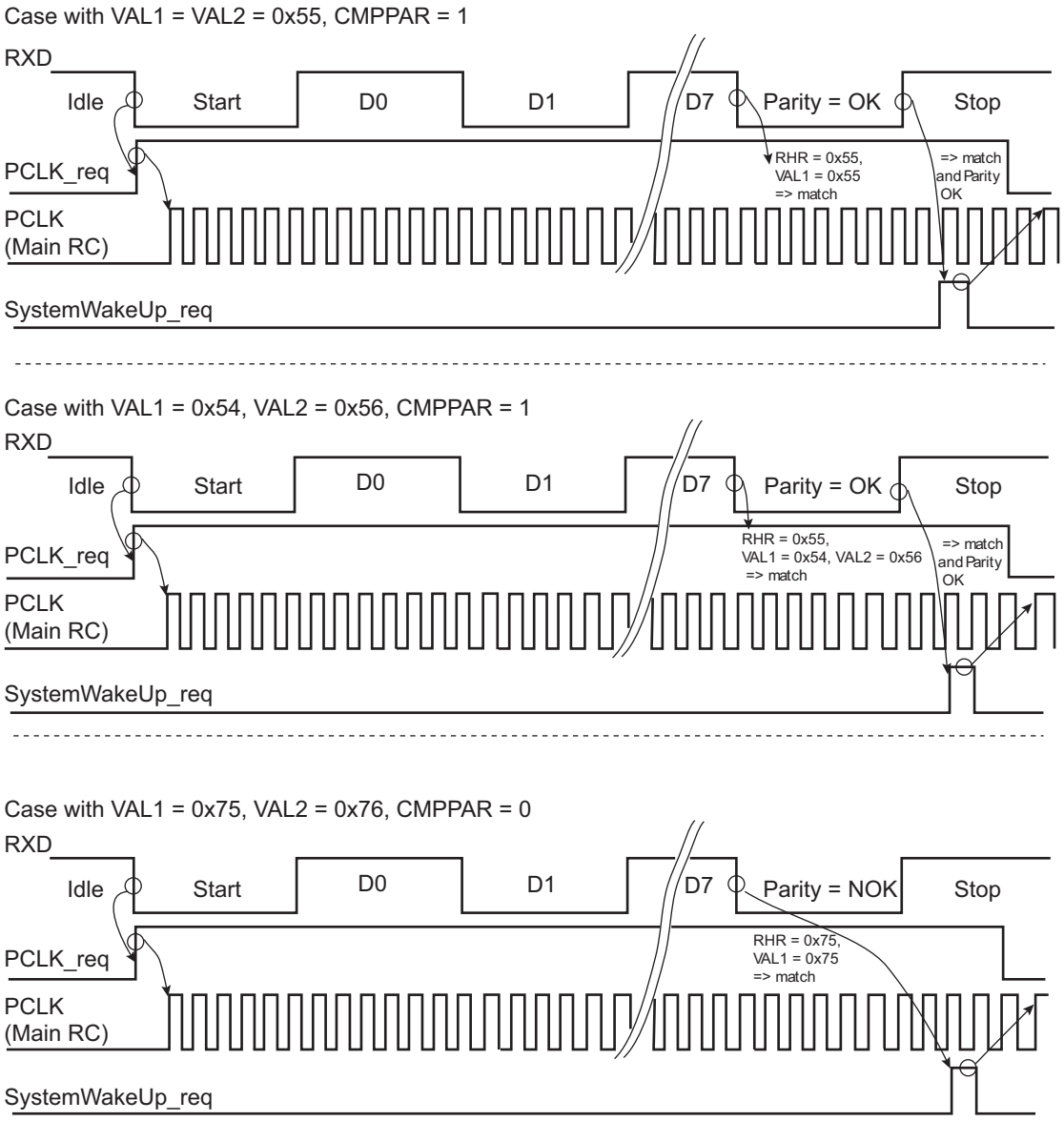
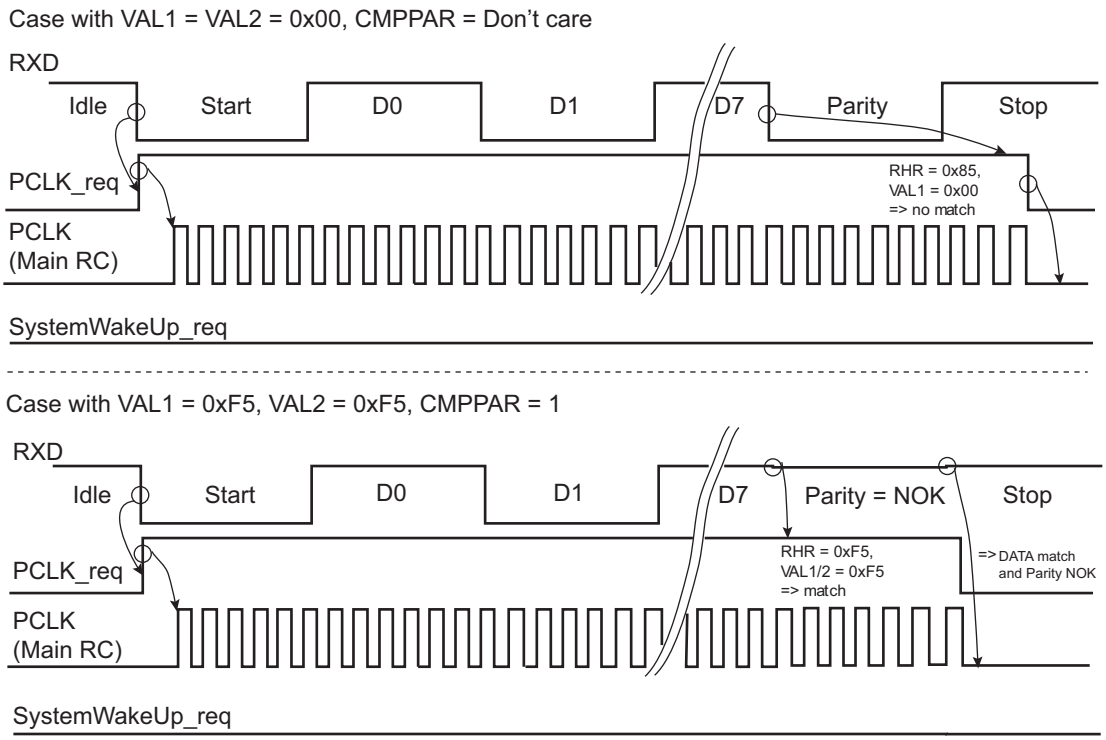


Figure 30-28. Asynchronous Event Generating Only Partial Wakeup



30.6.8 SPI Mode

The USART embeds a mode providing a basic Serial Peripheral Interface (SPI).

The SPI bus used in this 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 bit 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.

30.6.8.1 Modes of Operation

The USART can operate in SPI Master mode or in SPI Slave mode.

Operation in SPI Master mode is programmed by writing 0xE to the USART_MODE field in US_MR. In this case, the SPI lines must be connected as described below:

- The MOSI line is driven by the output pin TXD.
- The MISO line drives the input pin RXD.
- The SCK line is driven by the output pin SCK.
- The NSS line is driven by the output pin RTS.

Operation in SPI Slave mode is programmed by writing 0xF to 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 30.6.8.4 ”Receiver and Transmitter Control”.

30.6.8.2 Bit Rate

In SPI mode, the bit rate generator operates in the same way as in USART Synchronous mode (see Section 30.6.1.3 ”Baud Rate in Synchronous Mode or SPI Mode”). However, some restrictions apply:

In SPI Master mode:

- The external clock SCK must not be selected (USCLKS ≠ 0x3), and 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 greater than or equal to 6.
- If the divided peripheral clock is selected, the value programmed in CD must be even to ensure a 50:50 mark/space ratio on the SCK pin; this value can be odd if the peripheral clock is selected.

In SPI Slave mode:

- The external clock (SCK) selection is forced regardless of the value of the 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.

30.6.8.3 Data Transfer

Up to nine data bits are successively shifted out on the TXD pin at each rising or falling edge (depending of CPOL and CPHA) of the programmed serial clock. There is no Start bit, no Parity bit and no Stop bit.

The number of data bits is selected by the CHRL field and the MODE 9 bit in the US_MR. The nine bits are selected by setting the MODE 9 bit regardless of the CHRL field. The MSB data bit is always sent first in SPI mode (Master or Slave).

Four combinations of polarity and phase are available for data transfers. The clock polarity is programmed with the CPOL bit in the US_MR. The clock phase is programmed with the CPHA bit. These two parameters determine the edges of the clock signal upon which data is driven and sampled. Each of the two parameters has two possible states, resulting in four possible combinations that are incompatible with one another. Thus, a master/slave pair must use the same parameter pair values to communicate. If multiple slaves are used and fixed in different configurations, the master must reconfigure itself each time it needs to communicate with a different slave.

Table 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 30-29. SPI Transfer Format (CPHA = 1, 8 bits per transfer)

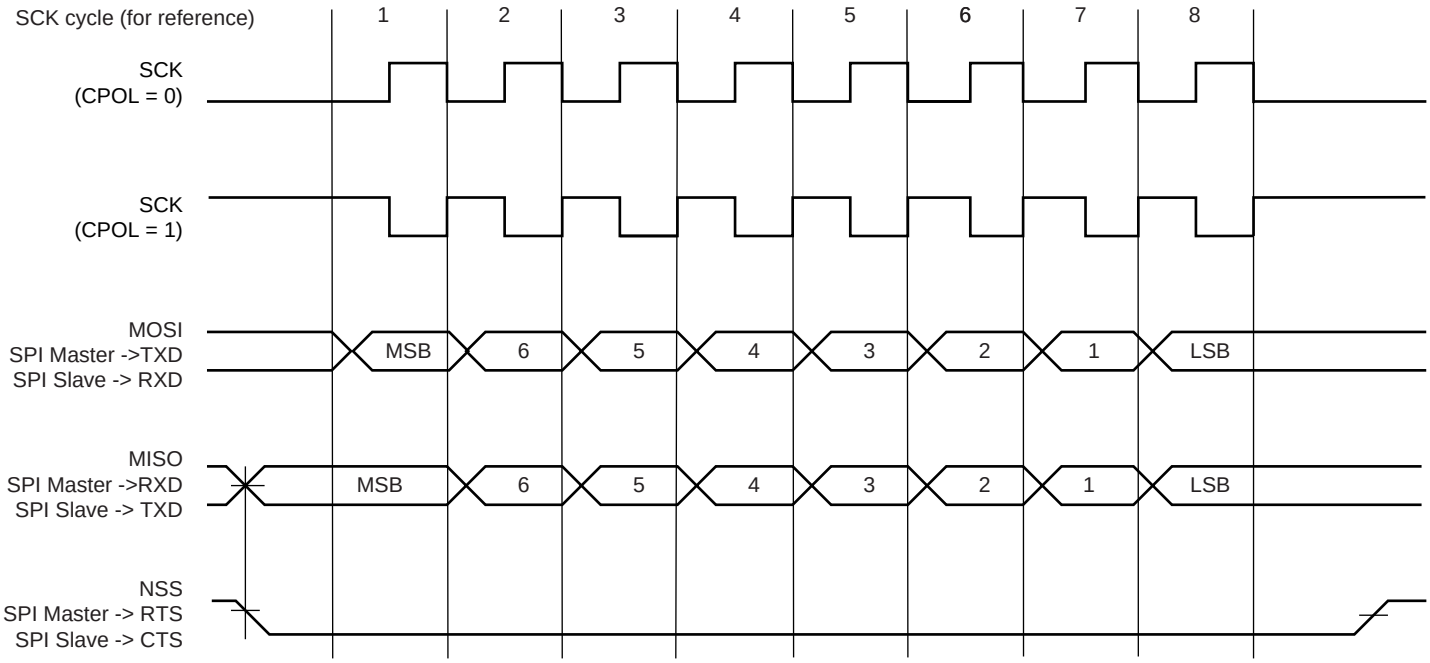
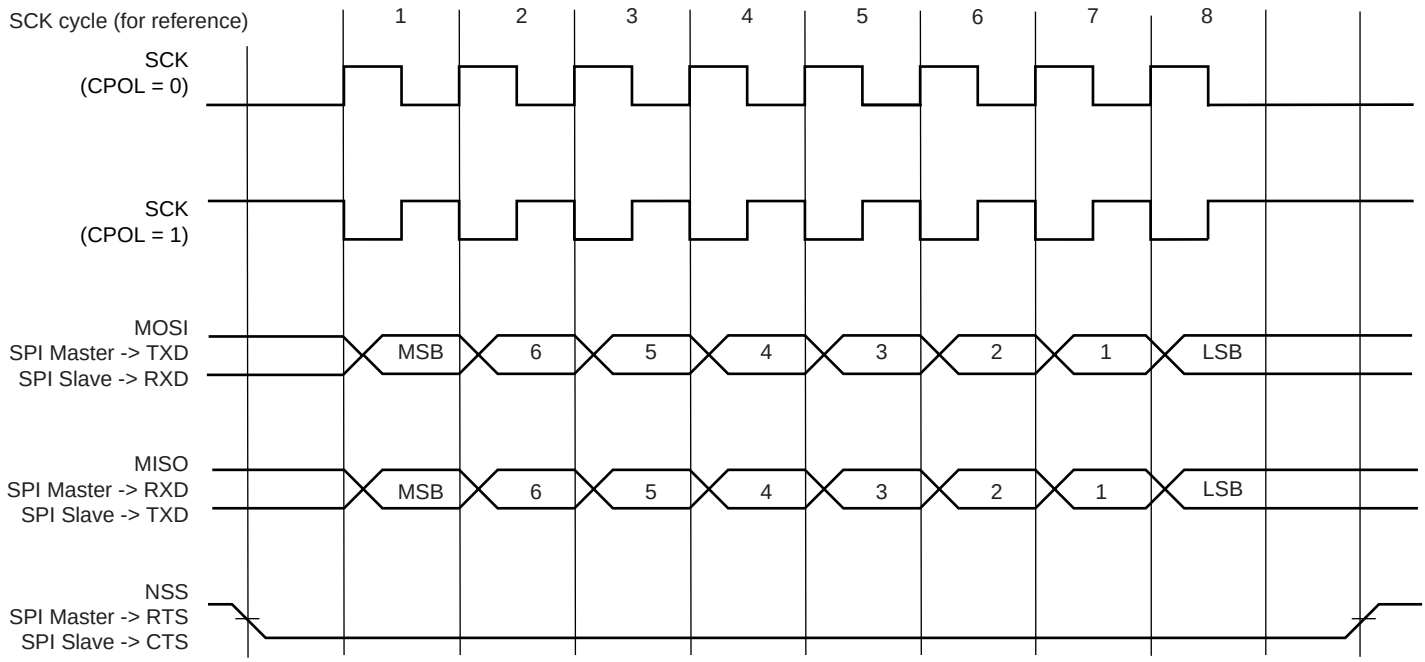


Figure 30-30. SPI Transfer Format (CPHA = 0, 8 bits per transfer)



30.6.8.4 Receiver and Transmitter Control

See Section 30.6.2 "Receiver and Transmitter Control".

30.6.8.5 Character Transmission

The characters are sent by writing in US_THR. An additional condition for transmitting a character can be added when the USART is configured in SPI Master mode. In the "USART Mode Register (SPI_MODE)" (USART_MR), the value configured on WRDBT can prevent any character transmission (even if US_THR has been written) while the receiver side is not ready (character not read). When WRDBT equals '0', the character is transmitted whatever the receiver status. If WRDBT is set to '1', the transmitter waits for the US_RHR to be read before transmitting the character (RXRDY flag cleared), thus preventing any overflow (character loss) on the receiver side.

The chip select line is deasserted for a period equivalent to three bits between the transmission of two data.

The transmitter reports two status bits in US_CSR: TXRDY (Transmitter Ready), which indicates that US_THR is empty and TXEMPTY, which indicates that all the characters written in US_THR have been processed. When the current character processing is completed, the last character written in US_THR is transferred into the Shift register of the transmitter and US_THR becomes empty, thus TXRDY rises.

Both TXRDY and TXEMPTY bits are low when the transmitter is disabled. Writing a character in US_THR while TXRDY is low has no effect and the written character is lost.

If the USART is in SPI Slave mode and if a character must be sent while the US_THR is empty, the UNRE (Underrun Error) bit is set. The TXD transmission line stays at high level during all this time. The UNRE bit is cleared by writing a 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 t_{bit} (Time bit) before the transmission of the MSB bit and released at high level 1 t_{bit} after the transmission of the LSB bit. So, the slave select line (NSS) is always released between each character transmission and a minimum delay of 3 t_{bit} always inserted. However, in order to address slave devices supporting the CSAAT mode (Chip Select Active After Transfer), the slave select line (NSS) can be forced at low level by writing a 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 t_{bit} before the first serial clock cycle corresponding to the MSB bit.

30.6.8.6 Character Reception

When a character reception is completed, it is transferred to the 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 in the US_CR.

To ensure correct behavior of the receiver in SPI Slave mode, the master device sending the frame must ensure a minimum delay of 1 t_{bit} between each character transmission. The receiver does not require a falling edge of the slave select line (NSS) to initiate a character reception but only a low level. However, this low level must be present on the slave select line (NSS) at least 1 t_{bit} before the first serial clock cycle corresponding to the MSB bit.

30.6.8.7 Receiver Timeout

Because the receiver bit rate clock is active only during data transfers in SPI mode, a receiver timeout is impossible in this mode, whatever the timeout value is (field TO) in the US_RTOR.

30.6.9 LIN Mode

The LIN mode provides master node and slave node connectivity on a LIN bus.

The LIN (Local Interconnect Network) is a serial communication protocol which efficiently supports the control of mechatronic nodes in distributed automotive applications.

The main properties of the LIN bus are:

- Single master/multiple slaves concept
- Low-cost silicon implementation based on common UART/SCI interface hardware, an equivalent in software, or as a pure state machine.
- Self synchronization without quartz or ceramic resonator in the slave nodes
- Deterministic signal transmission
- Low cost single-wire implementation
- Speed up to 20 kbit/s

LIN provides cost efficient bus communication where the bandwidth and versatility of CAN are not required.

The LIN mode enables processing LIN frames with a minimum of action from the microprocessor.

30.6.9.1 Modes of Operation

The USART can act either as a LIN master node or as a LIN slave node.

The node configuration is selected by configuring US_MR.USART_MODE:

- LIN master node (USART_MODE = 0xA)
- LIN slave node (USART_MODE = 0xB)

In order to avoid unpredicted behavior, any change of the LIN node configuration must be followed by a software reset of the transmitter and of the receiver (except the initial node configuration after a hardware reset). See Section 30.6.9.3 ”Receiver and Transmitter Control”.

30.6.9.2 Baud Rate Configuration

See Section 30.6.1.1 ”Baud Rate in Asynchronous Mode”.

- LIN master node: The baud rate is configured in US_BRGR.
- LIN slave node: The initial baud rate is configured in US_BRGR. This configuration is automatically copied in the LIN Baud Rate register (US_LINBRR) when writing US_BRGR. After the synchronization procedure, the baud rate is updated in US_LINBRR.

30.6.9.3 Receiver and Transmitter Control

See Section 30.6.2 "Receiver and Transmitter Control".

30.6.9.4 Character Transmission

See Section 30.6.3.1 "Transmitter Operations".

30.6.9.5 Character Reception

See Section 30.6.3.4 "Receiver Operations".

30.6.9.6 Header Transmission (master Node Configuration)

All the LIN Frames start with a header which is sent by the Master node and consists of a Synch Break Field, Synch Field and Identifier Field.

So in Master node configuration, the frame handling starts with the sending of the header.

The header is transmitted as soon as the identifier is written in the USART LIN Identifier register (US_LINIR). At this moment the flag TXRDY falls.

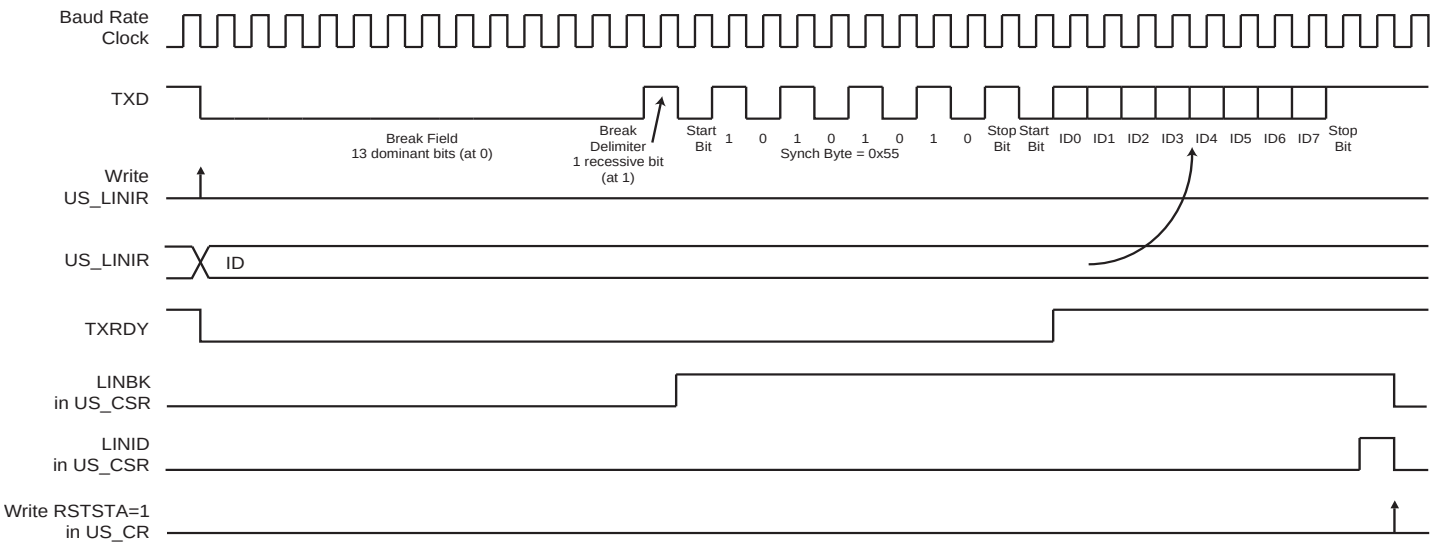
The Break Field, the Synch Field and the Identifier Field are sent automatically one after the other.

The Break Field consists of 13 dominant bits and 1 recessive bit, the Synch Field is the character 0x55 and the Identifier corresponds to the character written in US_LINIR. The Identifier parity bits can be automatically computed and sent (see Section 30.6.9.9 "Identifier Parity").

The flag TXRDY rises when the identifier character is transferred into the Shift register of the transmitter.

As soon as the Synch Break Field is transmitted, US_CSER.LINBK is set to '1'. Likewise, as soon as the Identifier Field is sent, the flag bit US_CSR.LINID is set to '1'. These flags are reset by writing a one to US_CR.RSTSTA.

Figure 30-31. Header Transmission



30.6.9.7 Header Reception (Slave Node Configuration)

All the LIN frames start with a header which is sent by the master node and consists of a Synch Break Field, Synch Field and Identifier Field.

In slave node configuration, the frame handling starts with the reception of the header.

The USART uses a break detection threshold of 11 nominal bit times at the actual baud rate. At any time, if 11 consecutive recessive bits are detected on the bus, the USART detects a Break Field. As long as a Break Field has not been detected, the USART stays idle and the received data are not taken in account.

When a Break Field has been detected, the flag LINBK in US_CSR is set to ’1’ and the USART expects the Synch Field character to be 0x55. This field is used to update the actual baud rate in order to stay synchronized (see Section 30.6.9.8 ”Slave Node Synchronization”). If the received Synch character is not 0x55, an Inconsistent Synch Field error is generated (see Section 30.6.9.14 ”LIN Errors”).

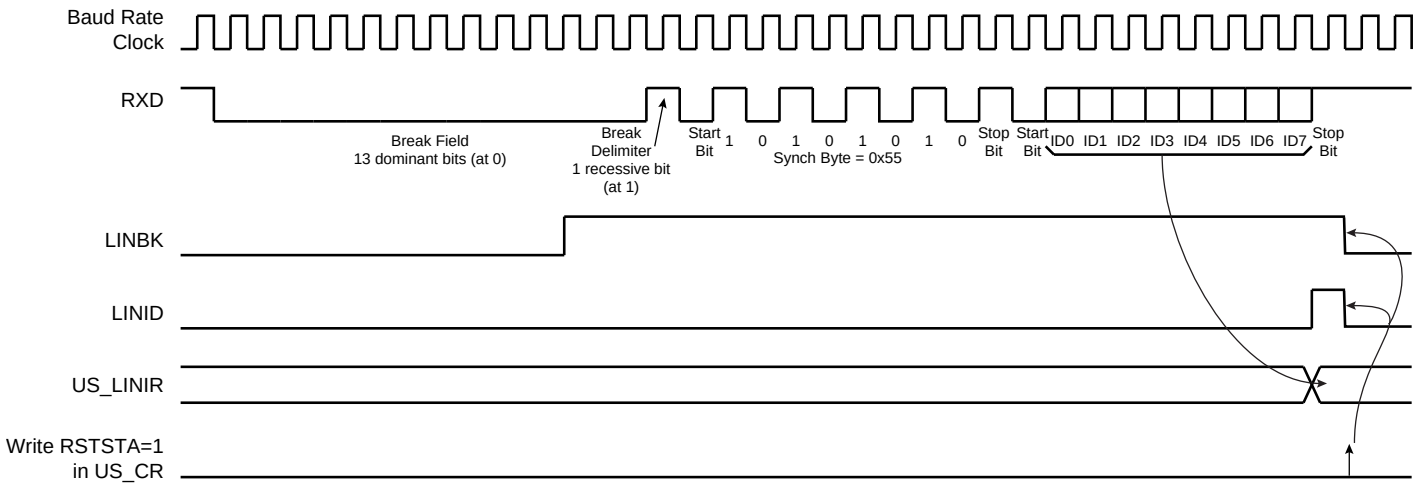
After receiving the Synch Field, the USART expects to receive the Identifier Field.

When the Identifier Field has been received, the flag bit US_CSR.LINID is set to ’1’. At this moment, US_LINIR.IDCHR is updated with the received character. The Identifier parity bits can be automatically computed and checked (see Section 30.6.9.9 ”Identifier Parity”).

If the Header is not entirely received within the time given by the maximum length of the header $t_{Header\_Maximum}$, the error flag US_CSR.LINHTE is set to ’1’.

The flag bits LINID, LINBK and LINHTE are reset by writing a one to US_CR.RSTSTA.

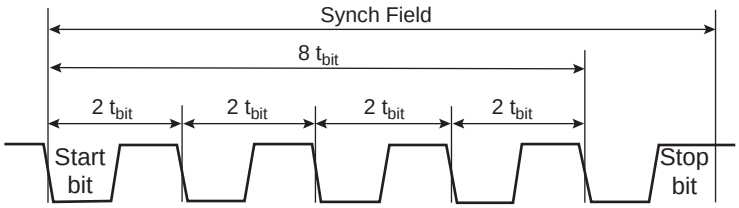
Figure 30-32. Header Reception



30.6.9.8 Slave Node Synchronization

The synchronization is done only in slave node configuration. The procedure is based on time measurement between falling edges of the Synch Field. The falling edges are available in distances of 2, 4, 6 and 8 bit times.

Figure 30-33. Synch Field



The time measurement is made by a 19-bit counter clocked by the sampling clock (see Section 30.6.1 ”Baud Rate Generator”).

When the start bit of the Synch Field is detected, the counter is reset. Then during the next 8 t_{bit} of the Synch Field, the counter is incremented. At the end of these 8 t_{bit}, the counter is stopped. At this moment, the 16 most significant bits of the counter (value divided by 8) give the new clock divider (LINCD) and the 3 least significant bits of this value (the remainder) give the new fractional part (LINFP).

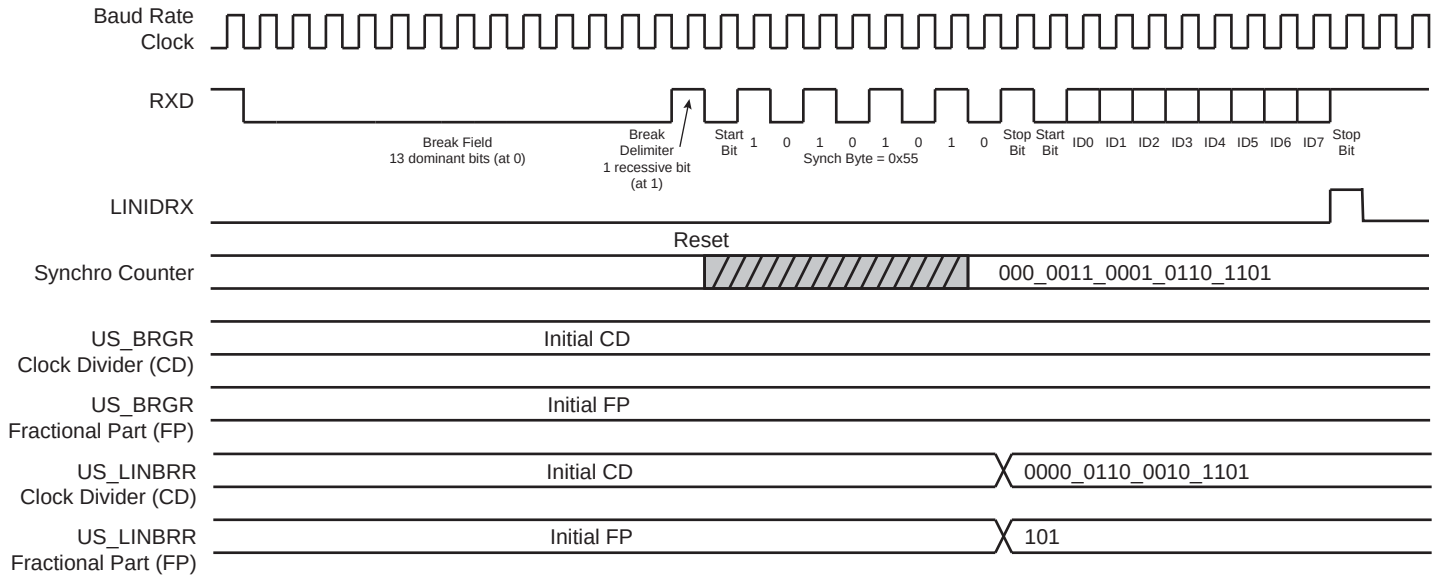
Once the Synch Field has been entirely received, the clock divider (LINCD) and the fractional part (LINFP) are updated in the LIN Baud Rate register (US_LINBRR) with the computed values if the Synchronization is not disabled by the SYNCDIS bit in the LIN Mode register (US_LINMR).

After reception of the Synch Field:

- If it appears that the computed baud rate deviation compared to the initial baud rate is superior to the maximum tolerance FTol_Unsynch (±15%), then the clock divider (LINCD) and the fractional part (LINFP) are not updated, and the error flag US_CSR.LINSTE is set to ’1’.
- If it appears that the sampled Synch character is not equal to 0x55, then the clock divider (LINCD) and the fractional part (LINFP) are not updated, and the error flag US_CSR.LINISFE is set to ’1’.

Flags LINSTE and LINISFE are reset by writing US_CR.RSTSTA to ’1’.

Figure 30-34. Slave Node Synchronization



The accuracy of the synchronization depends on several parameters:

- The nominal clock frequency (f_{Nom}) (the theoretical slave node clock frequency)
- The baud rate
- The oversampling (Over=0 => 16X or Over=1 => 8X)

The following formula is used to compute the deviation of the slave bit rate relative to the master bit rate after synchronization (f_{SLAVE} is the real slave node clock frequency).

$$\text{Baud rate deviation} = \left(100 \times \frac{[\alpha \times 8 \times (2 - \text{Over}) + \beta] \times \text{Baud rate}}{8 \times f_{SLAVE}}\right)\%$$

$$\text{Baud rate deviation} = \left(100 \times \frac{[\alpha \times 8 \times (2 - \text{Over}) + \beta] \times \text{Baud rate}}{8 \times \left(\frac{f_{TOL\_UNSYNCH}}{100}\right) \times f_{Nom}}\right)\%$$

$$-0.5 \le \alpha \le +0.5 \quad -1 < \beta < +1$$

$f_{TOL\_UNSYNCH}$ is the deviation of the real slave node clock from the nominal clock frequency. The LIN Standard imposes that it must not exceed ±15%. The LIN Standard imposes also that for communication between two

nodes, their bit rate must not differ by more than ±2%. This means that the baud rate deviation must not exceed ±1%.

It follows from that, a minimum value for the nominal clock frequency:

$$f_{Nom}(min) = \left(100 \times \frac{[0.5 \times 8 \times (2 - Over) + 1] \times Baudrate}{8 \times \left(\frac{-15}{100} + 1\right) \times 1\%} \right) Hz$$

Examples:

- Baud rate = 20 kbit/s, OVER=0 (Oversampling 16X) => f_{Nom}(min) = 2.64 MHz
- Baud rate = 20 kbit/s, OVER=1 (Oversampling 8X) => f_{Nom}(min) = 1.47 MHz
- Baud rate = 1 kbit/s, OVER=0 (Oversampling 16X) => f_{Nom}(min) = 132 kHz
- Baud rate = 1 kbit/s, OVER=1 (Oversampling 8X) => f_{Nom}(min) = 74 kHz

30.6.9.9 Identifier Parity

A protected identifier consists of two subfields; the identifier and the identifier parity. Bits 0 to 5 are assigned to the identifier and bits 6 and 7 are assigned to the parity.

The USART interface can generate/check these parity bits, but this feature can also be disabled. The user can choose between two modes using US_LINMR.PARDIS:

- PARDIS = 0:
  - During header transmission, the parity bits are computed and sent with the 6 least significant bits of US_LINIR.IDCHR. The bits 6 and 7 of this register are discarded.
  - During header reception, the parity bits of the identifier are checked. If the parity bits are wrong, an Identifier Parity error occurs (see Section 30.6.3.5 ”Parity”). Only the 6 least significant bits of the IDCHR field are updated with the received Identifier. Bits 6 and 7 are stuck to 0.
- PARDIS = 1:
  - During header transmission, all the bits of US_LINIR.IDCHR are sent on the bus.
  - During header reception, all the bits of IDCHR are updated with the received Identifier.

30.6.9.10 Node Action

Depending on the identifier, the node is affected – or not – by the LIN response. Consequently, after sending or receiving the identifier, the USART must be configured. There are three possible configurations:

- PUBLISH: the node sends the response.
- SUBSCRIBE: the node receives the response.
- IGNORE: the node is not concerned by the response, it does not send and does not receive the response.

This configuration is made by the field, Node Action (NACT), in US_LINMR (see Section 30.7.25 ”USART LIN Mode Register”).

Example: a LIN cluster that contains a master and two slaves:

- Data transfer from the master to slave1 and to slave2:

  NACT(master)=PUBLISH

  NACT(slave1)=SUBSCRIBE

  NACT(slave2)=SUBSCRIBE
- Data transfer from the master to slave1 only:

  NACT(master)=PUBLISH

  NACT(slave1)=SUBSCRIBE

NACT(slave2)=IGNORE

- Data transfer from slave1 to the master:

  NACT(master)=SUBSCRIBE

  NACT(slave1)=PUBLISH

  NACT(slave2)=IGNORE

- Data transfer from slave1 to slave2:

  NACT(master)=IGNORE

  NACT(slave1)=PUBLISH

  NACT(slave2)=SUBSCRIBE

- Data transfer from slave2 to the master and to slave1:

  NACT(master)=SUBSCRIBE

  NACT(slave1)=SUBSCRIBE

  NACT(slave2)=PUBLISH

30.6.9.11 Response Data Length

The LIN response data length is the number of data fields (bytes) of the response excluding the checksum.

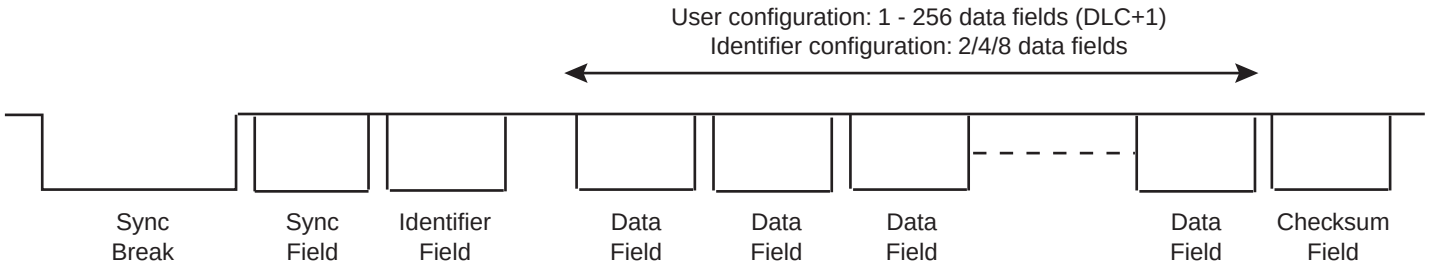
The response data length can either be configured by the user or be defined automatically by bits 4 and 5 of the Identifier (compatibility to LIN Specification 1.1). The user can choose between these two modes using US_LINMR.DLM:

- DLM = 0: The response data length is configured by the user via US_LINMR.DLC. The response data length is equal to (DLC + 1) bytes. DLC can be programmed from 0 to 255, so the response can contain from 1 data byte up to 256 data bytes.
- DLM = 1: The response data length is defined by the Identifier (US_LINIR.IDCHR) according to the table below. US_LINMR.DLC is discarded. The response can contain 2 or 4 or 8 data bytes.

Table 30-15. Response Data Length if DLM = 1

IDCHR[5]	IDCHR[4]	Response Data Length [Bytes]
0	0	2
0	1	2
1	0	4
1	1	8

Figure 30-35. Response Data Length



30.6.9.12 Checksum

The last field of a frame is the checksum. The checksum contains the inverted 8- bit sum with carry, over all data bytes or all data bytes and the protected identifier. Checksum calculation over the data bytes only is called classic checksum and it is used for communication with LIN 1.3 slaves. Checksum calculation over the data bytes and the protected identifier byte is called enhanced checksum and it is used for communication with LIN 2.0 slaves.

The USART can be configured to:

- Send/check an enhanced checksum automatically (CHKDIS = 0 & CHKTYP = 0)
- Send/check a classic checksum automatically (CHKDIS = 0 & CHKTYP = 1)
- Not send/check a checksum (CHKDIS = 1)

This configuration is made by the Checksum Type (CHKTYP) and Checksum Disable (CHKDIS) fields of US_LINMR.

If the checksum feature is disabled, the user can send it manually all the same, by considering the checksum as a normal data byte and by adding 1 to the response data length (see Section 30.6.9.11 "Response Data Length").

30.6.9.13 Frame Slot Mode

This mode is useful for master nodes only. It enforces the following rule: each frame slot must be longer than or equal to $t_{Frame\_Maximum}$.

If the Frame Slot mode is enabled (FSDIS = 0) and a frame transfer has been completed, the TXRDY flag is set again only after $t_{Frame\_Maximum}$ delay, from the start of frame. So the master node cannot send a new header if the frame slot duration of the previous frame is lower than $t_{Frame\_Maximum}$.

If the Frame Slot mode is disabled (FSDIS = 1) and a frame transfer has been completed, the TXRDY flag is set again immediately.

The $t_{Frame\_Maximum}$ is calculated as follows:

If the checksum is sent (CHKDIS = 0):

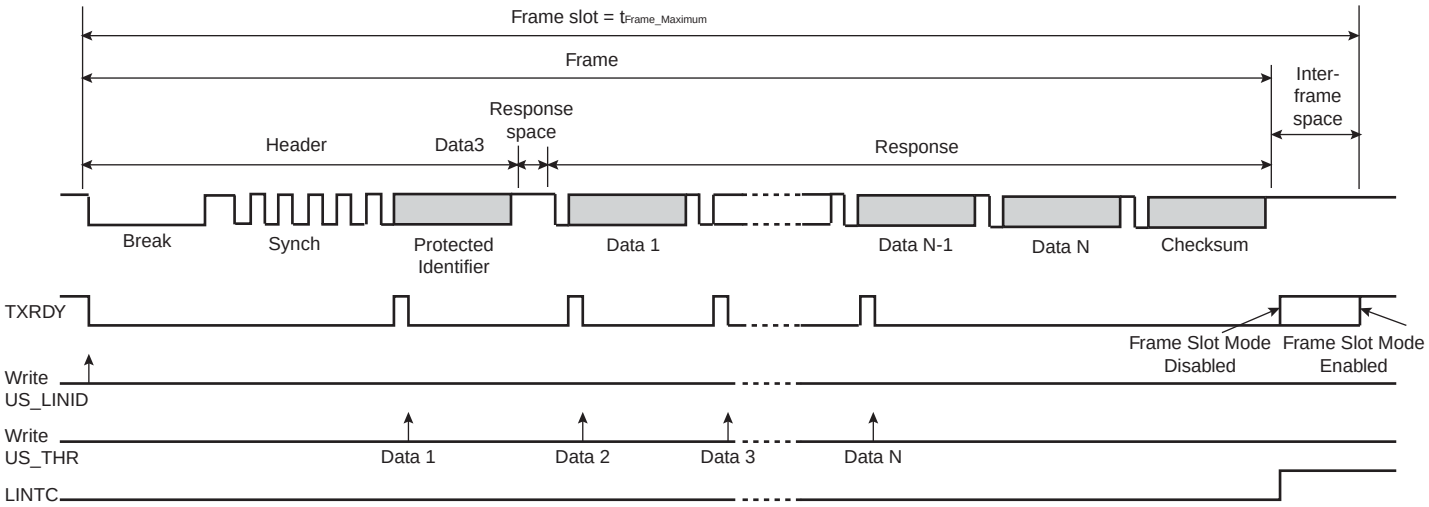
- $t_{Header\_Nominal} = 34 \times t_{bit}$
- $t_{Response\_Nominal} = 10 \times (NData + 1) \times t_{bit}$
- $t_{Frame\_Maximum} = 1.4 \times (t_{Header\_Nominal} + t_{Response\_Nominal} + 1)$(1)
- $t_{Frame\_Maximum} = 1.4 \times (34 + 10 \times (DLC + 1 + 1) + 1) \times t_{bit}$
- $t_{Frame\_Maximum} = (77 + 14 \times DLC) \times t_{bit}$

If the checksum is not sent (CHKDIS = 1):

- $t_{Header\_Nominal} = 34 \times t_{bit}$
- $t_{Response\_Nominal} = 10 \times NData \times t_{bit}$
- $t_{Frame\_Maximum} = 1.4 \times (t_{Header\_Nominal} + t_{Response\_Nominal} + 1)$(1)
- $t_{Frame\_Maximum} = 1.4 \times (34 + 10 \times (DLC + 1) + 1) \times t_{bit}$
- $t_{Frame\_Maximum} = (63 + 14 \times DLC) \times t_{bit}$

Note: 1. The term "+1" leads to an integer result for $t_{Frame\_Maximum}$ (LIN Specification 1.3).

Figure 30-36. Frame Slot Mode



30.6.9.14 LIN Errors

Bit Error

This error is generated in master of slave node configuration, when the USART is transmitting and if the transmitted value on the Tx line is different from the value sampled on the Rx line. If a bit error is detected, the transmission is aborted at the next byte border.

This error is reported by flag US_CSR.LINBE.

Inconsistent Synch Field Error

This error is generated in slave node configuration if the Synch Field character received is other than 0x55.

This error is reported by flag US_CSR.LINISFE.

Identifier Parity Error

This error is generated in slave node configuration, if the parity of the identifier is wrong. This error can be generated only if the parity feature is enabled (PARDIS = 0).

This error is reported by flag US_CSR.LINIPE.

Checksum Error

This error is generated in master of slave node configuration, if the received checksum is wrong. This flag can be set to ‘1’ only if the checksum feature is enabled (CHKDIS = 0).

This error is reported by flag US_CSR.LINCE.

Slave Not Responding Error

This error is generated in master of slave node configuration, when the USART expects a response from another node (NACT = SUBSCRIBE) but no valid message appears on the bus within the time given by the maximum length of the message frame, $t_{Frame\_Maximum}$ (see Section 30.6.9.13 ”Frame Slot Mode”). This error is disabled if the USART does not expect any message (NACT = PUBLISH or NACT = IGNORE).

This error is reported by flag US_CSR.LINSNRE.

Synch Tolerance Error

This error is generated in slave node configuration if, after the clock synchronization procedure, it appears that the computed baud rate deviation compared to the initial baud rate is superior to the maximum tolerance FTol_Unsynch (±15%).

This error is reported by flag US_CSR.LINSTE.

Header Timeout Error

This error is generated in slave node configuration, if the Header is not entirely received within the time given by the maximum length of the Header, $t_{Header\_Maximum}$.

This error is reported by flag US_CSR.LINHTE.

30.6.9.15 LIN Frame Handling

Master Node Configuration

- Write TXEN and RXEN in US_CR to enable both the transmitter and the receiver.
- Write USART_MODE in US_MR to select the LIN mode and the master node configuration.
- Write CD and FP in US_BRGR to configure the baud rate.
- Write NACT, PARDIS, CHKDIS, CHKTYPE, DLCM, FSDIS and DLC in US_LINMR to configure the frame transfer.
- Check that TXRDY in US_CSR is set to ‘1’.
- Write IDCHR in US_LINIR to send the header.

What comes next depends on the NACT configuration:

- Case 1: NACT = PUBLISH, the USART sends the response.
  - Wait until TXRDY in US_CSR rises.
  - Write TCHR in US_THR to send a byte.
  - If all the data have not been written, redo the two previous steps.
  - Wait until LINTC in US_CSR rises.
  - Check the LIN errors.
- Case 2: NACT = SUBSCRIBE, the USART receives the response.
  - Wait until RXRDY in US_CSR rises.
  - Read RCHR in US_RHR.
  - If all the data have not been read, redo the two previous steps.
  - Wait until LINTC in US_CSR rises.
  - Check the LIN errors.
- Case 3: NACT = IGNORE, the USART is not concerned by the response.
  - Wait until LINTC in US_CSR rises.
  - Check the LIN errors.

Figure 30-37. Master Node Configuration, NACT = PUBLISH

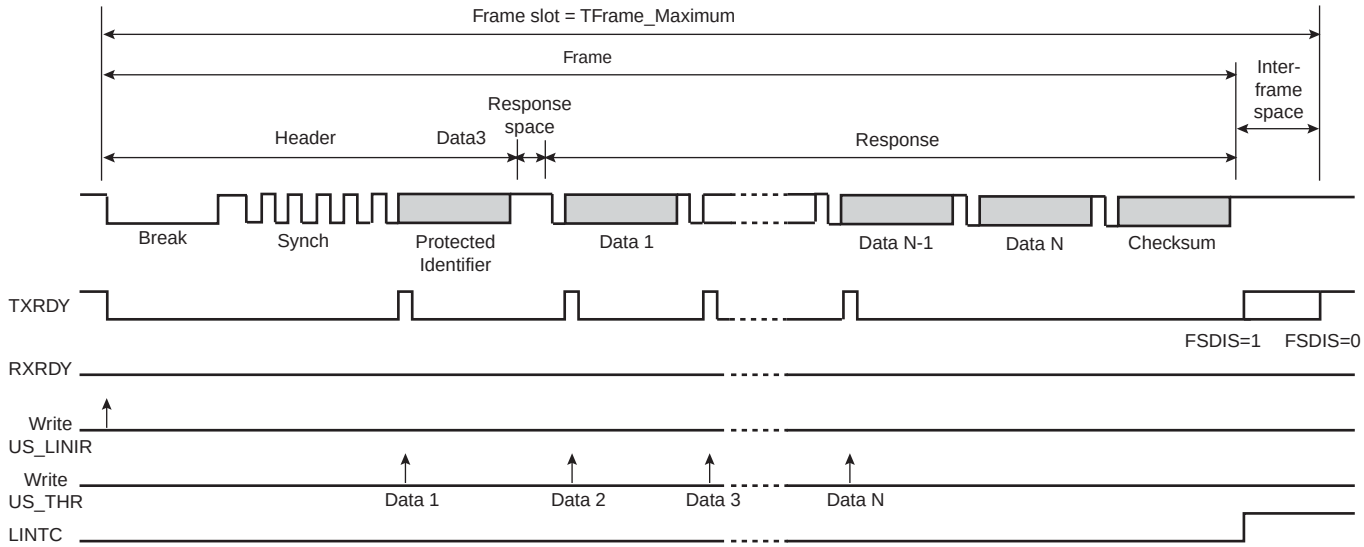


Figure 30-38. Master Node Configuration, NACT = SUBSCRIBE

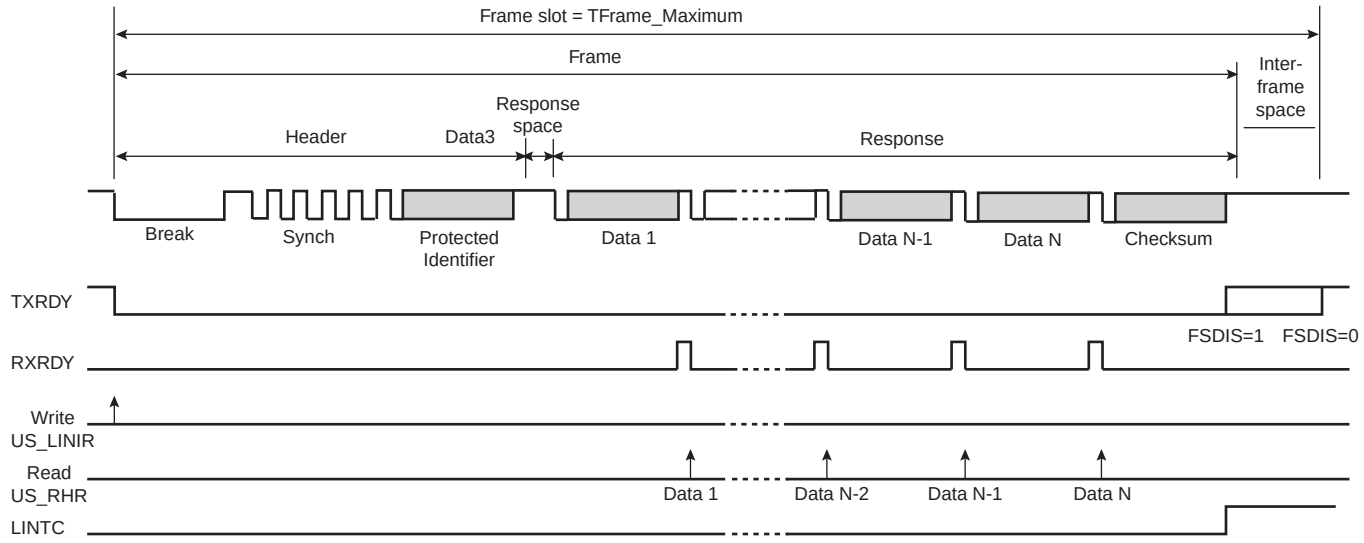
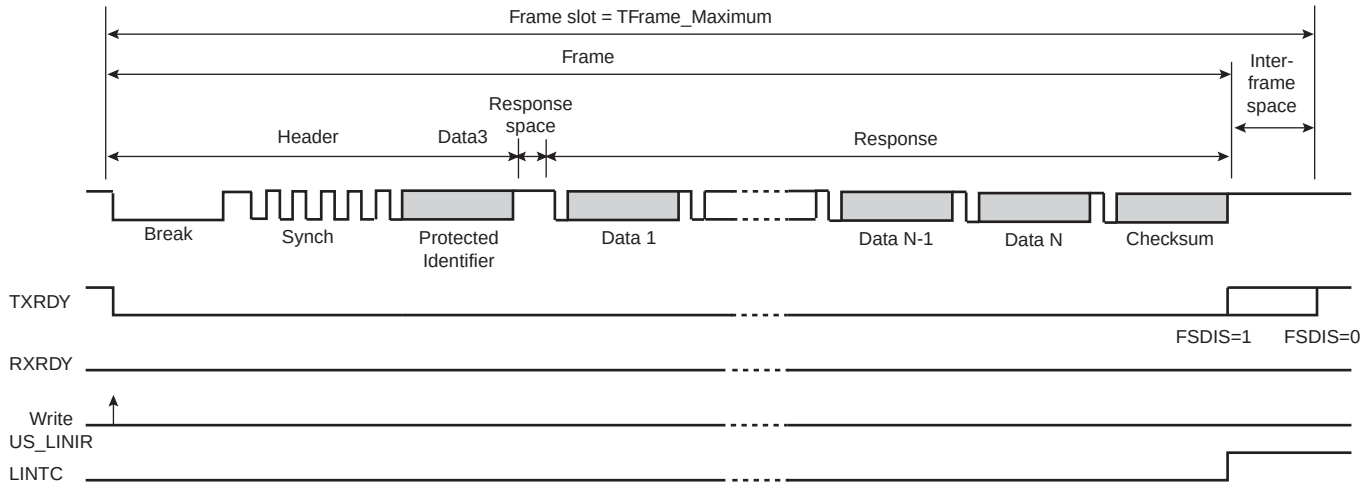


Figure 30-39. Master Node Configuration, NACT = IGNORE



Slave Node Configuration

- Write TXEN and RXEN in US_CR to enable both the transmitter and the receiver.
- Write USART_MODE in US_MR to select the LIN mode and the slave node configuration.
- Write CD and FP in US_BRGR to configure the baud rate.
- Wait until LINID in US_CSR rises.
- Check LINISFE and LINPE errors.
- Read IDCHR in US_RHR.
- Write NACT, PARDIS, CHKDIS, CHKTYPE, DLCM and DLC in US_LINMR to configure the frame transfer.

IMPORTANT: If the NACT configuration for this frame is PUBLISH, the US_LINMR must be written with NACT = PUBLISH even if this field is already correctly configured, in order to set the TXREADY flag and the corresponding write transfer request.

What comes next depends on the NACT configuration:

- Case 1: NACT = PUBLISH, the LIN controller sends the response.
  - Wait until TXRDY in US_CSR rises.
  - Write TCHR in US_THR to send a byte.
  - If all the data have not been written, redo the two previous steps.
  - Wait until LINTC in US_CSR rises.
  - Check the LIN errors.
- Case 2: NACT = SUBSCRIBE, the USART receives the response.
  - Wait until RXRDY in US_CSR rises.
  - Read RCHR in US_RHR.
  - If all the data have not been read, redo the two previous steps.
  - Wait until LINTC in US_CSR rises.
  - Check the LIN errors.
- Case 3: NACT = IGNORE, the USART is not concerned by the response.
  - Wait until LINTC in US_CSR rises.
  - Check the LIN errors.

Figure 30-40. Slave Node Configuration, NACT = PUBLISH

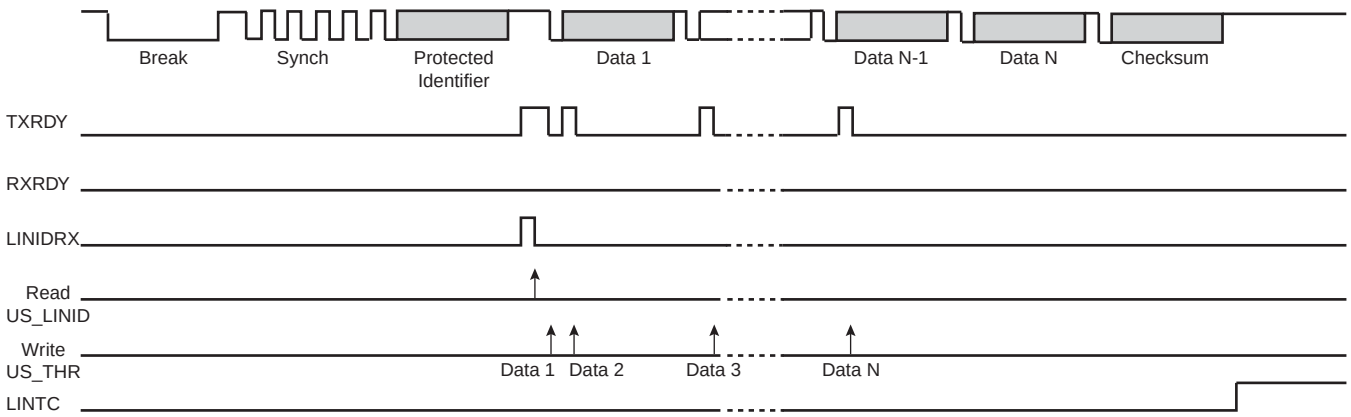


Figure 30-41. Slave Node Configuration, NACT = SUBSCRIBE

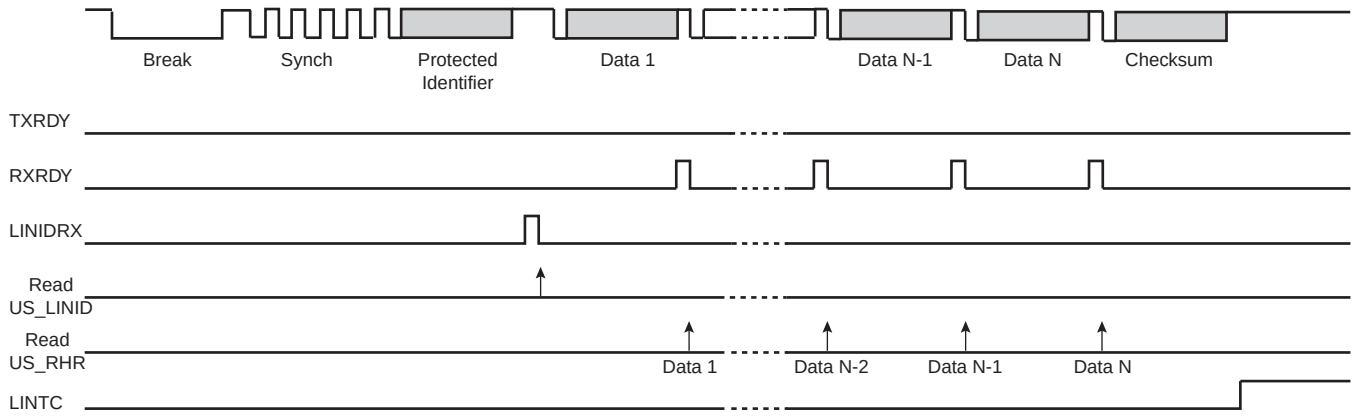
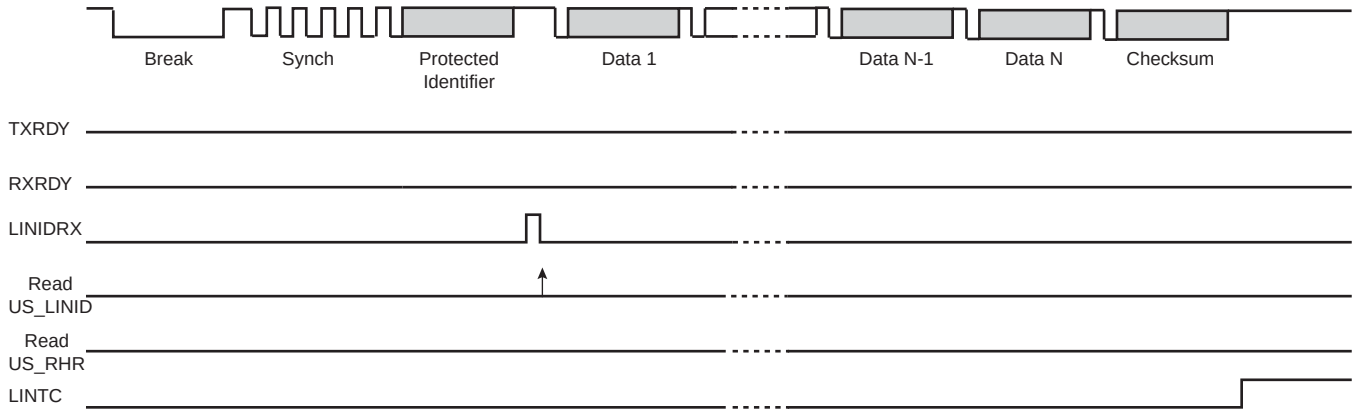


Figure 30-42. Slave Node Configuration, NACT = IGNORE



30.6.9.16 LIN Frame Handling with PDC

The USART can be used in association with the PDC in order to transfer data directly into/from the on- and off-chip memories without any processor intervention.

The PDC uses the trigger flags, TXRDY and RXRDY, to write or read into the USART. The PDC always writes in US_THR and it always reads in US_RHR. The size of the data written or read by the PDC in the USART is always a byte.

Master Node Configuration

The user can choose between two PDC modes by the PDCM bit in US_LINMR:

- PDCM = 1: the LIN configuration is stored in the WRITE buffer and it is written by the PDC in the US_THR (instead of the US_LINMR). Because the PDC transfer size is limited to a byte, the transfer is split into two accesses. During the first access the bits, NACT, PARDIS, CHKDIS, CHKTYP, DLM and FSDIS are written. During the second access, the 8-bit DLC field is written.
- PDCM = 0: the LIN configuration is not stored in the WRITE buffer and it must be written by the user in US_LINMR.

The WRITE buffer also contains the Identifier and the DATA, if the USART sends the response (NACT = PUBLISH).

The READ buffer contains the DATA if the USART receives the response (NACT = SUBSCRIBE).

Figure 30-43. Master Node with PDC (PDCM = 1)

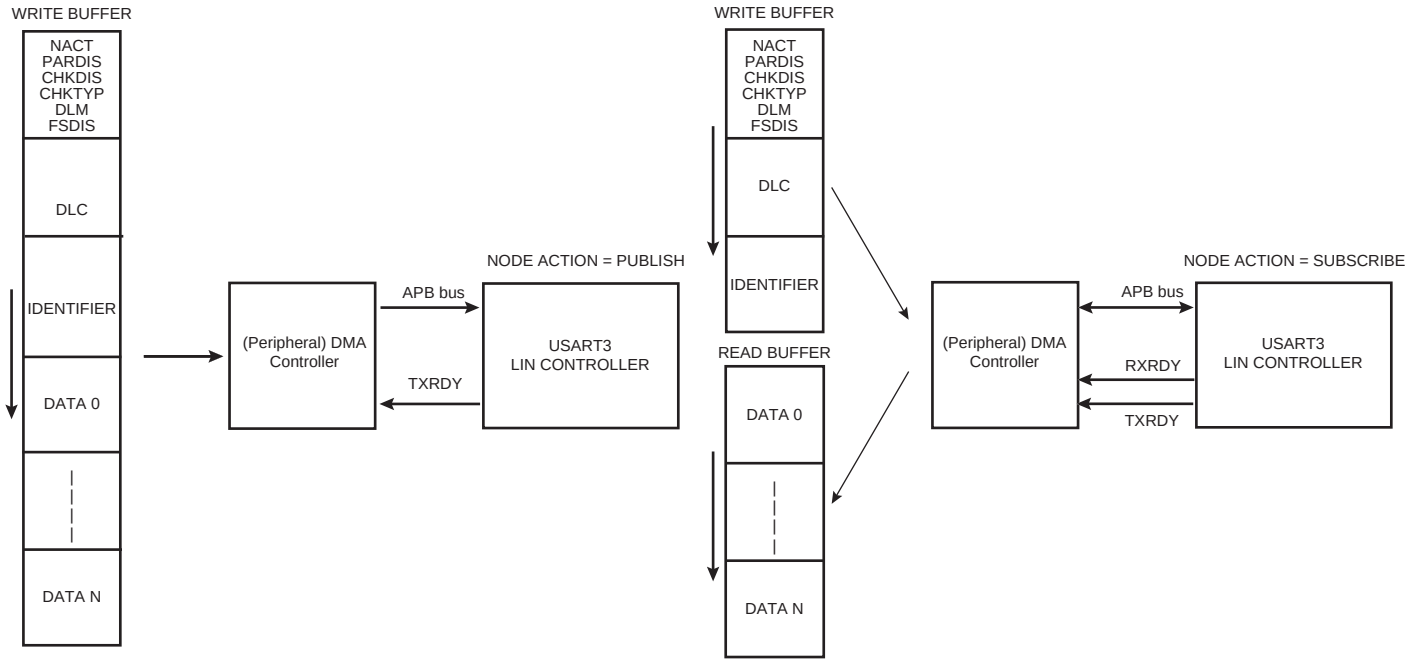
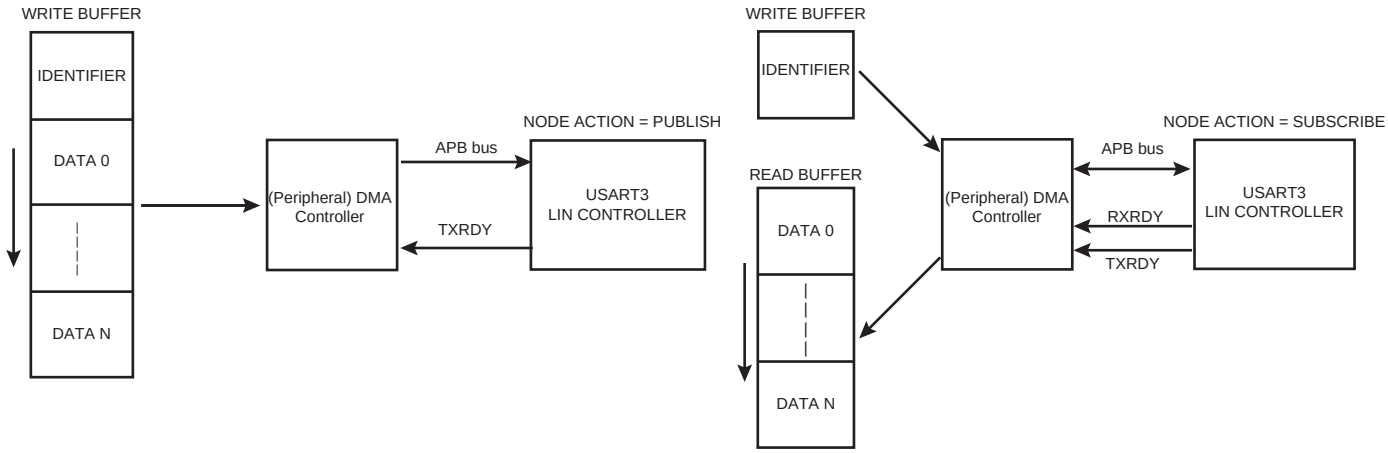


Figure 30-44. Master Node with PDC (PDCM = 0)



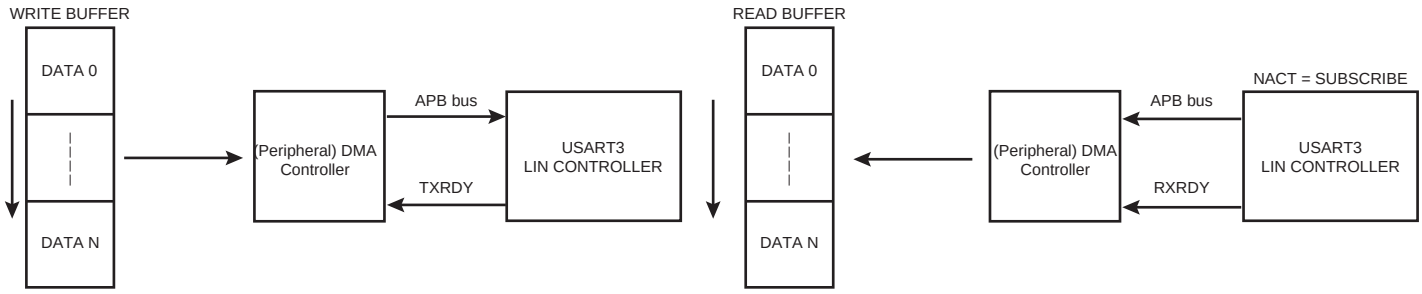
Slave Node Configuration

In this configuration, the PDC transfers only the DATA. The Identifier must be read by the user in the US_LINIR. The LIN mode must be written by the user in US_LINMR.

The WRITE buffer contains the DATA if the USART sends the response (NACT = PUBLISH).

The READ buffer contains the DATA if the USART receives the response (NACT = SUBSCRIBE).

Figure 30-45. Slave Node with PDC



30.6.9.17 Wakeup Request

Any node in a sleeping LIN cluster may request a wakeup.

In the LIN 2.0 specification, the wakeup request is issued by forcing the bus to the dominant state from 250 µs to 5 ms. For this, it is necessary to send the character 0xF0 in order to impose 5 successive dominant bits. Whatever the baud rate is, this character complies with the specified timings.

- Baud rate min = 1 kbit/s -> t_{bit} = 1 ms -> 5 t_{bit} = 5 ms
- Baud rate max = 20 kbit/s -> t_{bit} = 50 µs -> 5 t_{bit} = 250 µs

In the LIN 1.3 specification, the wakeup request should be generated with character 0x80 in order to impose eight successive dominant bits.

Using the WKUPTYP bit in US_LINMR, the user can choose to send either a LIN 2.0 wakeup request (WKUPTYP = 0) or a LIN 1.3 wakeup request (WKUPTYP = 1).

A wakeup request is transmitted by writing US_CR with the LINWKUP bit to ’1’. Once the transfer is completed, the LINTC flag is asserted in the Status register (US_SR). It is cleared by writing the US_CR with the RSTSTA bit to ’1’.

30.6.9.18 Bus Idle Timeout

If the LIN bus is inactive for a certain duration, the slave nodes automatically enter Sleep mode. In the LIN 2.0 specification, this timeout is set to 4 seconds. In the LIN 1.3 specification, it is set to 25,000 t_{bit}.

In slave node configuration, the receiver timeout detects an idle condition on the RXD line. When a timeout is detected, the US_CSR.TIMEOUT rises and can generate an interrupt, thus indicating to the driver to enter Sleep mode.

The timeout delay period (during which the receiver waits for a new character) is programmed in US_RTOR.TO. If a ‘0’ is written to TO, the Receiver Timeout is disabled and no timeout is detected. US_CSR.TIMEOUT remains at ‘0’. Otherwise, the receiver loads a 17-bit counter with the value programmed in TO. This counter is decremented at each bit period and reloaded each time a new character is received. If the counter reaches 0, US_CSR.TIMEOUT rises.

If US_CR.STTTO is written to ‘1’, the counter clock is stopped until a first character is received.

If US_CR.RETTO is written to ‘1’, the counter starts counting down immediately from the value TO.

Table 30-16. Receiver Timeout Programming

LIN Specification	Baud Rate	Timeout period	TO
2.0	1,000 bit/s	4s	4,000
	2,400 bit/s		9,600
	9,600 bit/s		38,400
	19,200 bit/s		76,800
	20,000 bit/s		80,000
1.3	–	25,000 t_{bit}	25,000

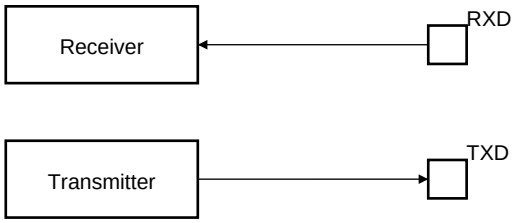
30.6.10 Test Modes

The USART can be programmed to operate in three different test modes. The internal loopback capability enables on-board diagnostics. In Loopback mode, the USART interface pins are disconnected or not, and reconfigured for loopback internally or externally.

30.6.10.1 Normal Mode

Normal mode connects the RXD pin on the receiver input and the transmitter output on the TXD pin.

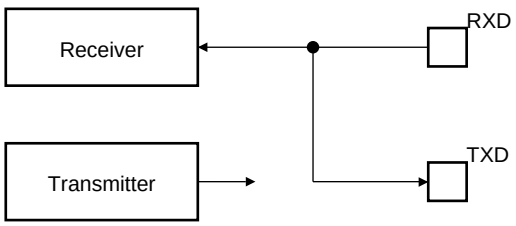
Figure 30-46. Normal Mode Configuration



30.6.10.2 Automatic Echo Mode

Automatic Echo mode is used for bit-by-bit retransmission. When a bit is received on the RXD pin, it is sent to the TXD pin, as shown in Figure 30-47. 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.

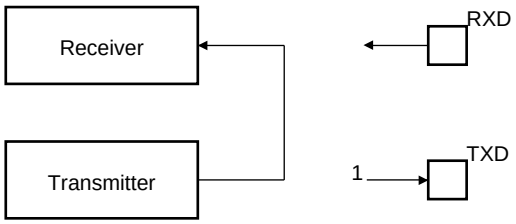
Figure 30-47. Automatic Echo Mode Configuration



30.6.10.3 Local Loopback Mode

Local Loopback mode connects the output of the transmitter directly to the input of the receiver, as shown in Figure 30-48. 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.

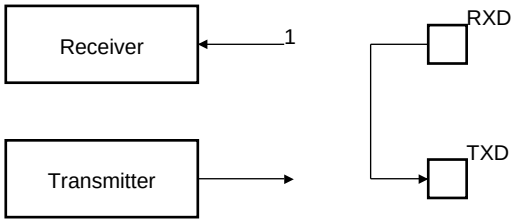
Figure 30-48. Local Loopback Mode Configuration



30.6.10.4 Remote Loopback Mode

Remote Loopback mode directly connects the RXD pin to the TXD pin, as shown in Figure 30-49. The transmitter and the receiver are disabled and have no effect. This mode is used for bit-by-bit retransmission.

Figure 30-49. Remote Loopback Mode Configuration



30.6.11 USART 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 Timeout Register"
- "USART Transmitter Timeguard Register"
- "USART FI DI RATIO Register"
- "USART IrDA FILTER Register"
- "USART Comparison Register"

30.7 Universal Synchronous Asynchronous Receiver Transmitter (USART) User Interface

Table 30-17. Register Mapping

Offset	Register	Name	Access	Reset
0x000	USART Control Register	US_CR	Write-only	–
0x004	USART Mode Register	US_MR	Read/Write	–
0x008	USART Interrupt Enable Register	US_IER	Write-only	–
0x00C	USART Interrupt Disable Register	US_IDR	Write-only	–
0x010	USART Interrupt Mask Register	US_IMR	Read-only	0x0
0x014	USART Channel Status Register	US_CSR	Read-only	–
0x018	USART Receive Holding Register	US_RHR	Read-only	0x0
0x01C	USART Transmit Holding Register	US_THR	Write-only	–
0x020	USART Baud Rate Generator Register	US_BRGR	Read/Write	0x0
0x024	USART Receiver Timeout Register	US_RTOR	Read/Write	0x0
0x028	USART Transmitter Timeguard Register	US_TTGR	Read/Write	0x0
0x02C–0x03C	Reserved	–	–	–
0x040	USART FI DI Ratio Register	US_FIDI	Read/Write	0x174
0x044	USART Number of Errors Register	US_NER	Read-only	–
0x048	Reserved	–	–	–
0x04C	USART IrDA Filter Register	US_IF	Read/Write	0x0
0x050	Reserved	–	–	–
0x054	USART LIN Mode Register	US_LINMR	Read/Write	0x0
0x058	USART LIN Identifier Register	US_LINIR	Read/Write(1)	0x0
0x05C	USART LIN Baud Rate Register	US_LINBRR	Read-only	0x0
0x060–0x088	Reserved	–	–	–
0x090	USART Comparison Register	US_CMPR	Read/Write	0x0
0x094–0x0E0	Reserved	–	–	–
0x0E4	USART Write Protection Mode Register	US_WPMR	Read/Write	0x0
0x0E8	USART Write Protection Status Register	US_WPSR	Read-only	0x0
0x0EC–0x0F8	Reserved	–	–	–

Notes: 1. Write is possible only in LIN master node configuration.

30.7.1 USART Control Register

Name: US_CR

Address: 0x4000C200 (0), 0x40020200 (1), 0x40024200 (2), 0x40018200 (3), 0x4001C200 (4), 0x40008200 (5), 0x40040200 (6), 0x40034200 (7)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	REQCLR	–	–	–	–
23	22	21	20	19	18	17	16
–	–	LINWKUP	LINABT	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)".

- **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: Clears the status bits PARE, FRAME, OVRE, LINBE, LINISFE, LINIPE, LINCE, LINSNRE, LINSTE, LINHTE, LINID, LINTC, LINBK, CMP and RXBRK in US_CSR.

- **STTBRK: Start Break**

0: No effect.

1: Starts transmission of a break after the characters present in the US_THR and the Transmit Shift Register have been transmitted. No effect if a break is already being transmitted.

- **STPBRK: Stop Break**

0: No effect.

1: Stops transmission of the break after a minimum of one character length and transmits a high level during 12-bit periods. No effect if no break is being transmitted.

- **STTTO: Clear TIMEOUT Flag and Start Timeout After Next Character Received**

0: No effect.

1: Starts waiting for a character before enabling the timeout counter. Immediately disables a timeout period in progress.

Clears 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: Clears the bit ITER in US_CSR. No effect if the ISO7816 is not enabled.

- **RSTNACK: Reset Non Acknowledge**

0: No effect

1: Clears NACK in US_CSR.

- **RETTO: Start Timeout Immediately**

0: No effect.

1: Immediately restarts timeout period.

- **RTSEN: Request to Send Enable**

0: No effect.

1: Drives the RTS pin to 1 if US_MR.USART_MODE field = 2, else drives the RTS pin to 0 if US_MR.USART_MODE field = 0.

- **RTSDIS: Request to Send Disable**

0: No effect.

1: Drives the RTS pin to 0 if US_MR.USART_MODE field = 2 (if PDC Receive buffer is not full), else drives the RTS pin to 1 if US_MR.USART_MODE field = 0.

- **LINABT: Abort LIN Transmission**

0: No effect.

1: Abort the current LIN transmission.

- **LINWKUP: Send LIN Wakeup Signal**

0: No effect:

1: Sends a wakeup signal on the LIN bus.

- **REQCLR: Request to Clear the Comparison Trigger**

SleepWalking enabled:

0: No effect.

1: Clears the potential clock request currently issued by the USART, thus the potential system wakeup is cancelled.

SleepWalking disabled:

0: No effect.

1: Restarts the comparison trigger to enable US_RHR loading.

30.7.2 USART Control Register (SPI_MODE)

Name: US_CR (SPI_MODE)

Address: 0x4000C200 (0), 0x40020200 (1), 0x40024200 (2), 0x40018200 (3), 0x4001C200 (4), 0x40008200 (5), 0x40040200 (6), 0x40034200 (7)

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".

- **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).

30.7.3 USART Mode Register

Name: US_MR

Address: 0x4000C204 (0), 0x40020204 (1), 0x40024204 (2), 0x40018204 (3), 0x4001C204 (4), 0x40008204 (5), 0x40040204 (6), 0x40034204 (7)

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".

For SPI configuration, see "USART Mode Register (SPI_MODE)".

- **USART_MODE: USART Mode of Operation**

Value	Name	Description
0x0	NORMAL	Normal mode
0x2	HW_HANDSHAKING	Hardware Handshaking
0x4	IS07816_T_0	IS07816 Protocol: T = 0
0x6	IS07816_T_1	IS07816 Protocol: T = 1
0x8	IRDA	IrDA mode
0xA	LIN_MASTER	LIN Master mode
0xB	LIN_SLAVE	LIN Slave mode
0xE	SPI_MASTER	SPI Master mode (CLKO must be written to 1 and USCLKS = 0, 1 or 2)
0xF	SPI_SLAVE	SPI Slave mode

The PDC transfers are supported in all USART modes of operation.

- **USCLKS: Clock Selection**

Value	Name	Description
0	MCK	Peripheral clock is selected
1	DIV	Peripheral clock Divided (DIV=8) is selected
2	PMC_PCK	PMC programmable clock (PCK) is selected. If the SCK pin is driven (CLKO=1), the CD field must be greater than 1.
3	SCK	External pin (SCK) is selected

- **CHRL: Character Length**

Value	Name	Description
0	5_BIT	Character length is 5 bits
1	6_BIT	Character length is 6 bits
2	7_BIT	Character length is 7 bits
3	8_BIT	Character length is 8 bits

- **SYNC: Synchronous Mode Select**

0: USART operates in Asynchronous mode (UART).

1: USART operates in Synchronous mode.

- **PAR: Parity Type**

Value	Name	Description
0	EVEN	Even parity
1	ODD	Odd parity
2	SPACE	Parity forced to 0 (Space)
3	MARK	Parity forced to 1 (Mark)
4	NO	No parity
6	MULTIDROP	Multidrop mode

- **NBSTOP: Number of Stop Bits**

Value	Name	Description
0	1_BIT	1 stop bit
1	1_5_BIT	1.5 stop bits (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 (Synchronous Slave mode or Asynchronous mode with external baud rate clock source).

1: The USART drives the SCK pin if USCLKS does not select the external clock SCK (USART Synchronous Master Mode).

- **OVER: Oversampling Mode**

0: 16x Oversampling.

1: 8x Oversampling.

- **INACK: Inhibit Non Acknowledge**

0: The NACK is generated.

1: The NACK is not generated.

- **DSNACK: Disable Successive NACK**

0: NACK is sent on the ISO line as soon as a parity error occurs in the received character (unless INACK is set).

1: Successive parity errors are counted up to the value specified in the MAX_ITERATION field. These parity errors generate a NACK on the ISO line. As soon as this value is reached, no additional NACK is sent on the ISO line. The flag ITER is asserted.

Note: The MAX_ITERATION field must be set to 0 if DSNACK is cleared.

- **INVDATA: Inverted Data**

0: The data field transmitted on TXD line is the same as the one written in US_THR or the content read in US_RHR is the same as RXD line. Normal mode of operation.

1: The data field transmitted on TXD line is inverted (voltage polarity only) compared to the value written on US_THR or the content read in US_RHR is inverted compared to what is received on RXD line (or ISO7816 IO line). Inverted mode of operation, useful for contactless card application. To be used with MSBF configuration bit.

- **MAX_ITERATION: Maximum Number of Automatic Iteration**

0–7: Defines the maximum number of iterations in mode ISO7816, protocol T = 0.

- **FILTER: Receive Line Filter**

0: The USART does not filter the receive line.

1: The USART filters the receive line using a three-sample filter (1/16-bit clock) (2 over 3 majority).

30.7.4 USART Mode Register (SPI_MODE)

Name: US_MR (SPI_MODE)

Address: 0x4000C204 (0), 0x40020204 (1), 0x40024204 (2), 0x40018204 (3), 0x4001C204 (4), 0x40008204 (5), 0x40040204 (6), 0x40034204 (7)

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	WRDBT	–	CLKO	–	CPOL
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	CPHA
7	6	5	4	3	2	1	0
CHRL		USCLKS		USART_MODE			

This configuration is relevant only if USART_MODE = 0xE or 0xF in "USART Mode Register".

This register can only be written if the WPEN bit is cleared in "USART Write Protection Mode Register".

- **USART_MODE: USART Mode of Operation**

Value	Name	Description
0xE	SPI_MASTER	SPI Master mode
0xF	SPI_SLAVE	SPI Slave mode

- **USCLKS: Clock Selection**

0	MCK	Peripheral clock is selected
1	DIV	Peripheral clock Divided (DIV=8) is selected
2	PMC_PCK	A PMC programmable clock (PCK) is selected
3	SCK	External pin (SCK) is selected

- **CHRL: Character Length**

Value	Name	Description
3	8_BIT	Character length is 8 bits

- **CPHA: SPI Clock Phase**
  - Applicable if USART operates in SPI mode (USART_MODE = 0xE or 0xF):

0: Data is changed on the leading edge of SPCK and captured on the following edge of SPCK.

1: Data is captured on the leading edge of SPCK and changed on the following edge of SPCK.

CPHA determines which edge of SPCK causes data to change and which edge causes data to be captured. CPHA is used with CPOL to produce the required clock/data relationship between master and slave devices.

- **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.

- **CLKO: Clock Output Select**

0: Mandatory for SPI slave mode, the USART receives the SCK clock.

1: Mandatory for SPI master mode, the USART drives the SCK pin if USCLKS does not select the external clock SCK.

- **WRDBT: Wait Read Data Before Transfer**

0: The character transmission starts as soon as a character is written into US_THR register (assuming TXRDY was set).

1: The character transmission starts when a character is written and only if RXRDY flag is cleared (US_RHR has been read).

30.7.5 USART Interrupt Enable Register

Name: US_IER

Address: 0x4000C208 (0), 0x40020208 (1), 0x40024208 (2), 0x40018208 (3), 0x4001C208 (4), 0x40008208 (5), 0x40040208 (6), 0x40034208 (7)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	CMP	–	–	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)".

For LIN specific configuration, see "USART Interrupt Enable Register (LIN_MODE)".

The following configuration values are valid for all listed bit names of this register:

0: No effect

1: Enables the corresponding interrupt.

- **RXRDY: RXRDY Interrupt Enable**
- **TXRDY: TXRDY Interrupt Enable**
- **RXBRK: Receiver Break Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable (available in all USART modes of operation)**
- **ENDTX: End of Transmit Buffer Interrupt Enable (available in all USART modes of operation)**
- **OVRE: Overrun Error Interrupt Enable**
- **FRAME: Framing Error Interrupt Enable**
- **PARE: Parity Error Interrupt Enable**
- **TIMEOUT: Timeout Interrupt Enable**
- **TXEMPTY: TXEMPTY Interrupt Enable**
- **ITER: Max number of Repetitions Reached Interrupt Enable**
- **TXBUFE: Transmit Buffer Empty Interrupt Enable (available in all USART modes of operation)**
- **RXBUFF: Receive Buffer Full Interrupt Enable (available in all USART modes of operation)**
- **NACK: Non Acknowledge Interrupt Enable**

- **CTSIC: Clear to Send Input Change Interrupt Enable**
- **CMP: Comparison Interrupt Enable**

30.7.6 USART Interrupt Enable Register (SPI_MODE)

Name: US_IER (SPI_MODE)

Address: 0x4000C208 (0), 0x40020208 (1), 0x40024208 (2), 0x40018208 (3), 0x4001C208 (4), 0x40008208 (5), 0x40040208 (6), 0x40034208 (7)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	CMP	–	–	NSSE	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	UNRE	TXEMPTY	–
7	6	5	4	3	2	1	0
–	–	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

This configuration is relevant only if USART_MODE = 0xE or 0xF in "USART Mode Register".

The following configuration values are valid for all listed bit names of this register:

0: No effect

1: Enables the corresponding interrupt.

- **RXRDY: RXRDY Interrupt Enable**
- **TXRDY: TXRDY Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable**
- **ENDTX: End of Transmit Buffer Interrupt Enable**
- **OVRE: Overrun Error Interrupt Enable**
- **TXEMPTY: TXEMPTY Interrupt Enable**
- **UNRE: SPI Underrun Error Interrupt Enable**
- **TXBUFE: Transmit Buffer Empty Interrupt Enable**
- **RXBUFF: Receive Buffer Full Interrupt Enable**
- **NSSE: NSS Line Event (CTS pin) Interrupt Enable**
- **CMP: Comparison Interrupt Enable**

30.7.7 USART Interrupt Enable Register (LIN_MODE)

Name: US_IER (LIN_MODE)

Address: 0x4000C208 (0), 0x40020208 (1), 0x40024208 (2), 0x40018208 (3), 0x4001C208 (4), 0x40008208 (5), 0x40040208 (6), 0x40034208 (7)

Access: Write-only

31	30	29	28	27	26	25	24
LINHTE	LINSTE	LINSNRE	LINCE	LINIPE	LINISFE	LINBE	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
LINTC	LINID	LINBK	RXBUFF	TXBUFE	–	TXEMPTY	TIMEOUT
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

This configuration is relevant only if USART_MODE = 0xA or 0xB in "USART Mode Register".

The following configuration values are valid for all listed bit names of this register:

0: No effect

1: Enables the corresponding interrupt.

- **RXRDY: RXRDY Interrupt Enable**
- **TXRDY: TXRDY Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable**
- **ENDTX: End of Transmit Buffer Interrupt Enable**
- **OVRE: Overrun Error Interrupt Enable**
- **FRAME: Framing Error Interrupt Enable**
- **PARE: Parity Error Interrupt Enable**
- **TIMEOUT: Timeout Interrupt Enable**
- **TXEMPTY: TXEMPTY Interrupt Enable**
- **TXBUFE: Transmit Buffer Empty Interrupt Enable**
- **RXBUFF: Receive Buffer Full Interrupt Enable**
- **LINBK: LIN Break Sent or LIN Break Received Interrupt Enable**
- **LINID: LIN Identifier Sent or LIN Identifier Received Interrupt Enable**
- **LINTC: LIN Transfer Completed Interrupt Enable**
- **LINBE: LIN Bus Error Interrupt Enable**

- **LINISFE: LIN Inconsistent Synch Field Error Interrupt Enable**
- **LINIPE: LIN Identifier Parity Interrupt Enable**
- **LINCE: LIN Checksum Error Interrupt Enable**
- **LINSNRE: LIN Slave Not Responding Error Interrupt Enable**
- **LINSTE: LIN Synch Tolerance Error Interrupt Enable**
- **LINHTE: LIN Header Timeout Error Interrupt Enable**

30.7.8 USART Interrupt Disable Register

Name: US_IDR

Address: 0x4000C20C (0), 0x4002020C (1), 0x4002420C (2), 0x4001820C (3), 0x4001C20C (4), 0x4000820C (5), 0x4004020C (6), 0x4003420C (7)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	CMP	–	–	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)".

For LIN specific configuration, see "USART Interrupt Disable Register (LIN_MODE)".

The following configuration values are valid for all listed bit names of this register:

0: No effect

1: Disables the corresponding interrupt.

- **RXRDY: RXRDY Interrupt Disable**
- **TXRDY: TXRDY Interrupt Disable**
- **RXBRK: Receiver Break Interrupt Disable**
- **ENDRX: End of Receive Buffer Interrupt Enable (available in all USART modes of operation)**
- **ENDTX: End of Transmit Buffer Interrupt Disable (available in all USART modes of operation)**
- **OVRE: Overrun Error Interrupt Enable**
- **FRAME: Framing Error Interrupt Disable**
- **PARE: Parity Error Interrupt Disable**
- **TIMEOUT: Timeout Interrupt Disable**
- **TXEMPTY: TXEMPTY Interrupt Disable**
- **ITER: Max Number of Repetitions Reached Interrupt Disable**
- **TXBUFE: Transmit Buffer Empty Interrupt Enable (available in all USART modes of operation)**
- **RXBUFF: Receive Buffer Full Interrupt Enable (available in all USART modes of operation)**
- **NACK: Non Acknowledge Interrupt Disable**

- **CTSIC: Clear to Send Input Change Interrupt Disable**
- **CMP: Comparison Interrupt Disable**

30.7.9 USART Interrupt Disable Register (SPI_MODE)

Name: US_IDR (SPI_MODE)

Address: 0x4000C20C (0), 0x4002020C (1), 0x4002420C (2), 0x4001820C (3), 0x4001C20C (4), 0x4000820C (5), 0x4004020C (6), 0x4003420C (7)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	CMP	–	–	NSSE	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	UNRE	TXEMPTY	–
7	6	5	4	3	2	1	0
–	–	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

This configuration is relevant only if USART_MODE = 0xE or 0xF in "USART Mode Register".

The following configuration values are valid for all listed bit names of this register:

0: No effect

1: Disables the corresponding interrupt.

- **RXRDY: RXRDY Interrupt Disable**
- **TXRDY: TXRDY Interrupt Disable**
- **ENDRX: End of Receive Buffer Interrupt Disable**
- **ENDTX: End of Transmit Buffer Interrupt Disable**
- **OVRE: Overrun Error Interrupt Disable**
- **TXEMPTY: TXEMPTY Interrupt Disable**
- **UNRE: SPI Underrun Error Interrupt Disable**
- **TXBUFE: Transmit Buffer Empty Interrupt Disable**
- **RXBUFF: Receive Buffer Full Interrupt Disable**
- **NSSE: NSS Line Event (CTS pin) Interrupt Disable**
- **CMP: Comparison Interrupt Disable**

30.7.10 USART Interrupt Disable Register (LIN_MODE)

Name: US_IDR (LIN_MODE)

Address: 0x4000C20C (0), 0x4002020C (1), 0x4002420C (2), 0x4001820C (3), 0x4001C20C (4), 0x4000820C (5), 0x4004020C (6), 0x4003420C (7)

Access: Write-only

31	30	29	28	27	26	25	24
LINHTE	LINSTE	LINSNRE	LINCE	LINIPE	LINISFE	LINBE	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
LINTC	LINID	LINBK	RXBUFF	TXBUFE	–	TXEMPTY	TIMEOUT
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

This configuration is relevant only if USART_MODE = 0xA or 0xB in "USART Mode Register".

The following configuration values are valid for all listed bit names of this register:

0: No effect

1: Disables the corresponding interrupt.

- **RXRDY: RXRDY Interrupt Disable**
- **TXRDY: TXRDY Interrupt Disable**
- **ENDRX: End of Receive Buffer Interrupt Disable**
- **ENDTX: End of Transmit Buffer Interrupt Disable**
- **OVRE: Overrun Error Interrupt Disable**
- **FRAME: Framing Error Interrupt Disable**
- **PARE: Parity Error Interrupt Disable**
- **TIMEOUT: Timeout Interrupt Disable**
- **TXEMPTY: TXEMPTY Interrupt Disable**
- **TXBUFE: Transmit Buffer Empty Interrupt Disable**
- **RXBUFF: Receive Buffer Full Interrupt Disable**
- **LINBK: LIN Break Sent or LIN Break Received Interrupt Disable**
- **LINID: LIN Identifier Sent or LIN Identifier Received Interrupt Disable**
- **LINTC: LIN Transfer Completed Interrupt Disable**
- **LINBE: LIN Bus Error Interrupt Disable**

- **LINISFE: LIN Inconsistent Synch Field Error Interrupt Disable**
- **LINIPE: LIN Identifier Parity Interrupt Disable**
- **LINCE: LIN Checksum Error Interrupt Disable**
- **LINSNRE: LIN Slave Not Responding Error Interrupt Disable**
- **LINSTE: LIN Synch Tolerance Error Interrupt Disable**
- **LINHTE: LIN Header Timeout Error Interrupt Disable**

30.7.11 USART Interrupt Mask Register

Name: US_IMR

Address: 0x4000C210 (0), 0x40020210 (1), 0x40024210 (2), 0x40018210 (3), 0x4001C210 (4), 0x40008210 (5), 0x40040210 (6), 0x40034210 (7)

Access: Read-only

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

23	22	21	20	19	18	17	16
–	CMP	–	–	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)".

For LIN specific configuration, see "USART Interrupt Mask Register (LIN_MODE)".

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

- **RXRDY: RXRDY Interrupt Mask**
- **TXRDY: TXRDY Interrupt Mask**
- **RXBRK: Receiver Break Interrupt Mask**
- **ENDRX: End of Receive Buffer Interrupt Mask (available in all USART modes of operation)**
- **ENDTX: End of Transmit Buffer Interrupt Mask (available in all USART modes of operation)**
- **OVRE: Overrun Error Interrupt Mask**
- **FRAME: Framing Error Interrupt Mask**
- **PARE: Parity Error Interrupt Mask**
- **TIMEOUT: Timeout Interrupt Mask**
- **TXEMPTY: TXEMPTY Interrupt Mask**
- **ITER: Max Number of Repetitions Reached Interrupt Mask**
- **TXBUFE: Transmit Buffer Empty Interrupt Mask (available in all USART modes of operation)**
- **RXBUFF: Receive Buffer Full Interrupt Mask (available in all USART modes of operation)**
- **NACK: Non Acknowledge Interrupt Mask**

- **CTSIC: Clear to Send Input Change Interrupt Mask**
- **CMP: Comparison Interrupt Mask**

30.7.12 USART Interrupt Mask Register (SPI_MODE)

Name: US_IMR (SPI_MODE)

Address: 0x4000C210 (0), 0x40020210 (1), 0x40024210 (2), 0x40018210 (3), 0x4001C210 (4), 0x40008210 (5), 0x40040210 (6), 0x40034210 (7)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	CMP	–	–	NSSE	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	UNRE	TXEMPTY	–
7	6	5	4	3	2	1	0
–	–	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

This configuration is relevant only if USART_MODE = 0xE or 0xF in "USART Mode Register".

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

- **RXRDY: RXRDY Interrupt Mask**
- **TXRDY: TXRDY Interrupt Mask**
- **ENDRX: End of Receive Buffer Interrupt Mask**
- **ENDTX: End of Transmit Buffer Interrupt Mask**
- **OVRE: Overrun Error Interrupt Mask**
- **TXEMPTY: TXEMPTY Interrupt Mask**
- **UNRE: SPI Underrun Error Interrupt Mask**
- **TXBUFE: Transmit Buffer Empty Interrupt Mask**
- **RXBUFF: Receive Buffer Full Interrupt Mask**
- **NSSE: NSS Line Event (CTS pin) Interrupt Mask**
- **CMP: Comparison Interrupt Mask**

30.7.13 USART Interrupt Mask Register (LIN_MODE)

Name: US_IMR (LIN_MODE)

Address: 0x4000C210 (0), 0x40020210 (1), 0x40024210 (2), 0x40018210 (3), 0x4001C210 (4), 0x40008210 (5), 0x40040210 (6), 0x40034210 (7)

Access: Read-only

31	30	29	28	27	26	25	24
LINHTE	LINSTE	LINSNRE	LINCE	LINIPE	LINISFE	LINBE	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
LINTC	LINID	LINBK	RXBUFF	TXBUFE	–	TXEMPTY	TIMEOUT
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

This configuration is relevant only if USART_MODE = 0xA or 0xB in "USART Mode Register".

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

- **RXRDY: RXRDY Interrupt Mask**
- **TXRDY: TXRDY Interrupt Mask**
- **ENDRX: End of Receive Buffer Interrupt Mask**
- **ENDTX: End of Transmit Buffer Interrupt Mask**
- **OVRE: Overrun Error Interrupt Mask**
- **FRAME: Framing Error Interrupt Mask**
- **PARE: Parity Error Interrupt Mask**
- **TIMEOUT: Timeout Interrupt Mask**
- **TXEMPTY: TXEMPTY Interrupt Mask**
- **TXBUFE: Transmit Buffer Empty Interrupt Mask**
- **RXBUFF: Receive Buffer Full Interrupt Mask**
- **LINBK: LIN Break Sent or LIN Break Received Interrupt Mask**
- **LINID: LIN Identifier Sent or LIN Identifier Received Interrupt Mask**
- **LINTC: LIN Transfer Completed Interrupt Mask**
- **LINBE: LIN Bus Error Interrupt Mask**

- **LINISFE: LIN Inconsistent Synch Field Error Interrupt Mask**
- **LINIPE: LIN Identifier Parity Interrupt Mask**
- **LINCE: LIN Checksum Error Interrupt Mask**
- **LINSNRE: LIN Slave Not Responding Error Interrupt Mask**
- **LINSTE: LIN Synch Tolerance Error Interrupt Mask**
- **LINHTE: LIN Header Timeout Error Interrupt Mask**

30.7.14 USART Channel Status Register

Name: US_CSR

Address: 0x4000C214 (0), 0x40020214 (1), 0x40024214 (2), 0x40018214 (3), 0x4001C214 (4), 0x40008214 (5), 0x40040214 (6), 0x40034214 (7)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
CTS	CMP	–	–	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)".

For LIN specific configuration, see "USART Channel Status Register (LIN_MODE)".

- **RXRDY: Receiver Ready (cleared by reading US_RHR)**

0: No complete character has been received since the last read of US_RHR or the receiver is disabled. If characters were being received when the receiver was disabled, RXRDY changes to 1 when the receiver is enabled.

1: At least one complete character has been received and US_RHR has not yet been read.

- **TXRDY: Transmitter Ready (cleared by writing US_THR)**

0: A character is in the US_THR waiting to be transferred to the Transmit Shift Register, or an STTBRK command has been requested, or the transmitter is disabled. As soon as the transmitter is enabled, TXRDY becomes 1.

1: There is no character in the US_THR.

- **RXBRK: Break Received/End of Break (cleared by writing a one to the bit US_CR.RSTSTA)**

0: No break received or end of break detected since the last RSTSTA.

1: Break received or end of break detected since the last RSTSTA.

- **ENDRX: End of RX Buffer (cleared by writing US_RCR or US_RNCR)**

0: The Receive Counter Register has not reached 0 since the last write in US_RCR or US_RNCR(1).

1: The Receive Counter Register has reached 0 since the last write in US_RCR or US_RNCR(1).

- **ENDTX: End of TX Buffer (cleared by writing US_TCR or US_TNCR)**

0: The Transmit Counter Register has not reached 0 since the last write in US_TCR or US_TNCR(1).

1: The Transmit Counter Register has reached 0 since the last write in US_TCR or US_TNCR(1).

- **OVRE: Overrun Error (cleared by writing a one to the bit US_CR.RSTSTA)**

0: No overrun error has occurred since the last RSTSTA.

1: At least one overrun error has occurred since the last RSTSTA.

- **FRAME: Framing Error (cleared by writing a one to the bit US_CR.RSTSTA)**

0: No stop bit has been detected low since the last RSTSTA.

1: At least one stop bit has been detected low since the last RSTSTA.

- **PARE: Parity Error (cleared by writing a one to the US_CR.RSTSTA)**

0: No parity error has been detected since the last RSTSTA.

1: At least one parity error has been detected since the last RSTSTA.

- **TIMEOUT: Receiver Timeout (cleared by writing a one to the bit US_CR.STTTO)**

0: There has not been a timeout since the last Start Timeout command (STTTO in US_CR) or the Timeout Register is 0.

1: There has been a timeout since the last Start Timeout command (STTTO in US_CR).

- **TXEMPTY: Transmitter Empty (cleared by writing US_THR)**

0: There are characters in either US_THR or the Transmit Shift Register, or the transmitter is disabled.

1: There are no characters in US_THR, nor in the Transmit Shift Register.

- **ITER: Max Number of Repetitions Reached (cleared by writing a one to the bit US_CR.RSTIT)**

0: Maximum number of repetitions has not been reached since the last RSTIT.

1: Maximum number of repetitions has been reached since the last RSTIT.

- **TXBUFE: TX Buffer Empty (cleared by writing US_TCR or US_TNCR)**

0: US_TCR or US_TNCR have a value other than 0[(1)].

1: Both US_TCR and US_TNCR have a value of 0[(1)].

- **RXBUFF: RX Buffer Full (cleared by writing US_RCR or US_RNCR)**

0: US_RCR or US_RNCR have a value other than 0[(1)].

1: Both US_RCR and US_RNCR have a value of 0[(1)].

Note: 1. US_RCR, US_RNCR, US_TCR and US_TNCR are PDC registers.

- **NACK: Non Acknowledge Interrupt (cleared by writing a one to the bit US_CR.RSTNACK)**

0: Non acknowledge has not been detected since the last RSTNACK.

1: At least one non acknowledge has been detected since the last RSTNACK.

- **CTSIC: Clear to Send Input Change Flag (cleared on read)**

0: No input change has been detected on the CTS pin since the last read of US_CSR.

1: At least one input change has been detected on the CTS pin since the last read of US_CSR.

- **CMP: Comparison Status (cleared by writing a one to the bit US_CR.RSTSTA command)**

0: No received character matched the comparison criteria programmed in VAL1, VAL2 fields and CMPPAR bit in since the last RSTSTA.

1: A received character matched the comparison criteria since the last RSTSTA.

- **CTS: Image of CTS Input**

0: CTS input is driven low.

1: CTS input is driven high.

30.7.15 USART Channel Status Register (SPI_MODE)

Name: US_CSR (SPI_MODE)

Address: 0x4000C214 (0), 0x40020214 (1), 0x40024214 (2), 0x40018214 (3), 0x4001C214 (4), 0x40008214 (5), 0x40040214 (6), 0x40034214 (7)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
NSS	CMP	–	–	NSSE	–	–	–
15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	UNRE	TXEMPTY	–
7	6	5	4	3	2	1	0
–	–	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

This configuration is relevant only if USART_MODE = 0xE or 0xF in "USART Mode Register".

- **RXRDY: Receiver Ready (cleared by reading US_RHR)**

0: No complete character has been received since the last read of US_RHR or the receiver is disabled. If characters were being received when the receiver was disabled, RXRDY changes to 1 when the receiver is enabled.

1: At least one complete character has been received and US_RHR has not yet been read.

- **TXRDY: Transmitter Ready (cleared by writing US_THR)**

0: A character is in the US_THR waiting to be transferred to the Transmit Shift Register or the transmitter is disabled. As soon as the transmitter is enabled, TXRDY becomes 1.

1: There is no character in the US_THR.

- **ENDRX: End of RX Buffer (cleared by writing US_RCR or US_RNCR)**

0: The Receive Counter Register has not reached 0 since the last write in US_RCR or US_RNCR(1).

1: The Receive Counter Register has reached 0 since the last write in US_RCR or US_RNCR(1).

- **ENDTX: End of TX Buffer (cleared by writing US_TCR or US_TNCR)**

0: The Transmit Counter Register has not reached 0 since the last write in US_TCR or US_TNCR(1).

1: The Transmit Counter Register has reached 0 since the last write in US_TCR or US_TNCR(1).

- **OVRE: Overrun Error (cleared by writing a one to the bit US_CR.RSTSTA)**

0: No overrun error has occurred since the last RSTSTA.

1: At least one overrun error has occurred since the last RSTSTA.

- **TXEMPTY: Transmitter Empty (cleared by writing US_THR)**

0: There are characters in either US_THR or the Transmit Shift Register, or the transmitter is disabled.

1: There are no characters in US_THR, nor in the Transmit Shift Register.

- **UNRE: Underrun Error (cleared by writing a one to the bit US_CR.RSTSTA)**

0: No SPI underrun error has occurred since the last RSTSTA.

1: At least one SPI underrun error has occurred since the last RSTSTA.

- **TXBUFE: TX Buffer Empty (cleared by writing US_TCR or US_TNCR)**

0: US_TCR or US_TNCR have a value other than 0[(1)].

1: Both US_TCR and US_TNCR have a value of 0[(1)].

- **RXBUFF: RX Buffer Full (cleared by writing US_RCR or US_RNCR)**

0: US_RCR or US_RNCR have a value other than 0[(1)].

1: Both US_RCR and US_RNCR have a value of 0[(1)].

Note: 1. US_RCR, US_RNCR, US_TCR and US_TNCR are PDC registers.

- **NSSE: NSS line event (CTS pin) (cleared on read)**

0: No NSS line event has been detected on the CTS pin since the last read of US_CSR.

1: A rising or falling edge has been detected on the NSS line since the last read of US_CSR.

- **CMP: Comparison Match (cleared by writing a one to the bit US_CR.RSTSTA)**

0: No received character matched the comparison criteria programmed in VAL1,VAL2 fields and CMPPAR bit in US_CMPR since the last RSTSTA.

1: A received character matched the comparison criteria since the last RSTSTA.

- **NSS: Image of NSS line event (CTS pin)**

This bit, associated with the NSSE bit, determines if the NSS line event is a rising or a falling edge.

0: NSS line is driven low.

1: NSS line is driven high.

30.7.16 USART Channel Status Register (LIN_MODE)

Name: US_CSR (LIN_MODE)

Address: 0x4000C214 (0), 0x40020214 (1), 0x40024214 (2), 0x40018214 (3), 0x4001C214 (4), 0x40008214 (5), 0x40040214 (6), 0x40034214 (7)

Access: Read-only

31	30	29	28	27	26	25	24
LINHTE	LINSTE	LINSNRE	LINCE	LINIPE	LINISFE	LINBE	–
23	22	21	20	19	18	17	16
LINBLS	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
LINTC	LINID	LINBK	RXBUFF	TXBUFE	–	TXEMPTY	TIMEOUT
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

This configuration is relevant only if USART_MODE = 0xA or 0xB in "USART Mode Register".

- **RXRDY: Receiver Ready (cleared by reading US_RHR)**

0: No complete character has been received since the last read of US_RHR or the receiver is disabled. If characters were being received when the receiver was disabled, RXRDY changes to 1 when the receiver is enabled.

1: At least one complete character has been received and US_RHR has not yet been read.

- **TXRDY: Transmitter Ready (cleared by writing US_THR)**

0: A character is in the US_THR waiting to be transferred to the Transmit Shift Register or the transmitter is disabled. As soon as the transmitter is enabled, TXRDY becomes 1.

1: There is no character in the US_THR.

- **ENDRX: End of RX Buffer (cleared by writing US_RCR or US_RNCR)**

0: The Receive Counter Register has not reached 0 since the last write in US_RCR or US_RNCR(1).

1: The Receive Counter Register has reached 0 since the last write in US_RCR or US_RNCR(1).

- **ENDTX: End of TX Buffer (cleared by writing US_TCR or US_TNCR)**

0: The Transmit Counter Register has not reached 0 since the last write in US_TCR or US_TNCR(1).

1: The Transmit Counter Register has reached 0 since the last write in US_TCR or US_TNCR(1).

- **OVRE: Overrun Error (cleared by writing a one to the bit US_CR.RSTSTA)**

0: No overrun error has occurred since the last RSTSTA.

1: At least one overrun error has occurred since the last RSTSTA.

- **FRAME: Framing Error (cleared by writing a one to the bit US_CR.RSTSTA)**

0: No stop bit has been detected low since the last RSTSTA.

1: At least one stop bit has been detected low since the last RSTSTA.

- **PARE: Parity Error (cleared by writing a one to the bit US_CR.RSTSTA)**

0: No parity error has been detected since the last RSTSTA.

1: At least one parity error has been detected since the last RSTSTA.

- **TIMEOUT: Receiver Timeout (cleared by writing a one to the bit US_CR.RSTSTA)**

0: There has not been a timeout since the last start timeout command (STTTO in US_CR) or the Timeout Register is 0.

1: There has been a timeout since the last start timeout command.

- **TXEMPTY: Transmitter Empty (cleared by writing US_THR)**

0: There are characters in either US_THR or the Transmit Shift Register, or the transmitter is disabled.

1: There are no characters in US_THR, nor in the Transmit Shift Register.

- **TXBUFE: TX Buffer Empty (cleared by writing US_TCR or US_TNCR)**

0: US_TCR or US_TNCR have a value other than 0[(1)].

1: Both US_TCR and US_TNCR have a value of 0[(1)].

- **RXBUFF: RX Buffer Full (cleared by writing US_RCR or US_RNCR)**

0: US_RCR or US_RNCR have a value other than 0[(1)].

1: Both US_RCR and US_RNCR have a value of 0[(1)].

Note: 1. US_RCR, US_RNCR, US_TCR and US_TNCR are PDC registers.

- **LINBK: LIN Break Sent or LIN Break Received (cleared by writing a one to the bit US_CR.RSTSTA)**

Applicable if USART operates in LIN Master mode (USART_MODE = 0xA):

0: No LIN break has been sent since the last RSTSTA.

1: At least one LIN break has been sent since the last RSTSTA

If USART operates in LIN Slave mode (USART_MODE = 0xB):

0: No LIN break has been received since the last RSTSTA.

1:At least one LIN break has been received since the last RSTSTA.

- **LINID: LIN Identifier Sent or LIN Identifier Received (cleared by writing a one to the bit US_CR.RSTSTA)**

If USART operates in LIN Master mode (USART_MODE = 0xA):

0: No LIN identifier has been sent since the last RSTSTA.

1:At least one LIN identifier has been sent since the last RSTSTA.

If USART operates in LIN Slave mode (USART_MODE = 0xB):

0: No LIN identifier has been received since the last RSTSTA.

1:At least one LIN identifier has been received since the last RSTSTA

- **LINTC: LIN Transfer Completed (cleared by writing a one to the bit US_CR.RSTSTA)**

0: The USART is idle or a LIN transfer is ongoing.

1: A LIN transfer has been completed since the last RSTSTA.

- **LINBLS: LIN Bus Line Status**

0: LIN bus line is set to 0.

1: LIN bus line is set to 1.

- **LINBE: LIN Bit Error (cleared by writing a one to the bit US_CR.RSTSTA)**

0: No bit error has been detected since the last RSTSTA.

1: A bit error has been detected since the last RSTSTA.

- **LINISFE: LIN Inconsistent Synch Field Error (cleared by writing a one to the bit US_CR.RSTSTA)**

0: No LIN inconsistent synch field error has been detected since the last RSTSTA.

1: The USART is configured as a slave node and a LIN Inconsistent synch field error has been detected since the last RSTSTA.

- **LINIPE: LIN Identifier Parity Error (cleared by writing a one to the bit US_CR.RSTSTA)**

0: No LIN identifier parity error has been detected since the last RSTSTA.

1: A LIN identifier parity error has been detected since the last RSTSTA.

- **LINCE: LIN Checksum Error (cleared by writing a one to the bit US_CR.RSTSTA)**

0: No LIN checksum error has been detected since the last RSTSTA.

1: A LIN checksum error has been detected since the last RSTSTA.

- **LINSNRE: LIN Slave Not Responding Error (cleared by writing a one to the bit US_CR.RSTSTA)**

0: No LIN slave not responding error has been detected since the last RSTSTA.

1: A LIN slave not responding error has been detected since the last RSTSTA.

- **LINSTE: LIN Synch Tolerance Error (cleared by writing a one to the bit US_CR.RSTSTA)**

0: No LIN synch tolerance error has been detected since the last RSTSTA.

1: A LIN synch tolerance error has been detected since the last RSTSTA.

- **LINHTE: LIN Header Timeout Error (cleared by writing a one to the bit US_CR.RSTSTA)**

0: No LIN header timeout error has been detected since the last RSTSTA.

1: A LIN header timeout error has been detected since the last RSTSTA.

30.7.17 USART Receive Holding Register

Name: US_RHR

Address: 0x4000C218 (0), 0x40020218 (1), 0x40024218 (2), 0x40018218 (3), 0x4001C218 (4), 0x40008218 (5), 0x40040218 (6), 0x40034218 (7)

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.

30.7.18 USART Transmit Holding Register

Name: US_THR

Address: 0x4000C21C (0), 0x4002021C (1), 0x4002421C (2), 0x4001821C (3), 0x4001C21C (4), 0x4000821C (5), 0x4004021C (6), 0x4003421C (7)

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.

30.7.19 USART Baud Rate Generator Register

Name: US_BRGR

Address: 0x4000C220 (0), 0x40020220 (1), 0x40024220 (2), 0x40018220 (3), 0x4001C220 (4), 0x40008220 (5), 0x40040220 (6), 0x40034220 (7)

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".

- **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	CD = Selected Clock / (16 × Baud Rate)	CD = Selected Clock / (8 × Baud Rate)	CD = Selected Clock / Baud Rate	CD = Selected Clock / (FI_DI_RATIO × Baud Rate)

- **FP: Fractional Part**

0: Fractional divider is disabled.

1–7: Baud rate resolution, defined by FP × 1/8.

Warning: When the value of field FP is greater than 0, the SCK (oversampling clock) generates nonconstant duty cycles. The SCK high duration is increased by "selected clock" period from time to time. The duty cycle depends on the value of the CD field.

30.7.20 USART Receiver Timeout Register

Name: US_RTOR

Address: 0x4000C224 (0), 0x40020224 (1), 0x40024224 (2), 0x40018224 (3), 0x4001C224 (4), 0x40008224 (5), 0x40040224 (6), 0x40034224 (7)

Access: Read/Write

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

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

15	14	13	12	11	10	9	8
TO							

7	6	5	4	3	2	1	0
TO							

This register can only be written if the WPEN bit is cleared in ”USART Write Protection Mode Register”.

- **TO: Timeout Value**

0: The receiver timeout is disabled.

1–131071: The receiver timeout is enabled and the timeout delay is TO × bit period.

30.7.21 USART Transmitter Timeguard Register

Name: US_TTGR

Address: 0x4000C228 (0), 0x40020228 (1), 0x40024228 (2), 0x40018228 (3), 0x4001C228 (4), 0x40008228 (5), 0x40040228 (6), 0x40034228 (7)

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".

- **TG: Timeguard Value**

0: The transmitter timeguard is disabled.

1–255: The transmitter timeguard is enabled and TG is timeguard delay / bit period.

30.7.22 USART FI DI RATIO Register

Name: US_FIDI

Address: 0x4000C240 (0), 0x40020240 (1), 0x40024240 (2), 0x40018240 (3), 0x4001C240 (4), 0x40008240 (5), 0x40040240 (6), 0x40034240 (7)

Access: Read/Write

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

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

15	14	13	12	11	10	9	8
FI_DI_RATIO							

7	6	5	4	3	2	1	0
FI_DI_RATIO							

This register can only be written if the WPEN bit is cleared in ”USART Write Protection Mode Register”.

- **FI_DI_RATIO: FI Over DI Ratio Value**

0: If ISO7816 mode is selected, the baud rate generator generates no signal.

3 – 2047: If ISO7816 mode is selected, the baud rate is the clock provided on SCK divided by FI_DI_RATIO.

30.7.23 USART Number of Errors Register

Name: US_NER

Address: 0x4000C244 (0), 0x40020244 (1), 0x40024244 (2), 0x40018244 (3), 0x4001C244 (4), 0x40008244 (5), 0x40040244 (6), 0x40034244 (7)

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”.

- **NB_ERRORS: Number of Errors**

Total number of errors that occurred during an ISO7816 transfer. This register automatically clears when read.

30.7.24 USART IrDA FILTER Register

Name: US_IF

Address: 0x4000C24C (0), 0x4002024C (1), 0x4002424C (2), 0x4001824C (3), 0x4001C24C (4), 0x4000824C (5), 0x4004024C (6), 0x4003424C (7)

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”.

This register can only be written if the WPEN bit is cleared in ”USART Write Protection Mode Register”.

- **IRDA_FILTER: IrDA Filter**

The IRDA_FILTER value must be defined to meet the following criteria:

$t_{peripheral\ clock} \times (IRDA\_FILTER + 3) < 1.41\ \mu s$

30.7.25 USART LIN Mode Register

Name: US_LINMR

Address: 0x4000C254 (0), 0x40020254 (1), 0x40024254 (2), 0x40018254 (3), 0x4001C254 (4), 0x40008254 (5), 0x40040254 (6), 0x40034254 (7)

Access: Read/Write

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

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

15	14	13	12	11	10	9	8
DLC							

7	6	5	4	3	2	1	0
WKUPTYP	FSDIS	DLM	CHKTYP	CHKDIS	PARDIS	NACT	

This register is relevant only if USART_MODE = 0xA or 0xB in ”USART Mode Register”.

This register can only be written if the WPEN bit is cleared in ”USART Write Protection Mode Register”.

- **NACT: LIN Node Action**

Value	Name	Description
00	PUBLISH	The USART transmits the response.
01	SUBSCRIBE	The USART receives the response.
10	IGNORE	The USART does not transmit and does not receive the response.

Values which are not listed in the table must be considered as “reserved”.

- **PARDIS: Parity Disable**

0: In master node configuration, the identifier parity is computed and sent automatically. In master node and slave node configuration, the parity is checked automatically.

1: Whatever the node configuration is, the Identifier parity is not computed/sent and it is not checked.

- **CHKDIS: Checksum Disable**

0: In master node configuration, the checksum is computed and sent automatically. In slave node configuration, the checksum is checked automatically.

1: Whatever the node configuration is, the checksum is not computed/sent and it is not checked.

- **CHKTYP: Checksum Type**

0: LIN 2.0 “enhanced” checksum

1: LIN 1.3 “classic” checksum

- **DLM: Data Length Mode**

0: The response data length is defined by the field DLC of this register.

1: The response data length is defined by the bits 5 and 6 of the identifier (IDCHR in US_LINIR).

- **FSDIS: Frame Slot Mode Disable**

0: The Frame Slot mode is enabled.

1: The Frame Slot mode is disabled.

- **WKUPTYP: Wakeup Signal Type**

0: Setting the bit LINWKUP in the control register sends a LIN 2.0 wakeup signal.

1: Setting the bit LINWKUP in the control register sends a LIN 1.3 wakeup signal.

- **DLC: Data Length Control**

0–255: Defines the response data length if DLM = 0,in that case the response data length is equal to DLC+1 bytes.

- **PDCM: PDC Mode**

0: The LIN mode register US_LINMR is not written by the PDC.

1: The LIN mode register US_LINMR (excepting that flag) is written by the PDC.

- **SYNCDIS: Synchronization Disable**

0: The synchronization procedure is performed in LIN slave node configuration.

1: The synchronization procedure is not performed in LIN slave node configuration.

30.7.26 USART LIN Identifier Register

Name: US_LINIR

Address: 0x4000C258 (0), 0x40020258 (1), 0x40024258 (2), 0x40018258 (3), 0x4001C258 (4), 0x40008258 (5), 0x40040258 (6), 0x40034258 (7)

Access: Read/Write or Read-only

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

This register is relevant only if USART_MODE = 0xA or 0xB in ”USART Mode Register”.

- **IDCHR: Identifier Character**

If USART_MODE = 0xA (master node configuration):

IDCHR is Read/Write and its value is the identifier character to be transmitted.

If USART_MODE = 0xB (slave node configuration):

IDCHR is Read-only and its value is the last identifier character that has been received.

30.7.27 USART LIN Baud Rate Register

Name: US_LINBRR

Address: 0x4000C25C (0), 0x4002025C (1), 0x4002425C (2), 0x4001825C (3), 0x4001C25C (4), 0x4000825C (5), 0x4004025C (6), 0x4003425C (7)

Access: Read-only

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

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

15	14	13	12	11	10	9	8
LINCD							

7	6	5	4	3	2	1	0
LINCD							

This register is relevant only if USART_MODE = 0xA or 0xB in ”USART Mode Register”.

Returns the baud rate value after the synchronization process completion.

- **LINCD: Clock Divider after Synchronization**
- **LINFP: Fractional Part after Synchronization**

30.7.28 USART Comparison Register

Name: US_CMPR

Address: 0x4000C290 (0), 0x40020290 (1), 0x40024290 (2), 0x40018290 (3), 0x4001C290 (4), 0x40008290 (5), 0x40040290 (6), 0x40034290 (7)

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	VAL2

23	22	21	20	19	18	17	16
VAL2							

15	14	13	12	11	10	9	8
–	CMPPAR	–	CMPMODE	–	–	–	VAL1

7	6	5	4	3	2	1	0
VAL1							

This register can only be written if the WPEN bit is cleared in ”USART Write Protection Mode Register”.

- **VAL1: First Comparison Value for Received Character**

0 to 511.

- **CMPMODE: Comparison Mode**

Value	Name	Description
0	FLAG_ONLY	Any character is received and comparison function drives CMP flag.
1	START_CONDITION	Comparison condition must be met to start reception of all incoming characters until REQCLR is set.

- **CMPPAR: Compare Parity**

0: The parity is not checked and a bad parity cannot prevent the system from waking up.

1: The system wakeup is performed if both a matching condition on data exists and the parity is correct.

- **VAL2: Second Comparison Value for Received Character**

0 to 511.

30.7.29 USART Write Protection Mode Register

Name: US_WPMR

Address: 0x4000C2E4 (0), 0x400202E4 (1), 0x400242E4 (2), 0x400182E4 (3), 0x4001C2E4 (4), 0x400082E4 (5), 0x400402E4 (6), 0x400342E4 (7)

Access: Read/Write

31	30	29	28	27	26	25	24
WPKEY							

23	22	21	20	19	18	17	16
WPKEY							

15	14	13	12	11	10	9	8
WPKEY							

7	6	5	4	3	2	1	0
—	—	—	—	—	—	—	WPEN

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x555341 (“USA” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x555341 (“USA” in ASCII).

See Section 30.6.11 ”USART Register Write Protection” for the list of registers that can be write-protected.

- **WPKEY: Write Protection Key**

Value	Name	Description
0x555341	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

30.7.30 USART Write Protection Status Register

Name: US_WPSR

Address: 0x4000C2E8 (0), 0x400202E8 (1), 0x400242E8 (2), 0x400182E8 (3), 0x4001C2E8 (4), 0x400082E8 (5), 0x400402E8 (6), 0x400342E8 (7)

Access: Read-only

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

23	22	21	20	19	18	17	16
WPVSRC							

15	14	13	12	11	10	9	8
WPVSRC							

7	6	5	4	3	2	1	0
—	—	—	—	—	—	—	WPVS

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of the US_WPSR.

1: A write protection violation has occurred since the last read of the US_WPSR. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protection Violation Source**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

31. Serial Peripheral Interface (SPI)

31.1 Description

The Serial Peripheral Interface (SPI) circuit is a synchronous serial data link that provides communication with external devices in Master or Slave mode. It also enables communication between processors if an external processor is connected to the system.

The Serial Peripheral Interface is essentially a Shift register that serially transmits data bits to other SPIs. During a data transfer, one SPI system acts as the “master”' which controls the data flow, while the other devices act as “slaves'' which have data shifted into and out by the master. Different CPUs can take turn being masters (multiple master protocol, contrary to single master protocol where one CPU is always the master while all of the others are always slaves). One master can simultaneously shift data into multiple slaves. However, only one slave can drive its output to write data back to the master at any given time.

A slave device is selected when the master asserts its NSS signal. If multiple slave devices exist, the master generates a separate slave select signal for each slave (NPCS).

The SPI system consists of two data lines and two control lines:

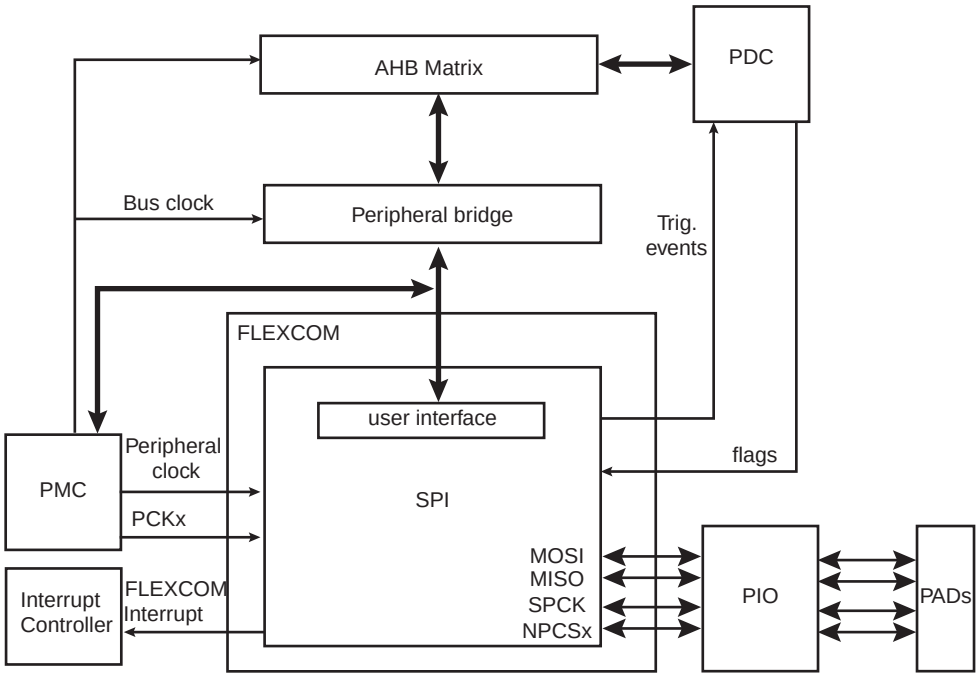
- Master Out Slave In (MOSI): This data line supplies the output data from the master shifted into the input(s) of the slave(s).
- Master In Slave Out (MISO): This data line supplies the output data from a slave to the input of the master. There may be no more than one slave transmitting data during any particular transfer.
- Serial Clock (SPCK): This control line is driven by the master and regulates the flow of the data bits. The master can transmit data at a variety of bit rates; there is one SPCK pulse for each bit that is transmitted.
- Slave Select (NSS): This control line allows slaves to be turned on and off by hardware.

31.2 Embedded Characteristics

- Master or Slave Serial Peripheral Bus Interface
  - 8-bit to 16-bit programmable data length per chip select
  - Programmable phase and polarity per chip select
  - Programmable transfer delay between consecutive transfers and delay before SPI clock per chip select
  - Programmable delay between chip selects
- Asynchronous Partial Wakeup (SleepWalking) Capability
- Master mode can drive SPCK up to Peripheral Clock
- Master mode bit rate can be independent of the processor/peripheral clock
- Slave mode operates on SPCK, asynchronously with core and bus clock
- Two chip selects with external decoder support allow communication with up to 3 peripherals
- Selectable Mode Fault Detection
- Communication with Serial External Devices Supported
  - Serial memories, such as DataFlash and 3-wire EEPROMs
  - Serial peripherals, such as ADCs, DACs, LCD controllers, CAN controllers and sensors
  - External coprocessors
- Connection to PDC Channel Capabilities, Optimizing Data Transfers
  - One channel for the receiver
  - One channel for the transmitter

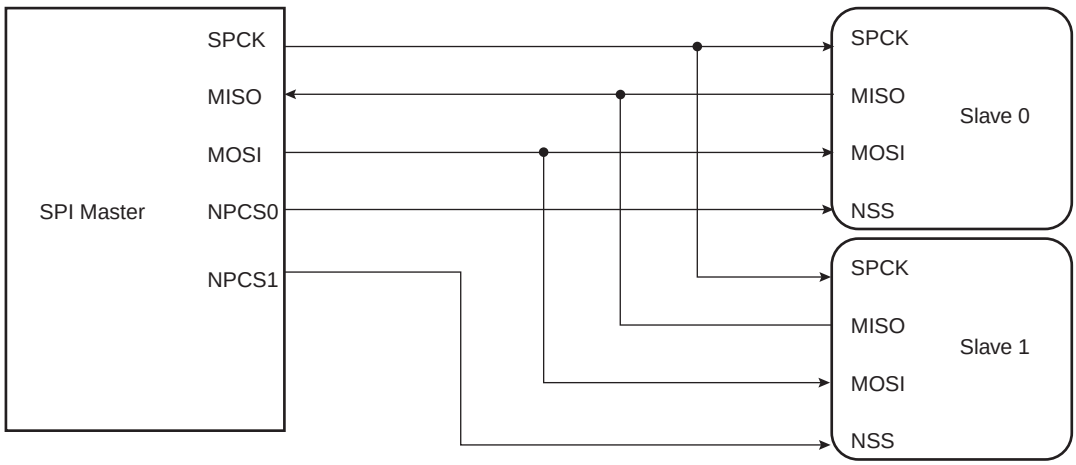
31.3 Block Diagram

Figure 31-1. SPI Block Diagram



31.4 Application Block Diagram

Figure 31-2. Application Block Diagram: Single Master/Multiple Slave Implementation



31.5 I/O Lines Description

Table 31-1. 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	Peripheral Chip Select	Output	Unused
NPCS0/NSS	Peripheral Chip Select/Slave Select	Output	Input

31.6 Product Dependencies

31.6.1 I/O Lines

The pins used for interfacing the SPI are multiplexed with the USART and TWI lines within the FLEXCOM module. The programmer must first program the PIO controller to assign the desired FLEXCOM pins to their peripheral function. If the I/O lines of the FLEXCOM are not used by the application, they can be used for other purposes by the PIO controller.

Table 31-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
SPI0	RXD0/SPI0_MISO/TWCK0	PA9	A
SPI0	SCK0/SPI0_SPCK	PB0	A
SPI0	SPI0_NPCS0/CTS0	PA25	A
SPI0	SPI0_NPCS1/RTS0	PA26	A
SPI0	TXD0/SPI0_MOSI/TWD0	PA10	A
SPI1	RXD1/SPI1_MISO/TWCK1	PB2	A
SPI1	SCK1/SPI1_SPCK	PA27	A
SPI1	SPI1_NPCS0/CTS1	PA28	A
SPI1	SPI1_NPCS1/RTS1	PA29	A
SPI1	TXD1/SPI1_MOSI/TWD1	PB3	A
SPI2	RXD2/SPI2_MISO/TWCK2	PA5	A
SPI2	SCK2/SPI2_SPCK	PA15	B
SPI2	SCK2/SPI2_SPCK	PA24	B
SPI2	SPI2_NPCS0/CTS2	PA16	A
SPI2	SPI2_NPCS1/RTS2	PA15	A
SPI2	TXD2/SPI2_MOSI/TWD2	PA6	A
SPI3	RXD3/SPI3_MISO/TWCK3	PA4	A
SPI3	SCK3/SPI3_SPCK	PB13	A
SPI3	SPI3_NPCS0/CTS3	PB14	A
SPI3	SPI3_NPCS1/RTS3	PB15	A

Table 31-2. I/O Lines (Continued)

SPI3	TXD3/SPI3_MOSI/TWD3	PA3	A
SPI4	RXD4/SPI4_MISO/TWCK4	PB9	A
SPI4	RXD4/SPI4_MISO/TWCK4	PB11	A
SPI4	SCK4/SPI4_SPCK	PB1	A
SPI4	SPI4_NPCS0/CTS4	PB8	B
SPI4	SPI4_NPCS1/RTS4	PB9	B
SPI4	TXD4/SPI4_MOSI/TWD4	PB8	A
SPI4	TXD4/SPI4_MOSI/TWD4	PB10	A
SPI5	RXD5/SPI5_MISO/TWCK5	PA12	A
SPI5	SCK5/SPI5_SPCK	PA14	A
SPI5	SPI5_NPCS0/CTS5	PA11	A
SPI5	SPI5_NPCS1/RTS5	PA5	B
SPI5	SPI5_NPCS1/RTS5	PB2	B
SPI5	TXD5/SPI5_MOSI/TWD5	PA13	A
SPI6	RXD6/SPI6_MISO/TWCK6	PB1	B
SPI6	RXD6/SPI6_MISO/TWCK6	PB11	B
SPI6	SCK6/SPI6_SPCK	PB13	B
SPI6	SPI6_NPCS0/CTS6	PB14	B
SPI6	SPI6_NPCS1/RTS6	PB15	B
SPI6	TXD6/SPI6_MOSI/TWD6	PB0	B
SPI6	TXD6/SPI6_MOSI/TWD6	PB10	B
SPI7	RXD7/SPI7_MISO/TWCK7	PA27	B
SPI7	SCK7/SPI7_SPCK	PA29	B
SPI7	SPI7_NPCS0/CTS7	PA30	B
SPI7	SPI7_NPCS1/RTS7	PA31	B
SPI7	TXD7/SPI7_MOSI/TWD7	PA28	B

31.6.2 Power Management

The peripheral clock is not continuously provided to the SPI. The programmer must first enable the FLEXCOM clock in the Power Management Controller (PMC) and set the OPMODE field to 2 in FLEXCOM Mode Register (FLEX_MR) before using the SPI.

If the OPMODE field differs from 2, the SPI clock is stopped.

In SleepWalking mode (asynchronous partial wakeup), the PMC must be configured to enable SleepWalking for the FLEXCOM in the Sleepwalking Enable Register (PMC_SLPWK_ER). The peripheral clock can be automatically provided to the FLEXCOM, depending on the instructions (requests) provided by the FLEXCOM to the PMC.

31.6.3 Interrupt

The SPI interrupt line is the FLEXCOM interrupt line if the field OPMODE = 2 in FLEX_MR. The FLEXCOM interrupt line is connected to one of the internal sources of the Interrupt Controller. Using the FLEXCOM interrupt requires the Interrupt Controller to be programmed first.

Table 31-3. Peripheral IDs

Instance	ID
SPI0	8
SPI1	9
SPI2	14
SPI3	19
SPI4	20
SPI5	21
SPI6	22
SPI7	7

31.7 SPI Functional Description

31.7.1 Modes of Operation

The SPI operates in Master mode or in Slave mode.

- The SPI operates in Master mode by setting the MSTR bit in the SPI Mode Register (SPI_MR):
  - The NPCS0 to NPCS1 pins are all configured as outputs
  - The SPCK pin is driven
  - The MISO line is wired on the receiver input
  - The MOSI line is driven as an output by the transmitter.
- The SPI operates in Slave mode if the MSTR bit in SPI_MR is cleared:
  - 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 NPCS1 pin is not driven and can be used for other purposes.

The data transfers are identically programmable for both modes of operations. The bit rate generator is activated only in Master mode.

31.7.2 Data Transfer

Four combinations of polarity and phase are available for data transfers. The clock polarity is programmed with the CPOL bit in the SPI Chip Select register (SPI_CSR). The clock phase is programmed with the NCPHA bit. These two parameters determine the edges of the clock signal on which data is driven and sampled. Each of the two parameters has two possible states, resulting in four possible combinations that are incompatible with one another. Consequently, a master/slave pair must use the same parameter pair values to communicate. If multiple slaves are connected and require different configurations, the master must reconfigure itself each time it needs to communicate with a different slave.

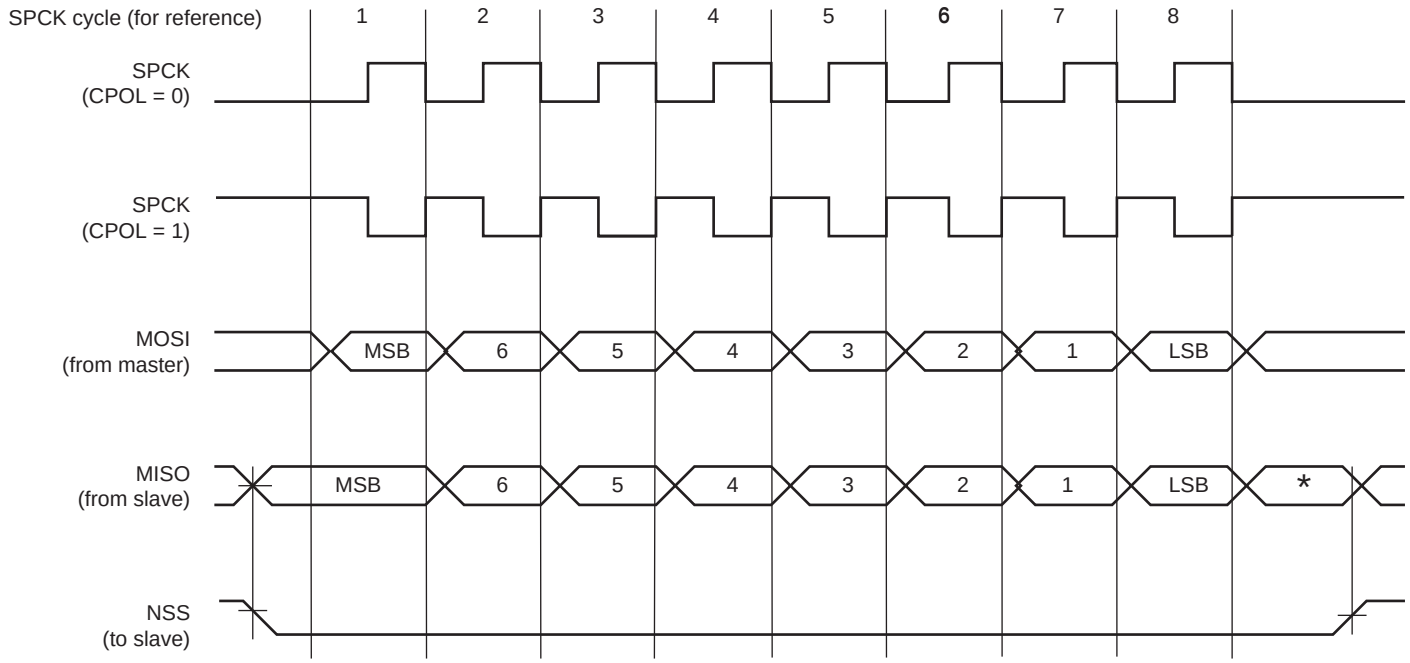
Table 31-4 shows the four modes and corresponding parameter settings.

Table 31-4. 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

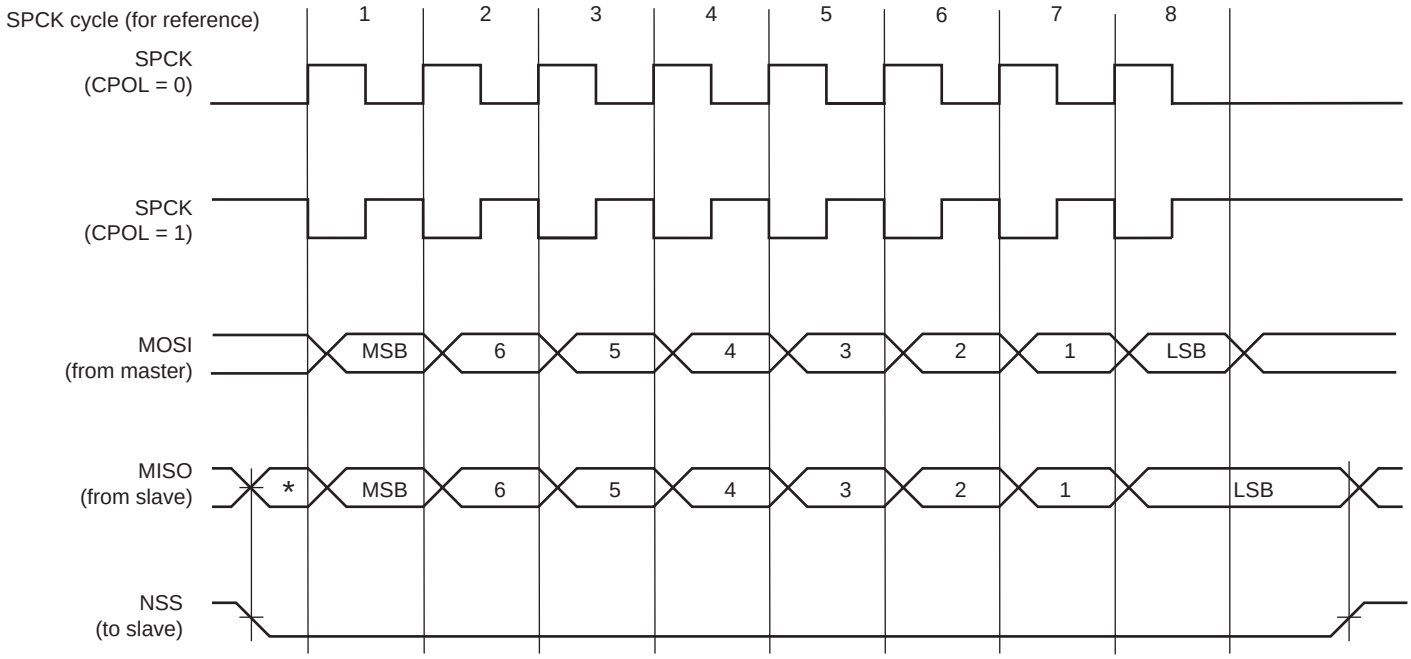
Figure 31-3 and Figure 31-4 show examples of data transfers.

Figure 31-3. SPI Transfer Format (NCPHA = 1, 8 bits per transfer)



* Not defined.

Figure 31-4. SPI Transfer Format (NCPHA = 0, 8 bits per transfer)



* Not defined.

31.7.3 Master Mode Operations

When configured in Master mode, the SPI operates on the clock generated by the internal programmable bit rate generator. It fully controls the data transfers to and from the slave(s) connected to the SPI bus. The SPI drives the chip select line to the slave and the serial clock signal (SPCK).

The SPI features two holding registers, the Transmit Data register (SPI_TDR) and the Receive Data register (SPI_RDR), and a single shift register. The holding registers maintain the data flow at a constant rate.

After enabling the SPI, a data transfer starts when the processor writes to SPI_TDR. The written data is immediately transferred in the Shift register and the transfer on the SPI bus starts. While the data in the Shift register is shifted on the MOSI line, the MISO line is sampled and shifted in the Shift register. Data cannot be loaded in the SPI_RDR without transmitting data. If there is no data to transmit, a dummy data can be used (SPI_TDR filled with ones). When the WDRBT bit is set, a new data cannot be transmitted if the SPI_RDR has not been read. If Receiving mode is not required, for example when communicating with a slave receiver only (such as an LCD), the receive status flags in the SPI Status register (SPI_SR) can be discarded.

Before writing the TDR, the PCS field in SPI_MR must be set in order to select a slave.

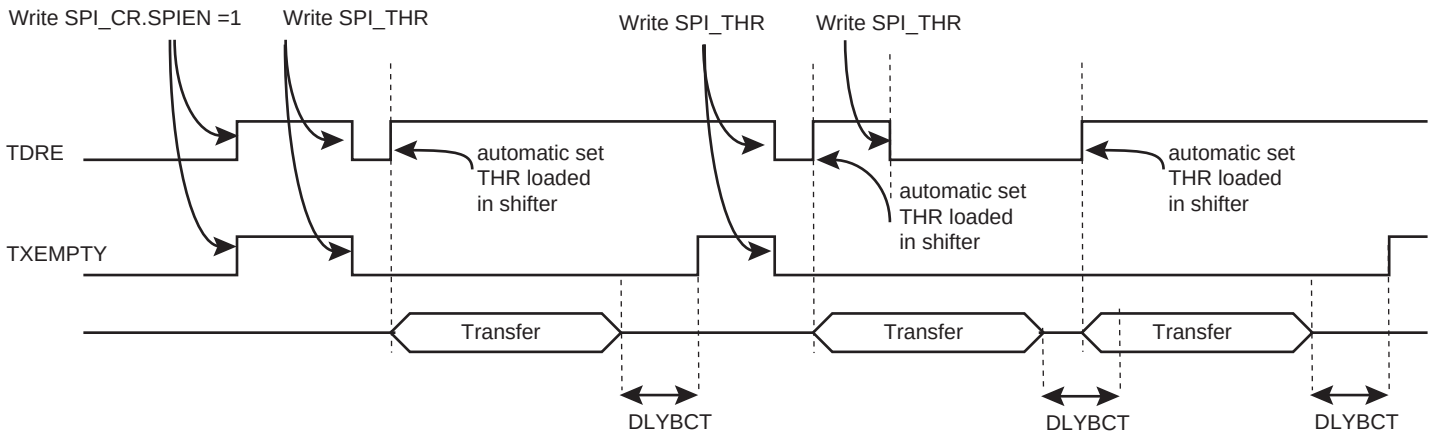
If new data is written in SPI_TDR during the transfer, it is kept in SPI_TDR until the current transfer is completed. Then, the received data is transferred from the Shift register to the SPI_RDR, the data in the SPI_TDR is loaded in the Shift register and a new transfer starts.

As soon as the SPI_TDR is written, the Transmit Data Register Empty (TDRE) flag in the SPI_SR is cleared. When the data written in the SPI_TDR is loaded into the Shift register, the TDRE flag in the SPI_SR is set. The TDRE bit is used to trigger the Transmit PDCchannel (see Figure 31-5).

The end of transfer is indicated by the TXEMPTY flag in SPI_SR. If a transfer delay (DLYBCT) is greater than 0 for the last transfer, TXEMPTY is set after the completion of this delay. The peripheral clock can be switched off at this time.

Note: When the SPI is enabled, the TDRE and TXEMPTY flags are set.

Figure 31-5. TDRE and TXEMPTY flag behavior



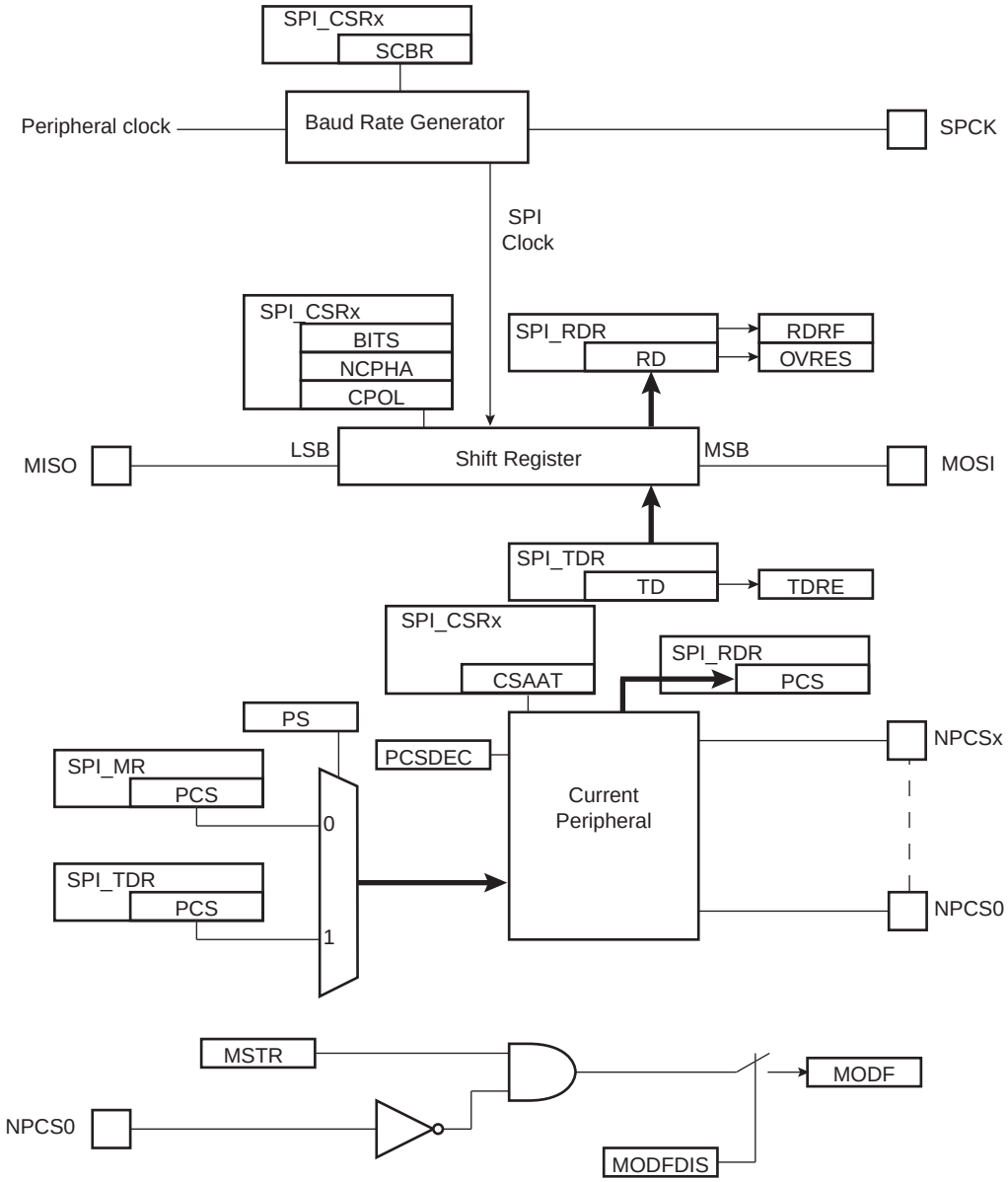
The transfer of received data from the Shift register to the SPI_RDR is indicated by the Receive Data Register Full bit (RDRF) in the SPI_SR. When the received data is read, the RDRF bit is cleared.

If SPI_RDR 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 31-6 shows a block diagram of the SPI when operating in Master mode. Figure 31-7 ”Master Mode Flow Diagram” shows a flow chart describing how transfers are handled.

31.7.3.1 Master Mode Block Diagram

Figure 31-6. Master Mode Block Diagram



31.7.3.2 Master Mode Flow Diagram

Figure 31-7. Master Mode Flow Diagram

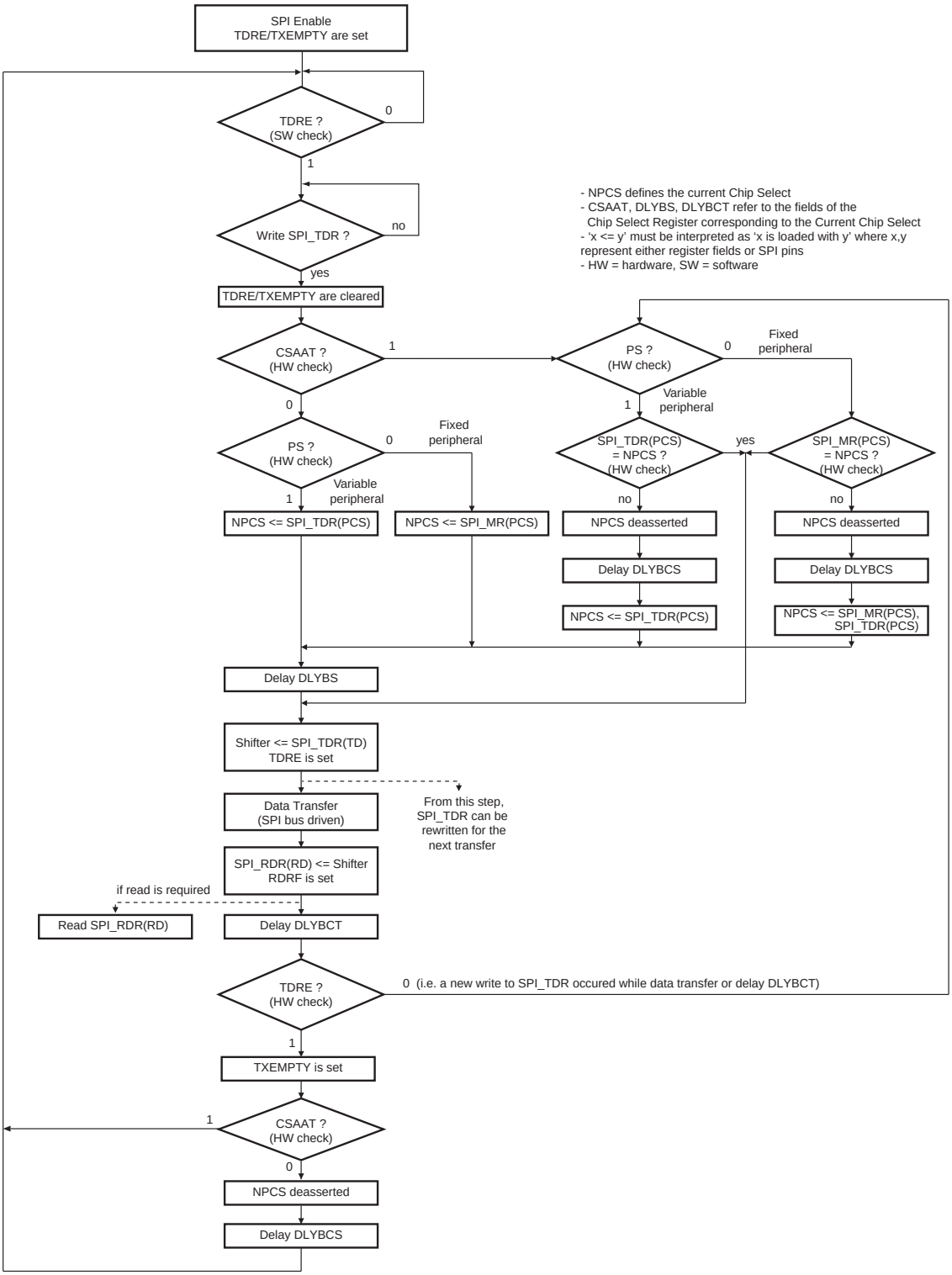


Figure 31-8 shows the behavior of Transmit Data Register Empty (TDRE), Receive Data Register (RDRF) and Transmission Register Empty (TXEMPTY) status flags within the SPI_SR during an 8-bit data transfer in Fixed mode without the PDC involved.

Figure 31-8. Status Register Flags Behavior

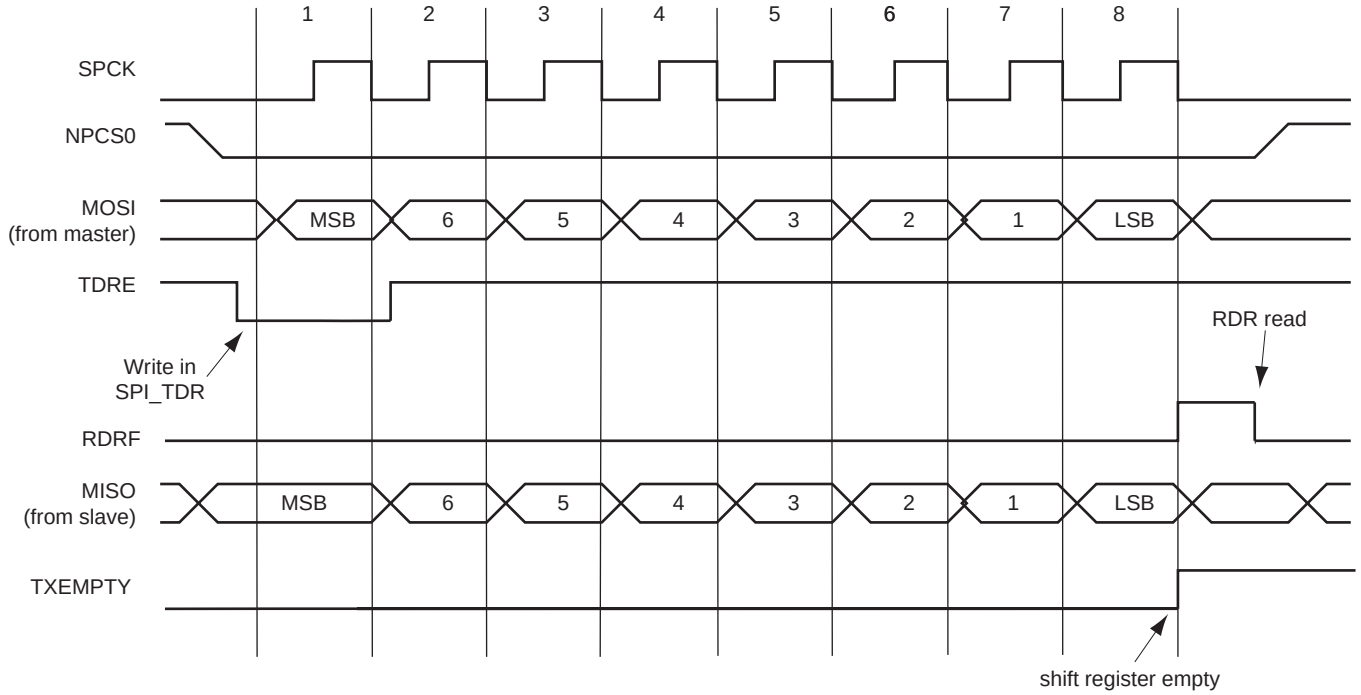
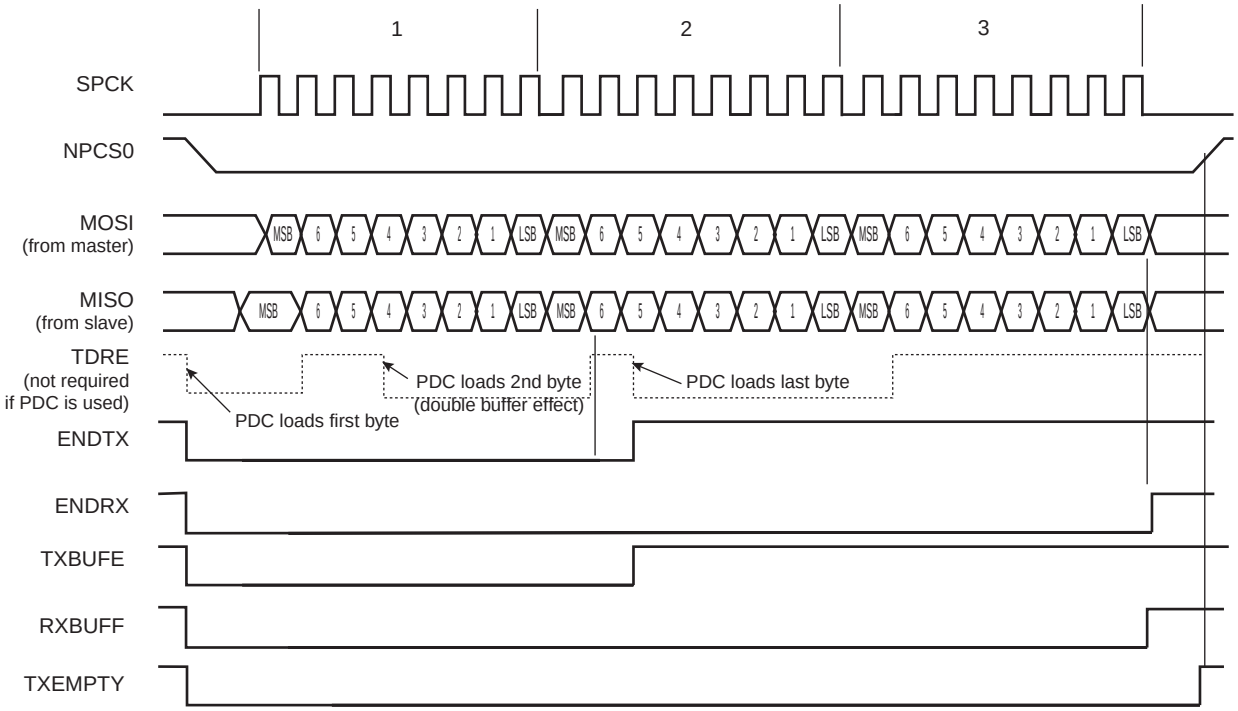


Figure 31-9 shows the behavior of Transmission Register Empty (TXEMPTY), End of RX buffer (ENDRX), End of TX buffer (ENDTX), RX Buffer Full (RXBUFF) and TX Buffer Empty (TXBUFE) status flags within SPI_SR during an 8-bit data transfer in Fixed mode with the Peripheral Data Controller involved. The PDC 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 31-9. PDC Status Register Flags Behavior



31.7.3.3 Clock Generation

The SPI bit rate clock is generated by dividing a source clock, which can be the peripheral clock or a programmable clock from the PMC (PCKx). The divider can be any value between 1 and 255.

If the SCBR field is programmed to 1and the clock source is PMC PCKx, the operating bit rate is peripheral clock (see the electrical characteristics section for the SPCK maximum frequency). Triggering a transfer while SCBR is at 0 can lead to unpredictable results.

At reset, SCBR is 0 and the user has to program it to a valid value before performing the first transfer.

The divisor can be defined independently for each chip select, as it has to be programmed in the SCBR field in SPI_CSR. This allows the SPI to automatically adapt the bit rate for each interfaced peripheral without reprogramming.

If PCKx is selected as a source clock (BRSRCCLK = 1 in SPI_MR), the bit rate is independent of the processor/bus clock. Thus the processor clock can be changed while the SPI is enabled. The processor clock frequency changes must be performed only by programming the PRES field in PMC_MCKR (see PMC section). Other methods to modify the processor/bus clock frequency (PLL multiplier, etc.) are forbidden when SPI is enabled.

The peripheral clock frequency must be at least tree times higher than PCKx.

31.7.3.4 Transfer Delays

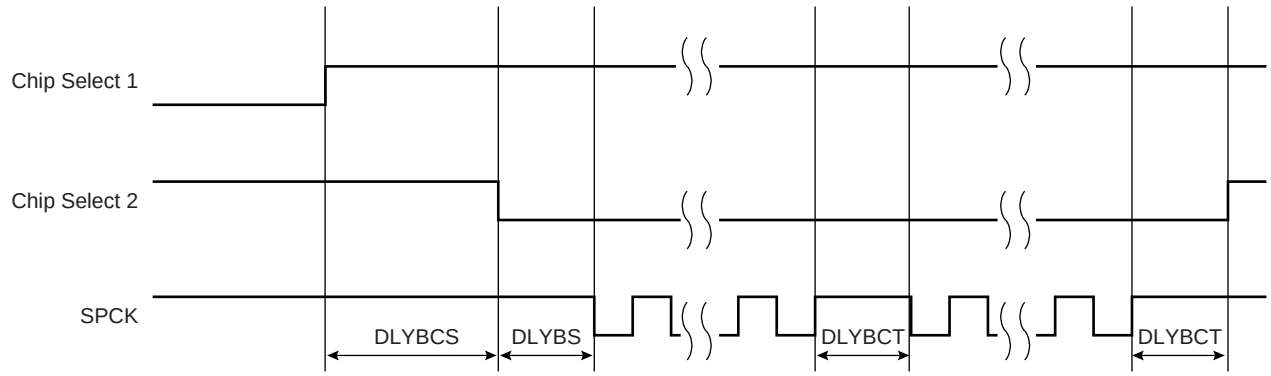
Figure 31-10 shows a chip select transfer change and consecutive transfers on the same chip select. Three delays can be programmed to modify the transfer waveforms:

- The delay between the chip selects. It is programmable only once for all chip selects by writing the DLYBCS field in SPI_MR. The SPI slave device deactivation delay is managed through DLYBCS. If there is only one SPI slave device connected to the master, the DLYBCS field does not need to be configured. If several slave devices are connected to a master, DLYBCS must be configured depending on the highest deactivation delay. Refer to the SPI slave device electrical characteristics.

- The delay before SPCK, independently programmable for each chip select by writing the DLYBS field. The SPI slave device activation delay is managed through DLYBS. Refer to the SPI slave device electrical characteristics to define DLYBS.
- The delay between consecutive transfers, independently programmable for each chip select by writing the DLYBCT field. The time required by the SPI slave device to process received data is managed through DLYBCT. This time depends on the SPI slave system activity.

These delays allow the SPI to be adapted to the interfaced peripherals and their speed and bus release time.

Figure 31-10. Programmable Delays



31.7.3.5 Peripheral Selection

The serial peripherals are selected through the assertion of the NPCS0 to NPCS1 signals. By default, all NPCS signals are high before and after each transfer.

- **Fixed Peripheral Select Mode**: SPI exchanges data with only one peripheral.
  Fixed peripheral select mode is enabled by clearing the PS bit in SPI_MR. In this case, the current peripheral is defined by the PCS field in SPI_MR and the PCS field in SPI_TDR has no effect.
- **Variable Peripheral Select Mode:** Data can be exchanged with more than one peripheral without having to reprogram the NPCS field in SPI_MR.
  Variable peripheral select Mode is enabled by setting the PS bit to one in SPI_MR. 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 SPI_TDR has the following format:

  [xxxxxxx(7-bit) + LASTXFER(1-bit)(1)+ xxxx(4-bit) + PCS (4-bit) + DATA (8 to 16-bit)] with PCS equals the chip select to assert, as defined in Section 31.8.4 ”SPI Transmit Data Register” and LASTXFER bit at 0 or 1 depending on the CSAAT bit.

Note: 1. Optional

CSAAT, LASTXFER and CSNAAT bits are discussed in Section 31.7.3.9 ”Peripheral Deselection with PDC”.

If LASTXFER is used, the command must be issued after writing the last character. Instead of LASTXFER, the user can use the SPIDIS command. After the end of the PDC transfer, it is necessary to wait for the TXEMPTY flag and then write SPIDIS into the SPI Control Register (SPI_CR). This does not change the configuration register values). The NPCS is disabled after the last character transfer. Then, another PDC transfer can be started if the SPIEN has previously been written in SPI_CR.

31.7.3.6 SPI Peripheral DMA Controller (PDC)

In both Fixed and Variable peripheral select modes, the Peripheral DMA Controller (PDC) can be used to reduce processor overhead.

The fixed peripheral selection allows buffer transfers with a single peripheral. Using the PDC is an optimal means, as the size of the data transfer between the memory and the SPI is either 8 bits or 16 bits. However, if the peripheral selection is modified, the SPI_MR must be reprogrammed.

The variable peripheral selection allows buffer transfers with multiple peripherals without reprogramming the SPI_MR. Data written in SPI_TDR is 32 bits wide and defines the real data to be transmitted and the destination peripheral. Using the PDC in this mode requires 32-bit wide buffers, with the data in the LSBs and the PCS and LASTXFER fields in the MSBs. However, the SPI still controls the number of bits (8 to16) to be transferred through MISO and MOSI lines with the chip select configuration registers (SPI_CSRx). This is not the optimal means in terms of memory size for the buffers, but it provides a very effective means to exchange data with several peripherals without any intervention of the processor.

Transfer Size

Depending on the data size to transmit, from 8 to 16 bits, the PDC manages automatically the type of pointer size it has to point to. The PDC performs the following transfer, depending on the mode and number of bits per data.

- Fixed mode:
  - 8-bit data:
    1-Byte transfer,
    PDC pointer address = address + 1 byte,
    PDC counter = counter - 1
  - 9-bit to 16-bit data:
    2-Byte transfer. n-bit data transfer with don't care data (MSB) filled with zeros,
    PDC pointer address = address + 2 bytes,
    PDC counter = counter - 1
- Variable mode:
  - In Variable mode, PDC pointer address = address +4 bytes and PDC counter = counter - 1 for 8 to 16-bit transfer size.
  - When using the PDC, the TDRE and RDRF flags are handled by the PDC. The user's application does not have to check these bits. Only End of RX Buffer (ENDRX), End of TX Buffer (ENDTX), Buffer Full (RXBUFF), TX Buffer Empty (TXBUFE) are significant. For further details about the Peripheral DMA Controller and user interface, refer to the PDC section of the product datasheet.

31.7.3.7 Peripheral Chip Select Decoding

The user can program the SPI to operate with up to 3 slave peripherals by decoding the two chip select lines, NPCS0 to NPCS1 with an external decoder/demultiplexer (refer to Figure 31-11). This can be enabled by setting the PCSDEC bit in SPI_MR.

When operating without decoding, the SPI makes sure that in any case only one chip select line is activated, i.e., one NPCS line driven low at a time. If two bits are defined low in a PCS field, only the lowest numbered chip select is driven low.

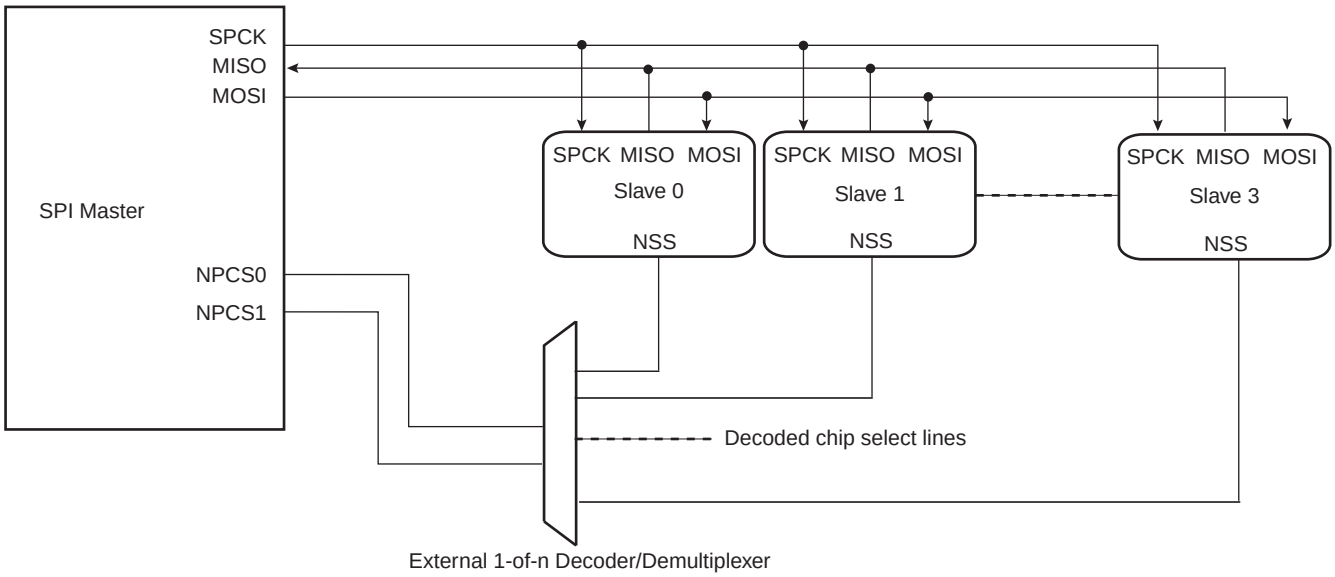
When operating with decoding, the SPI directly outputs the value defined by the PCS field on the NPCS lines of either SPI_MR or SPI_TDR (depending on PS).

As the SPI sets a default value of 0x3 on the chip select lines (i.e. all chip select lines at 1) when not processing any transfer, only 3 peripherals can be decoded.

The SPI has only two Chip Select registers. As a result, when external decoding is activated, each NPCS chip select defines the characteristics of up to two peripherals. As an example, SPI_CRS0 defines the characteristics of the externally decoded peripherals 0 to 1, corresponding to the PCS values 0x0 to 0x1. Consequently, the user has to make sure to connect compatible peripherals on the decoded chip select lines 0 to 1 and 2. Figure 31-11 shows this type of implementation.

If the CSAAT bit is used, with or without the PDC, the Mode Fault detection for NPCS0 line must be disabled. This is not required for all other chip select lines since mode fault detection is only on NPCS0.

Figure 31-11. Chip Select Decoding Application Block Diagram: Single Master/Multiple Slave Implementation



31.7.3.8 Peripheral Deselection without PDC

During a transfer of more than one data on a Chip Select without the PDC, SPI_TDR is loaded by the processor, the TDRE flag rises as soon as the content of SPI_TDR is transferred into the internal Shift register. When this flag is detected high, SPI_TDR can be reloaded. If this reload by the processor occurs before the end of the current transfer and if the next transfer is performed on the same chip select as the current transfer, the Chip Select is not de-asserted between the two transfers. But depending on the application software handling the SPI status register flags (by interrupt or polling method) or servicing other interrupts or other tasks, the processor may not reload the SPI_TDR in time to keep the chip select active (low). A null DLYBCT value (delay between consecutive transfers) in SPI_CSR, will give even less time for the processor to reload the SPI_TDR. With some SPI slave peripherals, if the chip select line must remain active (low) during a full set of transfers, communication errors can occur.

To facilitate interfacing with such devices, the Chip Select registers [CSR0...CSR1] can be programmed with the Chip Select Active After Transfer (CSAAT) bit to 1. This allows the chip select lines to remain in their current state (low = active) until a transfer to another chip select is required. Even if the SPI_TDR is not reloaded, the chip select remains active. To de-assert the chip select line at the end of the transfer, the Last transfer Bit (LASTXFER) in SPI_CR must be set after writing the last data to transmit into SPI_TDR.

31.7.3.9 Peripheral Deselection with PDC

PDC provides faster reloads of SPI_TDR compared to software. However, depending on the system activity, it is not guaranteed that the SPI_TDR is written with the next data before the end of the current transfer. Consequently, a data can be lost by the de-assertion of the NPCS line for SPI slave peripherals requiring the chip select line to remain active between two transfers. The only way to guarantee a safe transfer in this case is the use of the CSAAT and LASTXFER bits.

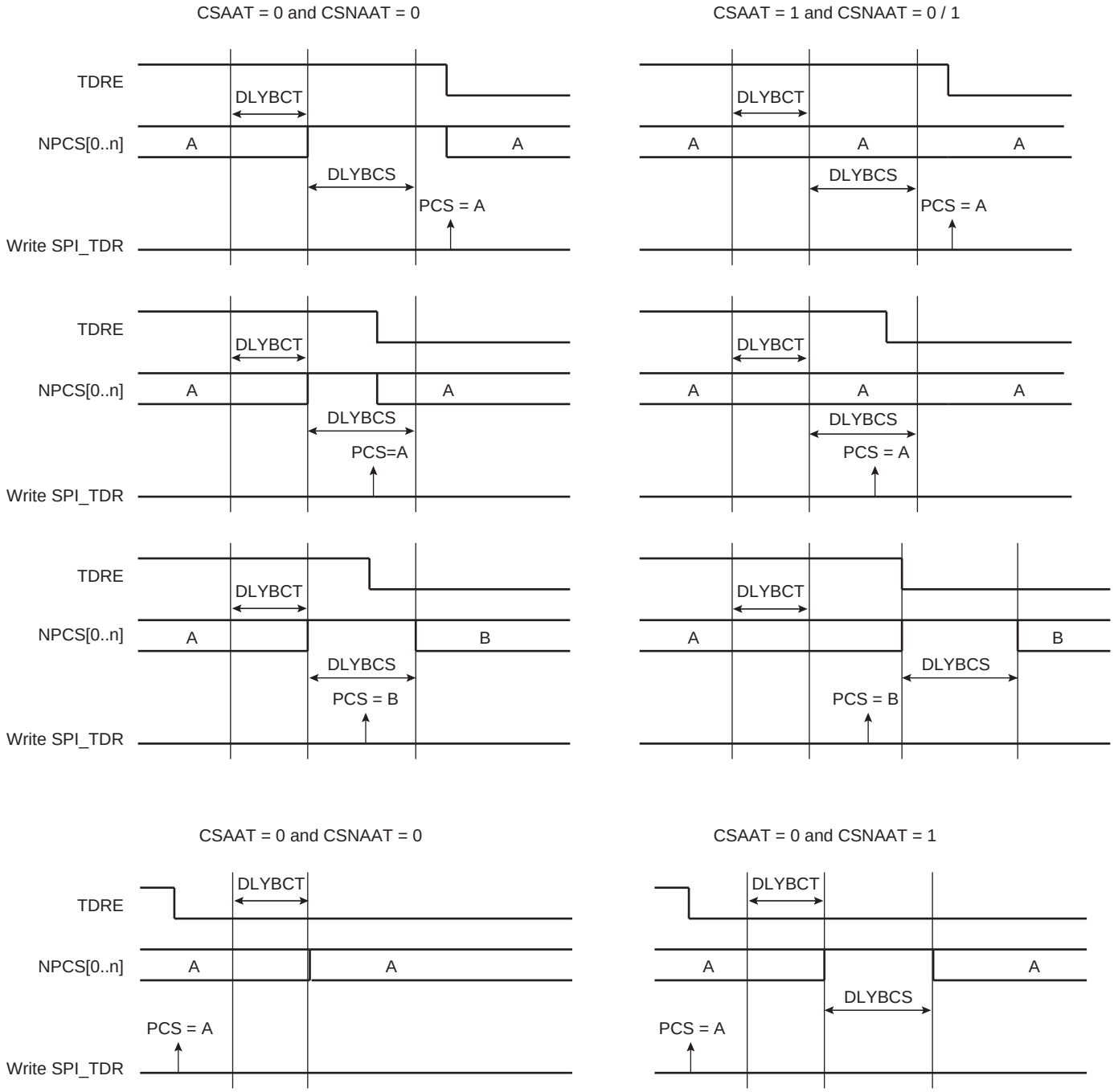
Figure 31-12 shows different peripheral deselection cases and the effect of the CSAAT bit.

When the CSAAT bit is set to 0, the NPCS does not rise in all cases between two transfers on the same peripheral. During a transfer on a Chip Select, the TDRE flag rises as soon as the content of SPI_TDR is transferred into the internal shift register. When this flag is detected, the SPI_TDR can be reloaded. If this reload occurs before the end of the current transfer and if the next transfer is performed on the same chip select as the current transfer, the Chip Select is not de-asserted between the two transfers. This can lead to difficulties to interface with some serial peripherals requiring the chip select to be de-asserted after each transfer. To facilitate interfacing with such devices, the SPI_CSR can be programmed with the Chip Select Not Active After Transfer (CSNAAT) bit at 1. This

allows the chip select lines to be de-asserted systematically during a time “DLYBCS” (the value of the CSNAAT bit is taken into account only if the CSAAT bit is set to 0 for the same chip select).

Figure 31-12 shows different peripheral deselection cases and the effect of the CSAAT and CSNAAT bits.

Figure 31-12. Peripheral Deselection



31.7.3.10 Mode Fault Detection

The SPI has the capability to operate in multi-master environment. Consequently, the NPCS0/NSS line must be monitored. If one of the masters on the SPI bus is currently transmitting, the NPCS0/NSS line is low and the SPI must not transmit a data. A mode fault is detected when the SPI is programmed in Master mode and a low level is driven by an external master on the NPCS0/NSS signal. In multi-master environment, NPCS0, MOSI, MISO and

SPCK pins must be configured in open drain (through the PIO controller). When a mode fault is detected, the MODF bit in SPI_SR is set until SPI_SR is read and the SPI is automatically disabled until it is re-enabled by setting the SPIEN bit in SPI_CR.

By default, the mode fault detection is enabled. The user can disable it by setting the MODFDIS bit in SPI_MR.

31.7.4 SPI Slave Mode

When operating in Slave mode, the SPI processes data bits on the clock provided on the SPI clock pin (SPCK).

The SPI waits until NSS goes active before receiving the serial clock from an external master. When NSS falls, the clock is validated and the data is loaded in SPI_RDR depending on the BITS field configured in SPI_CSR0. These bits are processed following a phase and a polarity defined respectively by the NCPHA and CPOL bits in SPI_CSR0. Note that 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.

Note: For more information on the BITS field, see also the note below the SPI_CSRx bitmap in Section 31.8.9 "SPI Chip Select Register".)

When all bits are processed, the received data is transferred in SPI_RDR and the RDRF bit rises. If SPI_RDR 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 must read SPI_SR to clear the OVRES bit.

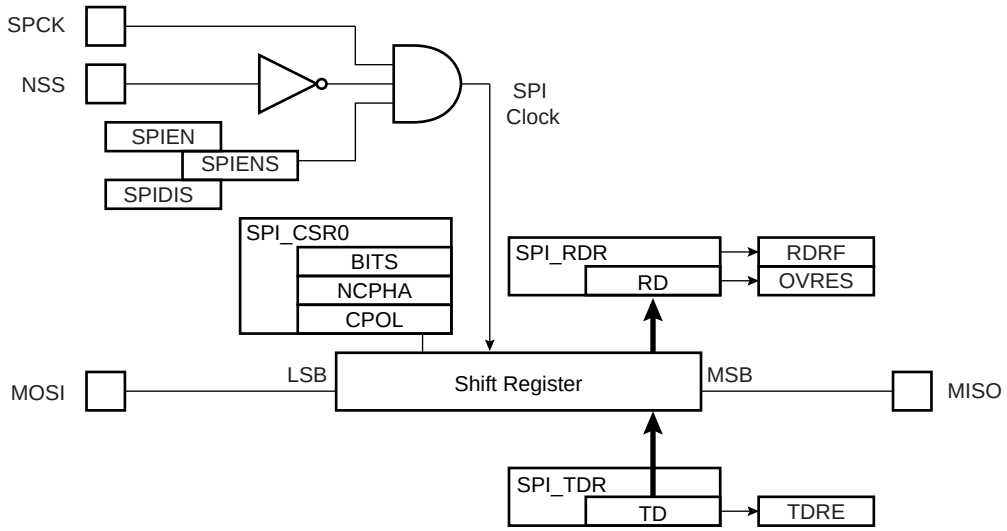
When a transfer starts, the data shifted out is the data present in the Shift register. If no data has been written in SPI_TDR, the last data received is transferred. If no data has been received since the last reset, all bits are transmitted low, as the Shift register resets to 0.

When a first data is written in SPI_TDR, it is transferred immediately in the Shift register and the TDRE flag 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 flag rises. This enables frequent updates of critical variables with single transfers.

Then, a new data is loaded in the Shift register from SPI_TDR. If 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 31-13 shows a block diagram of the SPI when operating in Slave mode.

Figure 31-13. Slave Mode Functional Block Diagram



31.7.5 SPI Comparison on Received Character

The SPI comparison feature is only relevant in SPI Slave mode (MSTR = 0 in SPI_MR).

The effect of a comparison match changes if the system is in Wait or Active mode.

In Wait mode, if asynchronous partial wakeup is enabled, a system wakeup is performed (see Section 31.7.6 "SPI Asynchronous and Partial Wakeup (SleepWalking)").

In Active mode, the CMP flag in SPI_SR is raised. It is set when the received character matches the conditions programmed in the SPI Comparison Register (SPI_CMPR). The CMP flag is set as soon as SPI_RDR is loaded with the new received character. The CMP flag is cleared by reading SPI_SR.

SPI_CMPR (see Section 31.8.10 "SPI Comparison Register") can be programmed to provide different comparison methods, as described below:

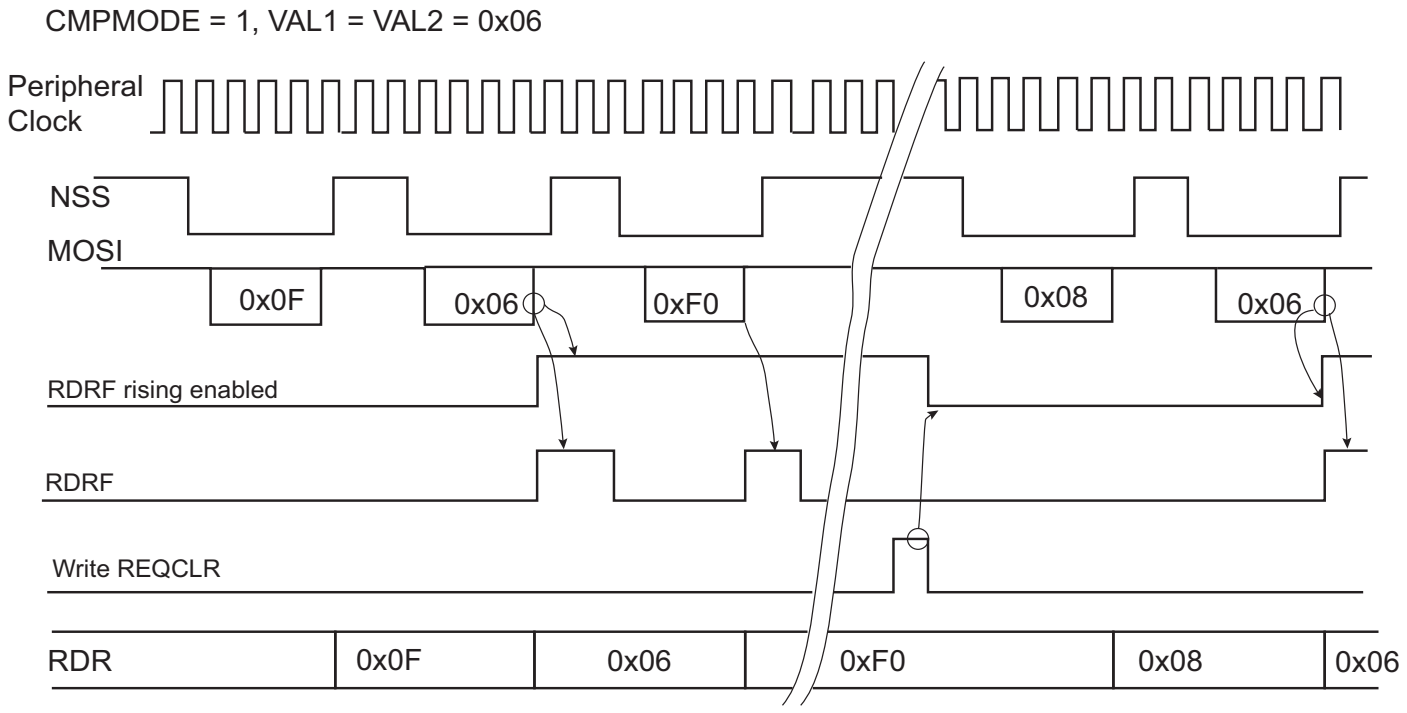
- If VAL1 equals VAL2, then the comparison is performed on a single value and the flag is set to 1 if the received character equals VAL1.
- If VAL1 is strictly lower than VAL2, then any value between VAL1 and VAL2 sets the CMP flag.
- If VAL1 is strictly higher than VAL2, then the CMP flag is set to 1 if any received character equals VAL1 or VAL2.

When the CMPMODE bit is cleared in SPI_CMPR, all received data is loaded in SPI_RDR and the CMP flag provides the status of the comparison result.

By setting the CMPMODE bit, the comparison result triggers the start of the SPI_RDR loading (see Figure 31-14). The trigger condition exists as soon as the received character value matches the conditions defined by VAL1 and VAL2 in SPI_CMPR. The comparison trigger event is restarted by setting the REQCLR bit in SPI_CR if SleepWalking mode is disabled.

The value programmed in VAL1 and VAL2 fields must not exceed the maximum value of the received character (see BITS field in SPI_CSR0).

Figure 31-14. Receive Data Register Management



31.7.6 SPI Asynchronous and Partial Wakeup (SleepWalking)

This operating mode is a means of data pre-processing that qualifies an incoming event, thus allowing the SPI to decide whether or not to wake up the system. Asynchronous and partial wakeup is mainly used when the system is in Wait mode (see the PMC section for further details). It can also be enabled when the system is fully running.

Asynchronous and partial wakeup can be used only when SPI is configured in Slave mode (MSTR is cleared in SPI_MR).

The maximum SPI clock (SPCK) frequency that can be provided by the SPI master is bounded by the peripheral clock frequency. The SPCK frequency must be lower than or equal to the peripheral clock. The NSS line must be de-asserted by the SPI master between two characters. The NSS de-assertion duration time must be greater than or equal to six peripheral clock periods. The time between the assertion of NSS line (falling edge) and the first edge of the SPI clock must be higher than 5 µs.

The SPI_RDR must be read before enabling the asynchronous and partial wakeup.

When asynchronous and partial wakeup is enabled for the SPI (see the PMC section), the PMC decodes a clock request from the SPI. The request is generated as soon as there is a falling edge on the NSS line as this may indicate the beginning of a frame. If the system is in Wait mode (processor and peripheral clocks switched off), the PMC restarts the fast RC oscillator and provides the clock only to the SPI.

The SPI processes the received frame and compares the received character with VAL1 and VAL2 in SPI_CMPR (Section 31.8.10 ”SPI Comparison Register”).

The SPI instructs the PMC to disable the peripheral clock if the received character value does not meet the conditions defined by the VAL1 and VAL2 fields in SPI_CMPR (see Figure 31-16 ”Asynchronous Event Generating Only Partial Wakeup”).

If the received character value meets these conditions, the SPI instructs the PMC to exit the system from Wait mode (see Figure 31-15 ”Asynchronous Wakeup Use Case Example”).

The VAL1 and VAL2 fields can be programmed to provide different comparison methods and thus matching conditions.

- If VAL1 equals VAL2, then the comparison is performed on a single value and the wakeup is triggered if the received character equals VAL1.
- If VAL1 is strictly lower than VAL2, then any value between VAL1 and VAL2 wakes up the system.
- If VAL1 is strictly higher than VAL2, the wakeup is triggered if any received character equals VAL1 or VAL2.
- If VAL1 = 0 and VAL2 = 65535, the wakeup is triggered as soon as a character is received.

If the processor and peripherals are running, the SPI can be configured in asynchronous and partial wakeup mode by enabling the PMC_SLPWK_ER (see PMC section). When activity is detected on the receive line, the SPI requests the clock from the PMC and the comparison is performed. If there is a comparison match, the SPI continues to request the clock. If there is no match, the clock is switched off for the SPI only, until a new activity is detected.

The CMPMODE configuration has no effect when asynchronous and partial wakeup mode is enabled for the SPI (see PMC_SLPWK_ER in PMC section).

When the system is in Active mode and the SPI enters asynchronous and partial wakeup mode, the RDRF flag must be programmed as the unique source of the SPI interrupt.

When the system exits Wait mode as the result of a matching condition, the RDRF flag is used to determine if the SPI is the source for the exit from Wait mode.

Figure 31-15. Asynchronous Wakeup Use Case Example

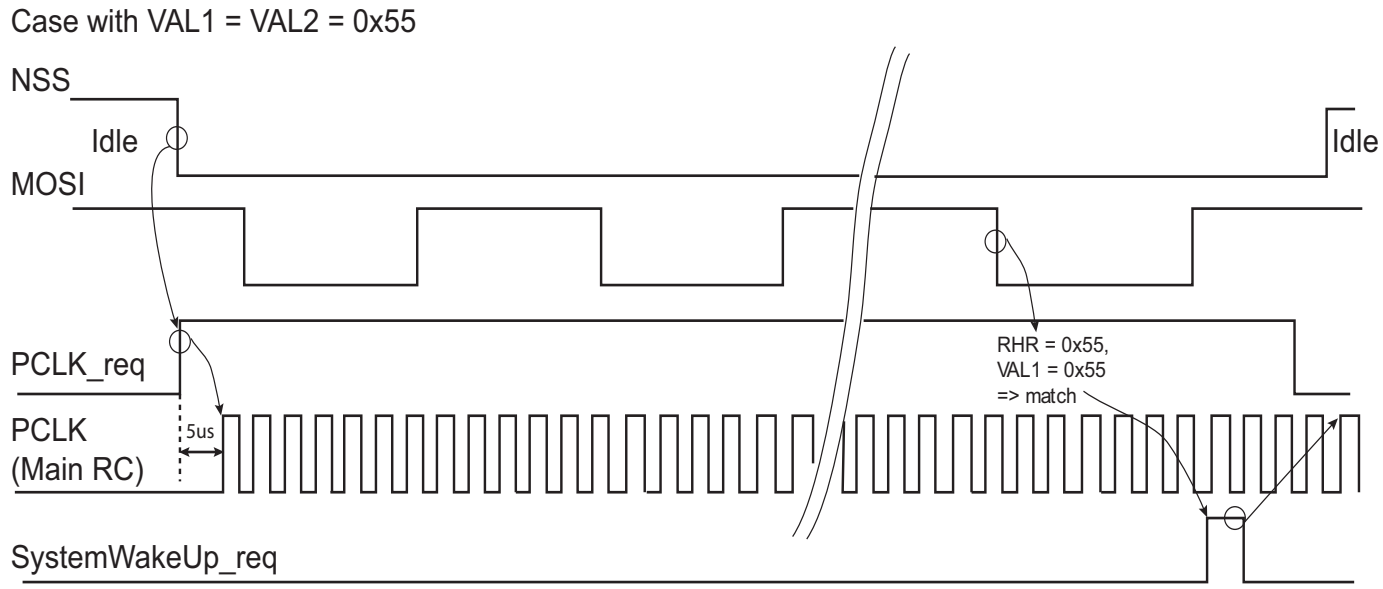
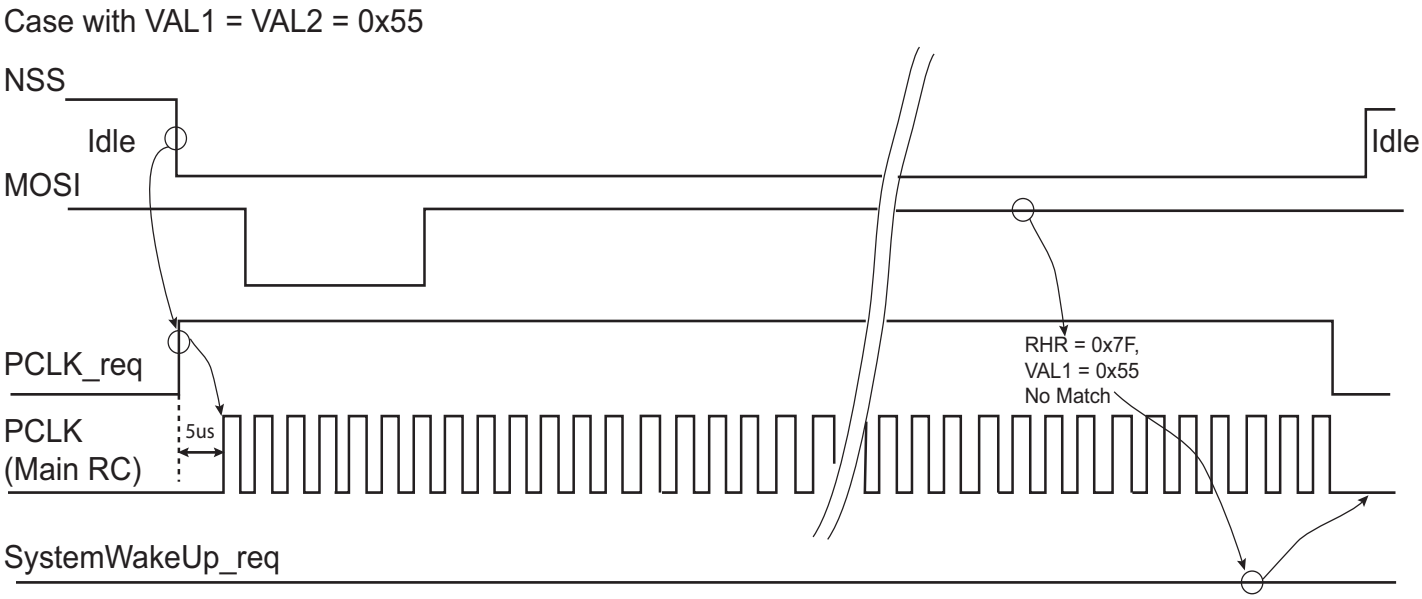


Figure 31-16. Asynchronous Event Generating Only Partial Wakeup



31.7.7 SPI Register Write Protection

To prevent any single software error from corrupting SPI behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the SPI Write Protection Mode Register (SPI_WPMR).

If a write access to a write-protected register is detected, the WPVS flag in the SPI Write Protection Status Register (SPI_WPSR) is set and the WPVSRC field indicates the register in which the write access has been attempted.

The WPVS bit is automatically cleared after reading SPI_WPSR.

The following registers can be write-protected:

- SPI Mode Register

- SPI Chip Select Register
- SPI Comparison Register

31.8 Serial Peripheral Interface (SPI) User Interface

Table 31-5. Register Mapping

Offset	Register	Name	Access	Reset
0x000	SPI Control Register	SPI_CR	Write-only	–
0x004	SPI Mode Register	SPI_MR	Read/Write	0x0
0x008	SPI Receive Data Register	SPI_RDR	Read-only	0x0
0x00C	SPI Transmit Data Register	SPI_TDR	Write-only	–
0x010	SPI Status Register	SPI_SR	Read-only	0x000000F0
0x014	SPI Interrupt Enable Register	SPI_IER	Write-only	–
0x018	SPI Interrupt Disable Register	SPI_IDR	Write-only	–
0x01C	SPI Interrupt Mask Register	SPI_IMR	Read-only	0x0
0x020–0x02C	Reserved	–	–	–
0x030	SPI Chip Select Register 0	SPI_CSR0	Read/Write	0x0
0x034	SPI Chip Select Register 1	SPI_CSR1	Read/Write	0x0
0x038–0x03C	Reserved	–	–	–
0x040–0x044	Reserved	–	–	–
0x048	SPI Comparison Register	SPI_CMPR	Read/Write	0x0
0x04C–0x0E0	Reserved	–	–	–
0x0E4	SPI Write Protection Mode Register	SPI_WPMR	Read/Write	0x0
0x0E8	SPI Write Protection Status Register	SPI_WPSR	Read-only	0x0
0x0EC–0x0F8	Reserved	–	–	–

31.8.1 SPI Control Register

Name: SPI_CR

Address: 0x4000C400 (0), 0x40020400 (1), 0x40024400 (2), 0x40018400 (3), 0x4001C400 (4), 0x40008400 (5), 0x40040400 (6), 0x40034400 (7)

Access: Write-only

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

- **SPIEN: SPI Enable**

0: No effect.

1: Enables the SPI to transfer and receive data.

- **SPIDIS: SPI Disable**

0: No effect.

1: Disables the SPI.

If a transfer is in progress when SPIDIS is set, the SPI completes the transmission of the shift register and does not start any new transfer, even if the SPI_THR is loaded.

All pins are set in Input mode after completion of the transmission in progress, if any.

- **SWRST: SPI Software Reset**

0: No effect.

1: Reset the SPI. A software-triggered hardware reset of the SPI interface is performed.

The SPI is in Slave mode after software reset.

PDC channels are not affected by software reset.

- **REQCLR: Request to Clear the Comparison Trigger**

SleepWalking enabled:

0: No effect.

1: Clears the potential clock request currently issued by the SPI, thus the potential system wakeup is cancelled.

SleepWalking disabled:

0: No effect.

1: Restarts the comparison trigger to enable SPI_RDR loading.

- **LASTXFER: Last Transfer**

0: No effect.

1: The current NPCS will be de-asserted after the character written in TD has been transferred. When CSAAT is set, the communication with the current serial peripheral can be closed by raising the corresponding NPCS line as soon as TD transfer is completed.

Refer to Section 31.7.3.5 ”Peripheral Selection”for more details.

31.8.2 SPI Mode Register

Name: SPI_MR

Address: 0x4000C404 (0), 0x40020404 (1), 0x40024404 (2), 0x40018404 (3), 0x4001C404 (4), 0x40008404 (5), 0x40040404 (6), 0x40034404 (7)

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
–	–	–	CMPMODE	–	–	–	–
7	6	5	4	3	2	1	0
LLB	–	WDRBT	MODFDIS	BRSRCCLK	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 two NPCS chip select lines are connected to a 2- to 4-bit decoder.

When PCSDEC equals one, up to 3 Chip Select signals can be generated with the two NPCS lines using an external 2- to 4-bit decoder. The Chip Select registers define the characteristics of the 3 chip selects, with the following rules:

SPI_CSR0 defines peripheral chip select signals 0 to 1.

SPI_CSR1 defines peripheral chip select signal 2.

- **BRSRCCLK: Bit Rate Source Clock**

0 (PERIPH_CLK): The peripheral clock is the source clock for the bit rate generation.

1 (PMC_PCK): PMC PCKx is the source clock for the bit rate generation, thus the bit rate can be independent of the core/peripheral clock.

Note: If the bit BRSRCCLK = 1, the CSAAT bit must be cleared in SPI_CSRx and the SCBR field must be programmed with a value greater than 1.

- **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 regardless of the SPI_RDR state.

1: In Master mode, a transfer can start only if the SPI_RDR is empty, i.e., does not contain any unread data. This mode prevents overrun error in reception.

- **CMPMODE: Comparison Mode**

Value	Name	Description
0	FLAG_ONLY	Any character is received and comparison function drives CMP flag.
1	START_CONDITION	Comparison condition must be met to start reception of all incoming characters until REQCLR is set.

- **LLB: Local Loopback Enable**

0: Local loopback path disabled.

1: Local loopback path enabled.

LLB controls the local loopback on the data shift register for testing in Master mode only (MISO is internally connected on MOSI).

- **PCS: Peripheral Chip Select**

This field is only used if fixed peripheral select is active (PS = 0).

If PCSDEC = 0:

PCS = x0 NPCS[1:0] = 10

PCS = 01 NPCS[1:0] = 01

PCS = 11 forbidden (no peripheral is selected)

(x = don't care)

If PCSDEC = 1:

NPCS[1:0] output signals = PCS

- **DLYBCS: Delay Between Chip Selects**

This field defines the delay between the inactivation and the activation of NPCS. The DLYBCS time guarantees non-overlapping chip selects and solves bus contentions in case of peripherals having long data float times.

If DLYBCS is less than or equal to six, six peripheral clock periods are inserted by default.

Otherwise, the following equations determine the delay:

if SPI_MR.BRSRCCLK = 0:

$$\text{Delay Between Chip Selects} = \frac{\text{DLYBCS}}{f_{\text{peripheral clock}}}$$

if SPI_MR.BRSRCCLK = 1:

$$\text{Delay Between Chip Selects} = \frac{\text{DLYBCS}}{f_{\text{PCKx}}}$$

31.8.3 SPI Receive Data Register

Name: SPI_RDR

Address: 0x4000C408 (0), 0x40020408 (1), 0x40024408 (2), 0x40018408 (3), 0x4001C408 (4), 0x40008408 (5), 0x40040408 (6), 0x40034408 (7)

Access: Read-only

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

- **RD: Receive Data**

Data received by the SPI Interface is stored in this register in a right-justified format. Unused bits are read as zero.

- **PCS: Peripheral Chip Select**

In Master mode only, these bits indicate the value on the NPCS pins at the end of a transfer. Otherwise, these bits are read as zero.

Note: When using Variable peripheral select mode (PS = 1 in SPI_MR), it is mandatory to set the WDRBT field to 1 if the PCS field must be processed in SPI_RDR.

31.8.4 SPI Transmit Data Register

Name: SPI_TDR

Address: 0x4000C40C (0), 0x4002040C (1), 0x4002440C (2), 0x4001840C (3), 0x4001C40C (4), 0x4000840C (5), 0x4004040C (6), 0x4003440C (7)

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 = x0 NPCS[1:0] = 10

PCS = 01 NPCS[1:0] = 01

PCS = 11 forbidden (no peripheral is selected)

(x = don’t care)

If PCSDEC = 1:

NPCS[1:0] output signals = PCS

- **LASTXFER: Last Transfer**

0: No effect.

1: The current NPCS is de-asserted after the transfer of the character written in TD. When CSAAT is set, the communication with the current serial peripheral can be closed by raising the corresponding NPCS line as soon as TD transfer is completed.

This field is only used if variable peripheral select is active (PS = 1).

31.8.5 SPI Status Register

Name: SPI_SR

Address: 0x4000C410 (0), 0x40020410 (1), 0x40024410 (2), 0x40018410 (3), 0x4001C410 (4), 0x40008410 (5), 0x40040410 (6), 0x40034410 (7)

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
–	–	–	–	CMP	UNDES	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

- **RDRF: Receive Data Register Full (cleared by reading SPI_RDR)**

0: No data has been received since the last read of SPI_RDR.

1: Data has been received and the received data has been transferred from the shift register to SPI_RDR since the last read of SPI_RDR.

- **TDRE: Transmit Data Register Empty (cleared by writing SPI_TDR)**

0: Data has been written to SPI_TDR and not yet transferred to the shift register.

1: The last data written in the SPI_TDR has been transferred to the shift register.

TDRE equals zero when the SPI is disabled or at reset. The SPI enable command sets this bit to 1.

- **MODF: Mode Fault Error (cleared on read)**

0: No mode fault has been detected since the last read of SPI_SR.

1: A mode fault occurred since the last read of SPI_SR.

- **OVRES: Overrun Error Status (cleared on read)**

0: No overrun has been detected since the last read of SPI_SR.

1: An overrun has occurred since the last read of SPI_SR.

An overrun occurs when SPI_RDR is loaded at least twice from the shift register since the last read of the SPI_RDR.

- **ENDRX: End of RX buffer (cleared by writing SPI_RCR or SPI_RNCR)**

0: The Receive Counter register has not reached 0 since the last write in SPI_RCR(1) or SPI_RNCR(1).

1: The Receive Counter register has reached 0 since the last write in SPI_RCR(1) or SPI_RNCR(1).

- **ENDTX: End of TX buffer (cleared by writing SPI_TCR or SPI_TNCR)**

0: The Transmit Counter register has not reached 0 since the last write in SPI_TCR(1) or SPI_TNCR(1).

1: The Transmit Counter register has reached 0 since the last write in SPI_TCR(1) or SPI_TNCR(1).

- **RXBUFF: RX Buffer Full (cleared by writing SPI_RCR or SPI_RNCR)**

0: SPI_RCR(1) or SPI_RNCR(1) has a value other than 0.

1: Both SPI_RCR(1) and SPI_RNCR(1) have a value of 0.

- **TXBUFE: TX Buffer Empty (cleared by writing SPI_TCR or SPI_TNCR)**

0: SPI_TCR(1) or SPI_TNCR(1) has a value other than 0.

1: Both SPI_TCR(1) and SPI_TNCR(1) have a value of 0.

- **NSSR: NSS Rising (cleared on read)**

0: No rising edge detected on NSS pin since the last read of SPI_SR.

1: A rising edge occurred on NSS pin since the last read of SPI_SR.

- **TXEMPTY: Transmission Registers Empty (cleared by writing SPI_TDR)**

0: As soon as data is written in SPI_TDR.

1: SPI_TDR and internal shift register are empty. If a transfer delay has been defined, TXEMPTY is set after the end of this delay.

- **UNDES: Underrun Error Status (slave mode only) (cleared on read)**

0: No underrun has been detected since the last read of SPI_SR.

1: A transfer started whereas no data has been loaded in SPI_TDR, occurred since the last read of SPI_SR.

- **SPIENS: SPI Enable Status**

0: SPI is disabled.

1: SPI is enabled.

- **CMP: Comparison Status (cleared on read)**

0: No received character matched the comparison criteria programmed in the VAL1 and VAL2 fields in SPI_CMPR since the last SPI_SR read.

1: A received character matched the comparison criteria since the last SPI_SR read.

Note: 1. SPI_RCR, SPI_RNCR, SPI_TCR, SPI_TNCR are PDC registers.

31.8.6 SPI Interrupt Enable Register

Name: SPI_IER

Address: 0x4000C414 (0), 0x40020414 (1), 0x40024414 (2), 0x40018414 (3), 0x4001C414 (4), 0x40008414 (5), 0x40040414 (6), 0x40034414 (7)

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
–	–	–	–	CMP	UNDES	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **RDRF: Receive Data Register Full Interrupt Enable**
- **TDRE: SPI Transmit Data Register Empty Interrupt Enable**
- **MODF: Mode Fault Error Interrupt Enable**
- **OVRES: Overrun Error Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable**
- **ENDTX: End of Transmit Buffer Interrupt Enable**
- **RXBUFF: Receive Buffer Full Interrupt Enable**
- **TXBUFE: Transmit Buffer Empty Interrupt Enable**
- **NSSR: NSS Rising Interrupt Enable**
- **TXEMPTY: Transmission Registers Empty Enable**
- **UNDES: Underrun Error Interrupt Enable**
- **CMP: Comparison Interrupt Enable**

31.8.7 SPI Interrupt Disable Register

Name: SPI_IDR

Address: 0x4000C418 (0), 0x40020418 (1), 0x40024418 (2), 0x40018418 (3), 0x4001C418 (4), 0x40008418 (5), 0x40040418 (6), 0x40034418 (7)

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
–	–	–	–	CMP	UNDES	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Disables the corresponding interrupt.

- **RDRF: Receive Data Register Full Interrupt Disable**
- **TDRE: SPI Transmit Data Register Empty Interrupt Disable**
- **MODF: Mode Fault Error Interrupt Disable**
- **OVRES: Overrun Error Interrupt Disable**
- **ENDRX: End of Receive Buffer Interrupt Disable**
- **ENDTX: End of Transmit Buffer Interrupt Disable**
- **RXBUFF: Receive Buffer Full Interrupt Disable**
- **TXBUFE: Transmit Buffer Empty Interrupt Disable**
- **NSSR: NSS Rising Interrupt Disable**
- **TXEMPTY: Transmission Registers Empty Disable**
- **UNDES: Underrun Error Interrupt Disable**
- **CMP: Comparison Interrupt Disable**

31.8.8 SPI Interrupt Mask Register

Name: SPI_IMR

Address: 0x4000C41C (0), 0x4002041C (1), 0x4002441C (2), 0x4001841C (3), 0x4001C41C (4), 0x4000841C (5), 0x4004041C (6), 0x4003441C (7)

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
–	–	–	–	CMP	UNDES	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

- **RDRF: Receive Data Register Full Interrupt Mask**
- **TDRE: SPI Transmit Data Register Empty Interrupt Mask**
- **MODF: Mode Fault Error Interrupt Mask**
- **OVRES: Overrun Error Interrupt Mask**
- **ENDRX: End of Receive Buffer Interrupt Mask**
- **ENDTX: End of Transmit Buffer Interrupt Mask**
- **RXBUFF: Receive Buffer Full Interrupt Mask**
- **TXBUFE: Transmit Buffer Empty Interrupt Mask**
- **NSSR: NSS Rising Interrupt Mask**
- **TXEMPTY: Transmission Registers Empty Mask**
- **UNDES: Underrun Error Interrupt Mask**
- **CMP: Comparison Interrupt Mask**

31.8.9 SPI Chip Select Register

Name: SPI_CSRx[x=0..1]

Address: 0x4000C430 (0), 0x40020430 (1), 0x40024430 (2), 0x40018430 (3), 0x4001C430 (4), 0x40008430 (5), 0x40040430 (6), 0x40034430 (7)

Access: Read/Write

31	30	29	28	27	26	25	24
DLYBCT							
23	22	21	20	19	18	17	16
DLYBS							
15	14	13	12	11	10	9	8
SCBR							
7	6	5	4	3	2	1	0
BITS				CSAAT	CSNAAT	NCPHA	CPOL

This register can only be written if the WPEN bit is cleared in the "SPI Write Protection Mode Register".

Note: SPI_CSRx registers must be written even if the user wants to use the default reset values. The BITS field is not updated with the translated value unless the register is written.

- **CPOL: Clock Polarity**

0: The inactive state value of SPCK is logic level zero.

1: The inactive state value of SPCK is logic level one.

CPOL is used to determine the inactive state value of the serial clock (SPCK). It is used with NCPHA to produce the required clock/data relationship between master and slave devices.

- **NCPHA: Clock Phase**

0: Data is changed on the leading edge of SPCK and captured on the following edge of SPCK.

1: Data is captured on the leading edge of SPCK and changed on the following edge of SPCK.

NCPHA determines which edge of SPCK causes data to change and which edge causes data to be captured. NCPHA is used with CPOL to produce the required clock/data relationship between master and slave devices.

- **CSNAAT: Chip Select Not Active After Transfer (Ignored if CSAAT = 1)**

0: The Peripheral Chip Select 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 inactive after the end of transfer for a minimal duration of:

If SPI_MR.BRSRCCLK = 0:

- $\frac{DLYBCS}{f_{peripheral\ clock}}$ (if DLYBCS field is different from 0)

If SPI_MR.BRSRCCLK = 1:

- $\frac{DLYBCS}{f_{PCKx}}$

If field DLYBCS is lower than 6, a minimum of six periods is introduced.

- **CSAAT: Chip Select Active After Transfer**

0: The Peripheral Chip Select Line rises as soon as the last transfer is achieved.

1: The Peripheral Chip Select does not rise after the last transfer is achieved. It remains active until a new transfer is requested on a different chip select.

Note: If the bit BRSRCCLK = 1, the CSAAT bit must be cleared.

- **BITS: Bits Per Transfer**

(See the note below the SPI_CSRx bitmap.)

The BITS field determines the number of data bits transferred.

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

- **SCBR: Serial Clock Bit Rate**

In Master mode, the SPI Interface uses a modulus counter to derive the SPCK bit rate from the clock defined by the bit BRSRCCLK. The bit rate is selected by writing a value from 1 to 255 in the SCBR field. The following equations determine the SPCK bit rate:

if SPI_MR.BRSRCCLK = 0:

$$\text{SPCK Bit Rate} = \frac{f_{\text{peripheral clock}}}{\text{SCBR}}$$

if SPI_MR.BRSRCCLK = 1:

$$\text{SPCK Bit Rate} = \frac{f_{\text{PCKx}}}{\text{SCBR}}$$

Programming the SCBR field to 0 is forbidden. Triggering a transfer while SCBR is at 0 can lead to unpredictable results.

If BRSRCCLK = 1 in SPI_MR, SCBR must be programmed with a value greater than 1.

At reset, SCBR is 0 and the user has to program it at a valid value before performing the first transfer.

Note: If one of the SCBR fields in SPI_CSRx is set to 1, the other SCBR fields in SPI_CSRx must be set to 1 as well if they are used to process transfers. If they are not used to transfer data, they can be set at any value.

- **DLYBS: Delay Before SPCK**

This field defines the delay from NPCS falling edge (activation) to the first valid SPCK transition.

When DLYBS field value equals zero, this delay is half the SPCK clock period.

Otherwise, the following equations determine the delay:

if SPI_MR.BRSRCCLK = 0:

$$\text{Delay Before SPCK} = \frac{\text{DLYBS}}{f_{\text{peripheral clock}}}$$

if SPI_MR.BRSRCCLK = 1:

$$\text{Delay Before SPCK} = \frac{\text{DLYBS}}{f_{PCKx}}$$

- **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 equations determine the delay:

if SPI_MR.BRSRCCLK = 0:

$$\text{Delay Between Consecutive Transfers} = \frac{32 \times \text{DLYBCT}}{f_{\text{peripheral clock}}}$$

if SPI_MR.BRSRCCLK = 1:

$$\text{Delay Between Consecutive Transfers} = \frac{32 \times \text{DLYBCT}}{f_{PCKx}}$$

31.8.10 SPI Comparison Register

Name: SPI_CMPR

Address: 0x4000C448 (0), 0x40020448 (1), 0x40024448 (2), 0x40018448 (3), 0x4001C448 (4), 0x40008448 (5), 0x40040448 (6), 0x40034448 (7)

Access: Read/Write

31	30	29	28	27	26	25	24
VAL2							
23	22	21	20	19	18	17	16
VAL2							
15	14	13	12	11	10	9	8
VAL1							
7	6	5	4	3	2	1	0
VAL1							

This register can only be written if the WPEN bit is cleared in ”SPI Write Protection Mode Register”.

- **VAL1: First Comparison Value for Received Character**

0–65535

- **VAL2: Second Comparison Value for Received Character**

0–65535

31.8.11 SPI Write Protection Mode Register

Name: SPI_WPMR

Address: 0x4000C4E4 (0), 0x400204E4 (1), 0x400244E4 (2), 0x400184E4 (3), 0x4001C4E4 (4), 0x400084E4 (5), 0x400404E4 (6), 0x400344E4 (7)

Access: Read/Write

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

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x535049 (“SPI” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x535049 (“SPI” in ASCII)

See Section 31.7.7 ”SPI Register Write Protection” for the list of registers that can be write-protected.

- **WPKEY: Write Protection Key**

Value	Name	Description
0x535049	PASSWD	Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

31.8.12 SPI Write Protection Status Register

Name: SPI_WPSR

Address: 0x4000C4E8 (0), 0x400204E8 (1), 0x400244E8 (2), 0x400184E8 (3), 0x4001C4E8 (4), 0x400084E8 (5), 0x400404E8 (6), 0x400344E8 (7)

Access: Read-only

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

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of SPI_WPSR.

1: A write protection violation has occurred since the last read of SPI_WPSR. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protection Violation Source**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

32. Two-wire Interface (TWI)

32.1 Description

The 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 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(1), 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.

A configurable baud rate generator permits the output data rate to be adapted to a wide range of core clock frequencies.

Note: 1. See Table 32-1 for details on compatibility with I²C Standard.

Table 32-1. Atmel TWI Compatibility with I²C Standard

I²C 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
Repeated Start (Sr) Condition	Supported
ACK and NACK Management	Supported
Input Filtering	Supported
Slope Control	Not supported
Clock Stretching	Supported
Multi Master Capability	Supported

Note: 1. 10-bit support in Master mode only

32.2 Embedded Characteristics

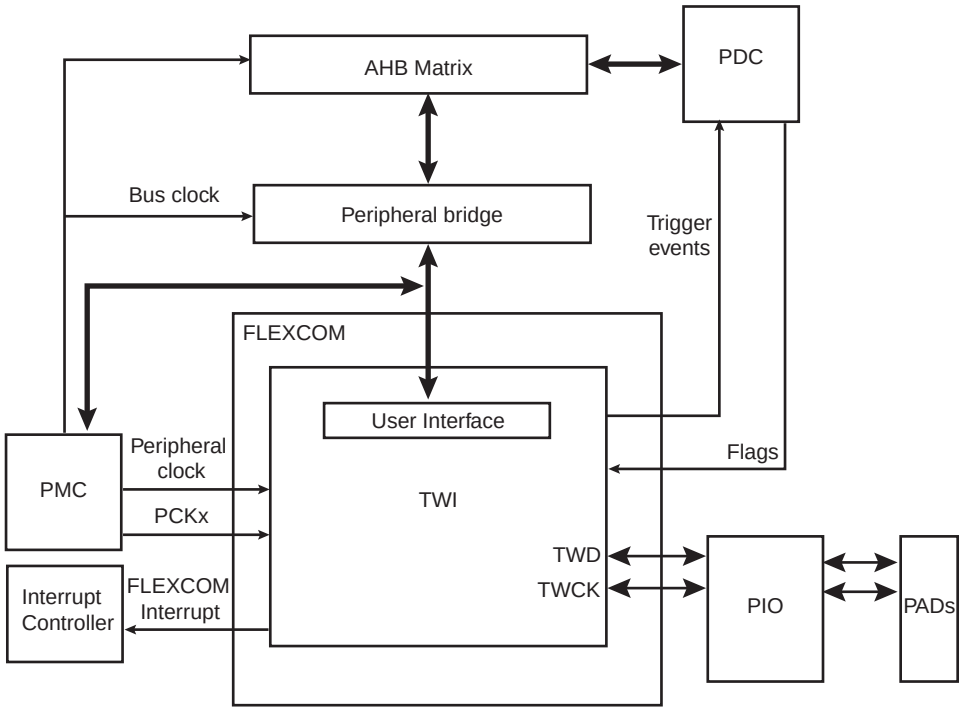
TWI/SMBUS Characteristics

- Bit Rate: Up to 400 kbit/s in Fast Mode and 3.4 Mbit/s in High-Speed Mode (Slave Only)
- Bit Rate can be independent of the Processor/Peripheral Clock
- SMBus support
- Asynchronous Partial Wakeup (SleepWalking) Capability
- Compatible with Atmel Two-wire Interface Serial Memory and I²C Compatible Devices(1)
- Master and Multi-Master Operation (Standard and Fast Mode Only)
- Slave Mode Operation (Standard, Fast and High-Speed Mode)
- One, Two or Three Bytes for Slave Address
- Sequential Read/Write Operations
- General Call Supported in Slave Mode
- Connection to Peripheral DMA Controller (PDC) Channels Optimizes Data Transfers
  - One Channel for the Receiver, One Channel for the Transmitter
- Register Write Protection

Note: 1. See Table 32-1 for details on compatibility with I²C Standard.

32.3 Block Diagram

Figure 32-1. TWI Block Diagram



32.4 I/O Lines Description

Table 32-2. I/O Lines Description

Name	Description	Type
TWD	Two-wire Serial Data (drives external serial data line – SDA)	I/O
TWCK	Two-wire Serial Clock (drives external serial clock line – SCL)	I/O

32.5 Product Dependencies

32.5.1 I/O Lines

The pins used for interfacing the TWI are multiplexed with the USART and SPI lines within FLEXCOM module. The programmer must first program the PIO controller to assign the desired FLEXCOM pins to their peripheral function. If I/O lines of the FLEXCOM are not used by the application, they can be used for other purposes by the PIO Controller.

Both TWD and TWCK are bidirectional lines, connected to a positive supply voltage via a current source or pull-up resistor. When the bus is free, both lines are high. The output stages of devices connected to the bus must have an open-drain or open-collector to perform the wired-AND function.

TWD and TWCK pins may be multiplexed with PIO lines. To enable the TWI, the user must program the PIO Controller to dedicate TWD and TWCK as peripheral lines.

The user must not program TWD and TWCK as open-drain. This is already done by the hardware.

Table 32-3. I/O Lines

Instance	Signal	I/O Line	Peripheral
TWI0	RXD0/SPI0_MISO/TWCK0	PA9	A
TWI0	TXD0/SPI0_MOSI/TWD0	PA10	A
TWI1	RXD1/SPI1_MISO/TWCK1	PB2	A
TWI1	TXD1/SPI1_MOSI/TWD1	PB3	A
TWI2	RXD2/SPI2_MISO/TWCK2	PA5	A
TWI2	TXD2/SPI2_MOSI/TWD2	PA6	A
TWI3	RXD3/SPI3_MISO/TWCK3	PA4	A
TWI3	TXD3/SPI3_MOSI/TWD3	PA3	A
TWI4	RXD4/SPI4_MISO/TWCK4	PB9	A
TWI4	RXD4/SPI4_MISO/TWCK4	PB11	A
TWI4	TXD4/SPI4_MOSI/TWD4	PB8	A
TWI4	TXD4/SPI4_MOSI/TWD4	PB10	A
TWI5	RXD5/SPI5_MISO/TWCK5	PA12	A
TWI5	TXD5/SPI5_MOSI/TWD5	PA13	A
TWI6	RXD6/SPI6_MISO/TWCK6	PB1	B
TWI6	RXD6/SPI6_MISO/TWCK6	PB11	B
TWI6	TXD6/SPI6_MOSI/TWD6	PB0	B
TWI6	TXD6/SPI6_MOSI/TWD6	PB10	B
TWI7	RXD7/SPI7_MISO/TWCK7	PA27	B
TWI7	TXD7/SPI7_MOSI/TWD7	PA28	B

32.5.2 Power Management

The TWI is not continuously clocked. The programmer must first enable the FLEXCOM Clock in the Power Management Controller (PMC) and configure the field OPMODE = 3 in the FLEXCOM Mode Register (FLEX_MR) before using the TWI. If the OPMODE field differs from 3, the TWI clock is stopped.

In SleepWalking mode (asynchronous partial wakeup), the PMC must be configured to enable SleepWalking for the FLEXCOM in the Sleepwalking Enable Register (PMC_SLPWK_ER). Depending on the instructions (requests) provided by the FLEXCOM to the PMC, the system clock may or may not be automatically provided to the FLEXCOM.

32.5.3 Interrupt Sources

The TWI interrupt line is the FLEXCOM interrupt line if the field OPMODE = 3 in FLEX_MR. The FLEXCOM interrupt line is connected on one of the internal sources of the Interrupt Controller. Using the FLEXCOM interrupt requires the Interrupt Controller to be programmed first.

Table 32-4. Peripheral IDs

Instance	ID
TWI0	8
TWI1	9
TWI2	14
TWI3	19
TWI4	20
TWI5	21
TWI6	22
TWI7	7

32.6 TWI Functional Description

32.6.1 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 32-3).

Each transfer begins with a START condition and terminates with a STOP condition (see Figure 32-2).

- A high-to-low transition on the TWD line while TWCK is high defines the START condition.
- A low-to-high transition on the TWD line while TWCK is high defines a STOP condition.

Figure 32-2. START and STOP Conditions

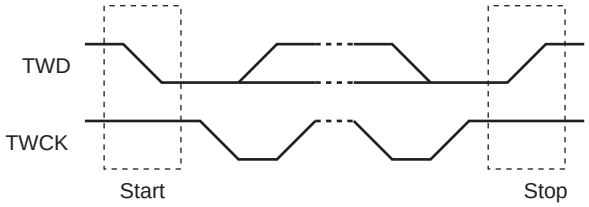
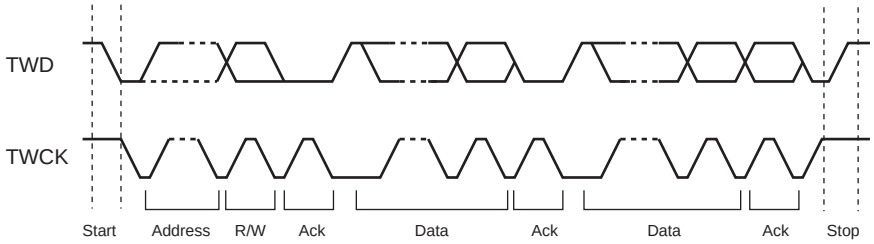


Figure 32-3. Transfer Format



32.6.2 Modes of Operation

The TWI has different modes of operation:

- Master Transmitter mode (Standard and Fast modes only)
- Master Receiver mode (Standard and Fast modes only)
- Multi-master Transmitter mode (Standard and Fast modes only)
- Multi-master Receiver mode (Standard and Fast modes only)
- Slave Transmitter mode (Standard, Fast and High-speed mode)
- Slave Receiver mode (Standard, Fast and High-speed modes)

These modes are described in the following sections.

32.6.3 Master Mode

32.6.3.1 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.

32.6.3.2 Programming Master Mode

The following fields must be programmed before entering Master mode:

1. TWI_MMR.DADR (+ TWI_MMR.IADRSZ + TWI_IADR.IADR if a 10-bit device is addressed): The device address is used to access slave devices in Read or Write mode.
2. In TWI_CWGR, CKDIV + CHDIV + CLDIV: Clock waveform.
3. TWI_CR.SVDIS: Disables the Slave mode.
4. TWI_CR.MSEN: Enables the Master mode.

Note: If the TWI is already in Master mode, the device address (DADR) can be configured without disabling the Master mode.

32.6.3.3 Transfer Speed/ Bit Rate

The TWI speed is defined in the TWI Clock Waveform Generator Register (TWI_CWGR).

The TWI bit rate can be based either on the peripheral clock if the BRSRCCLK bit value is 0 or on a programmable clock source provided by the PMC (PCKx) if the BRSRCCLK bit value is 1.

If BRSRCCLK = 1, the bit rate is independent of the processor/peripheral clock and thus the processor/peripheral clock frequency can be changed without affecting the TWI transfer rate.

The PMC PCKx frequency must be at least three times lower than the peripheral clock frequency.

32.6.3.4 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 TWI 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 (ninth pulse), the master releases the data line (HIGH), enabling the slave to pull it down in order to generate the acknowledge. If the slave does not acknowledge the byte, then the Not Acknowledge flag (NACK) is set in the TWI Status Register (TWI_SR) of the master and a STOP condition is sent. The NACK flag must be cleared by reading TWI_SR before the next write into TWI_THR. As with the other status bits, an interrupt can be generated if enabled in the TWI Interrupt Enable Register (TWI_IER). If the slave acknowledges the byte, the data written in 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 TWI_THR.

TXRDY is used as transmit ready for the PDC transmit channel.

While no new data is written in TWI_THR, the serial clock line is tied low. When new data is written in 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 bit of the TWI Control Register (TWI_CR).

After a master write transfer, the Serial Clock line is stretched (tied low) while no new data is written in TWI_THR or until a STOP command is performed.

See Figure 32-4, Figure 32-5, and Figure 32-6.

Note: In Master mode, the TXRDY flag can be cleared by first setting the bit TWI_CR.MSDIS and then setting the bit TWI_CR.MSEN.

Figure 32-4. Master Write with One Data Byte

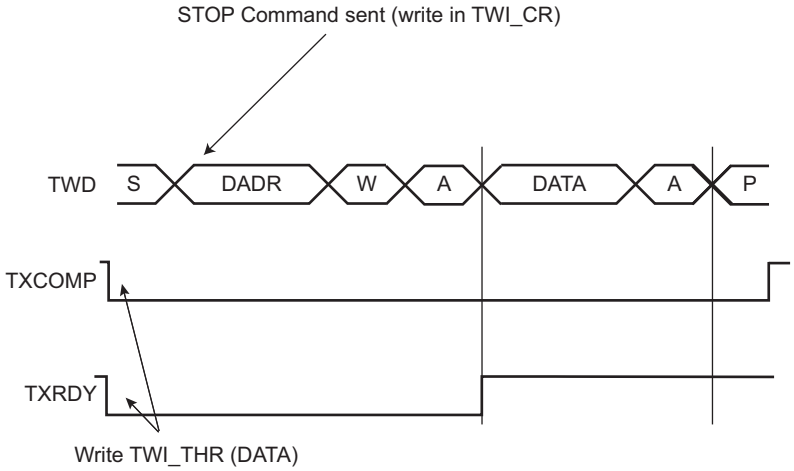


Figure 32-5. Master Write with Multiple Data Bytes

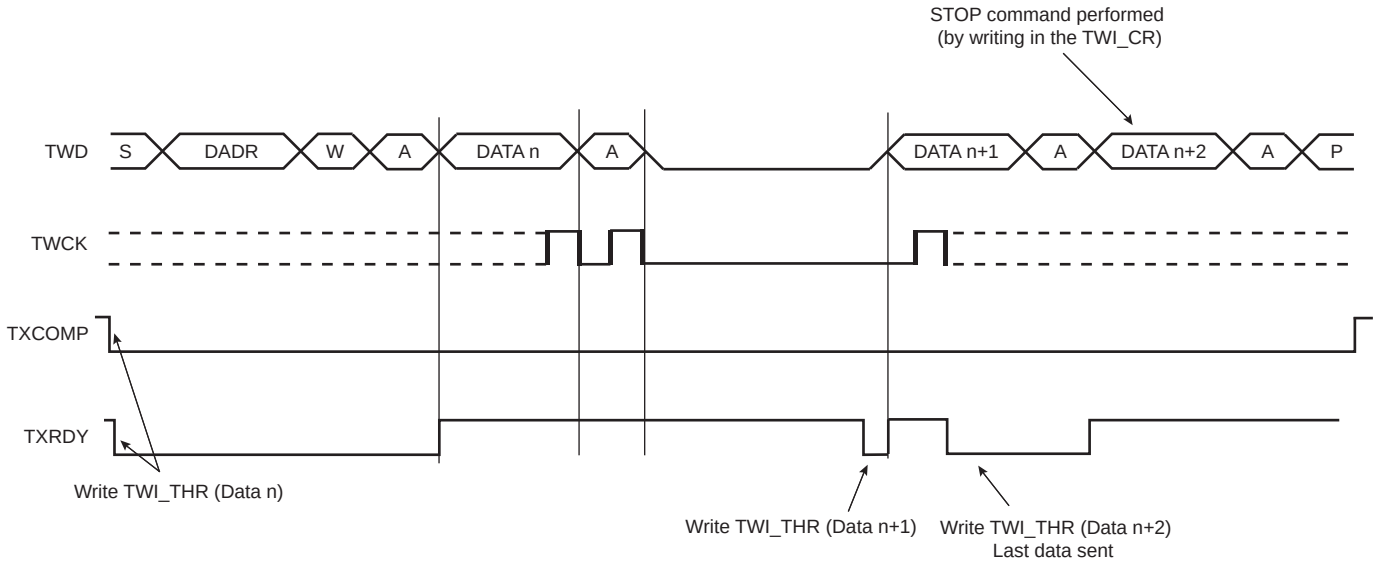
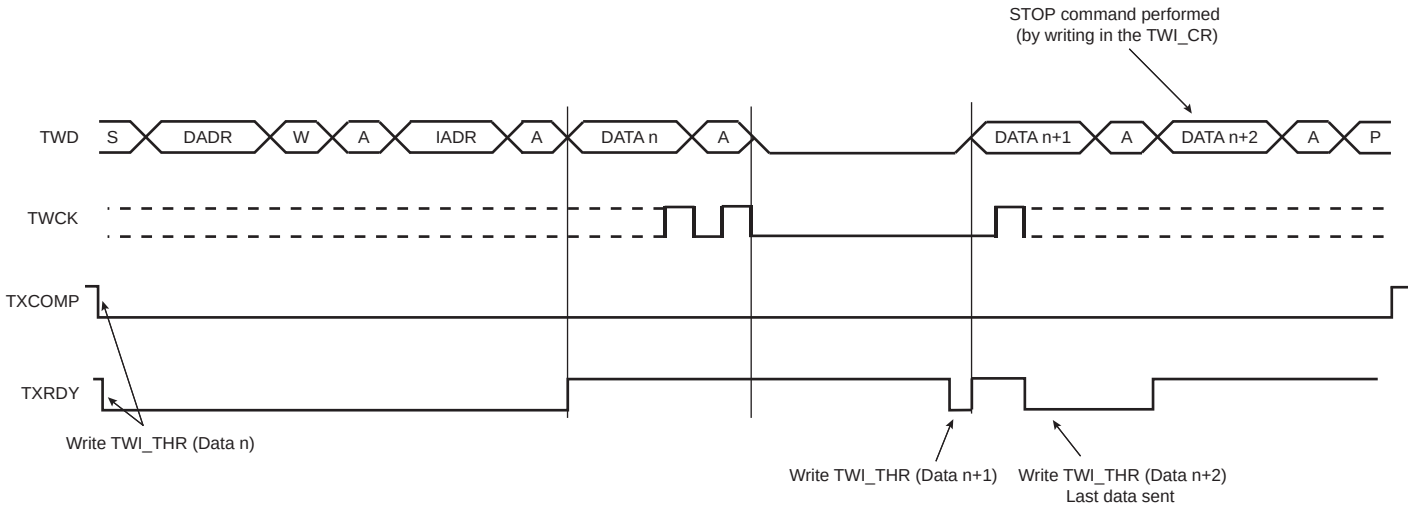


Figure 32-6. Master Write with One Byte Internal Address and Multiple Data Bytes



32.6.3.5 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 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 TWI_SR if the slave does not acknowledge the byte.

If an acknowledge is received, the master is then ready to receive data from the slave. After data has been received, the master sends an acknowledge condition to notify the slave that the data has been received except for the last data (see Figure 32-7). When the RXRDY bit is set in TWI_SR, a character has been received in the TWI Receive-Holding Register (TWI_RHR). The RXRDY bit is reset when reading 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 32-7, "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 32-8, "Master Read with Multiple Data Bytes". For internal address usage, see Section 32.6.3.6 "Internal Address".

If 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 TWI_RHR is read. Once TWI_RHR is read, the master will stop stretching the serial clock line and end the data reception, see Figure 32-9.

Warning: When receiving multiple bytes in Master Read mode, if the next-to-last access is not read (the RXRDY flag remains high), the last access will not be completed until TWI_RHR is read. The last access stops on the next-to-last bit (clock stretching). When TWI_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 TWI_RHR on the next-to-last access (within IT handler).

Figure 32-7. Master Read with One Data Byte

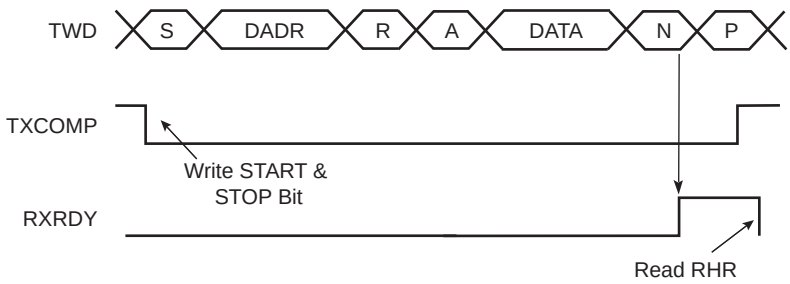


Figure 32-8. Master Read with Multiple Data Bytes

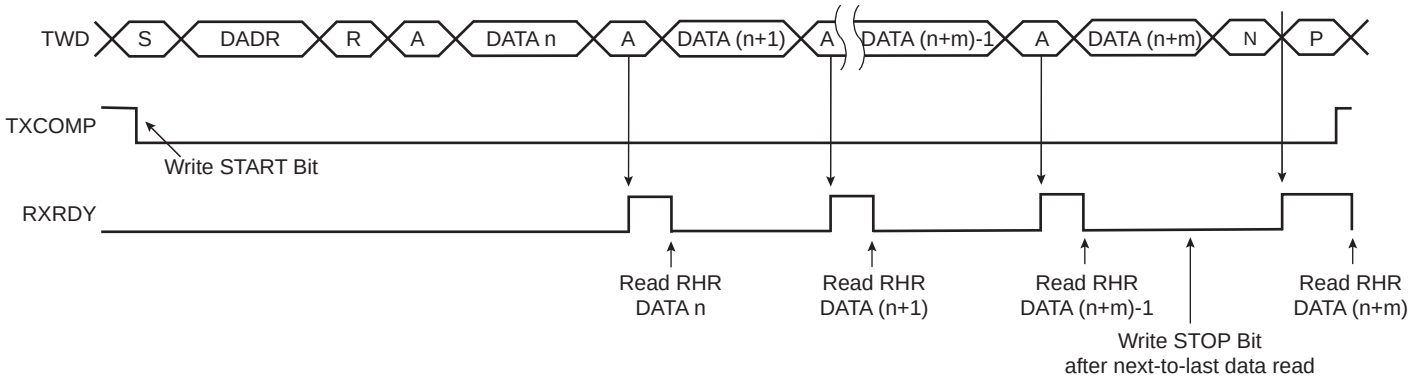
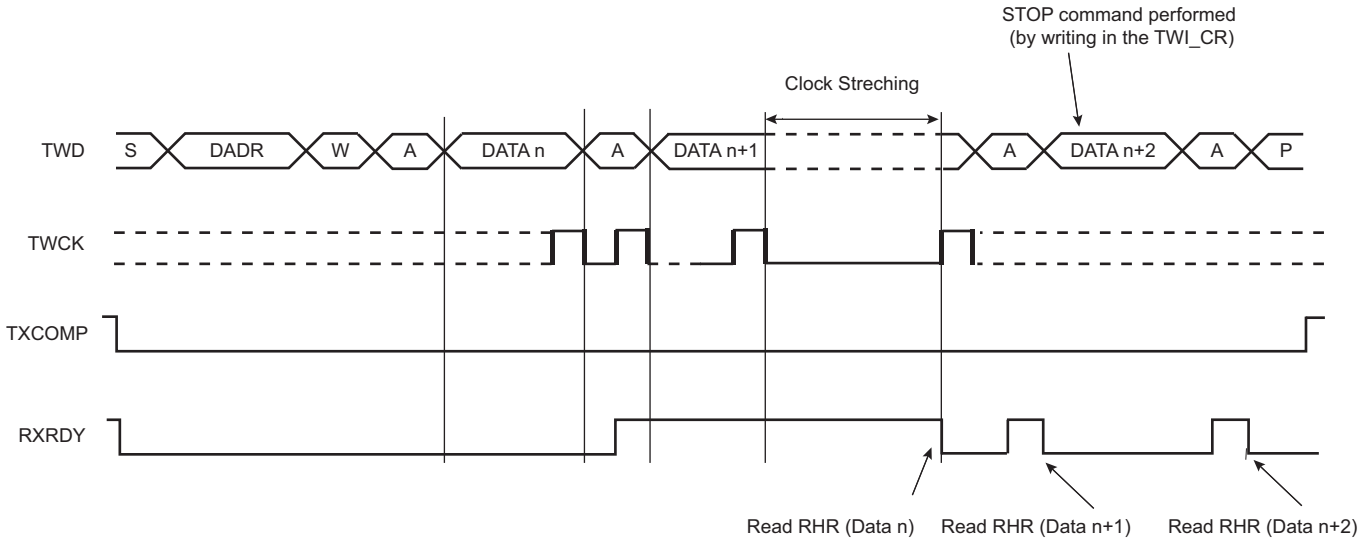


Figure 32-9. Master Read Clock Stretching with Multiple Data Bytes



RXRDY is used as receive ready trigger event for the PDC receive channel.

32.6.3.6 Internal Address

The TWI 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 I^2C fully-compatible devices. See Figure 32-11, ”Master Read with One, Two or Three Bytes Internal Address and One Data Byte”.

See Figure 32-10, ”Master Write with One, Two or Three Bytes Internal Address and One Data Byte” and Figure 32-12, ”Internal Address Usage” for the master write operation with internal address.

The three internal address bytes are configurable through the TWI 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.

The abbreviations listed below are used in Figure 32-10 and Figure 32-11:

S	Start
Sr	Repeated Start
P	Stop
W	Write
R	Read
A	Acknowledge
N	Not Acknowledge
DADR	Device Address
IADR	Internal Address

Figure 32-10. Master Write with One, Two or Three Bytes Internal Address and One Data Byte

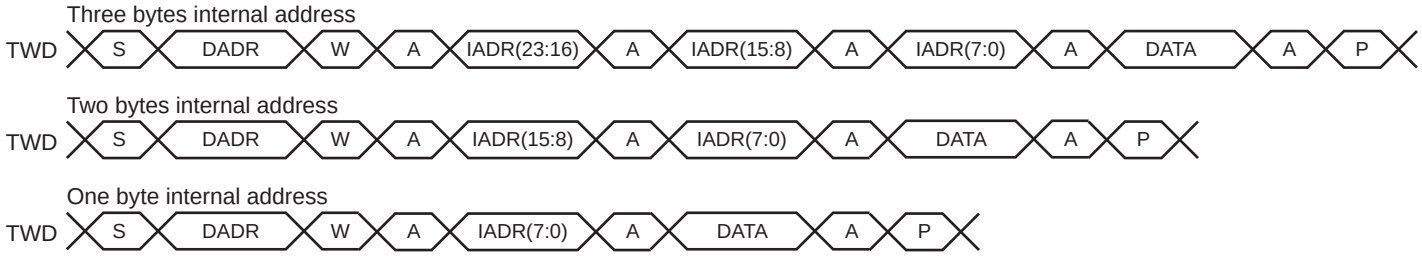
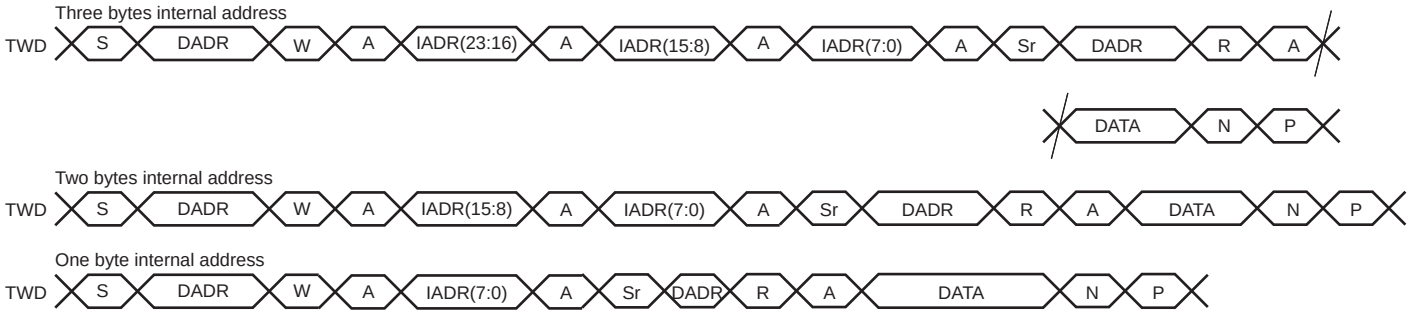


Figure 32-11. Master Read with One, Two or Three Bytes Internal Address and One Data Byte



10-bit Slave Addressing

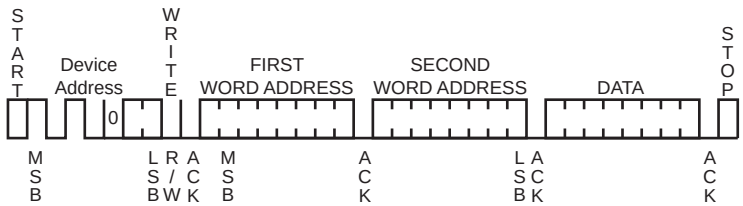
For a slave address higher than seven bits, the user must configure the address size **(**IADRSZ) and set the other slave address bits in the TWI Internal Address Register (TWI_IADR). The two remaining internal address bytes, IADR[15:8] and IADR[23:16], can be used the same way as in 7-bit slave addressing.

Example: Address a 10-bit device (10-bit device address is b1 b2 b3 b4 b5 b6 b7 b8 b9 b10)

1. Program IADRSZ = 1
2. Program DADR with 1 1 1 1 0 b1 b2 (b1 is the MSB of the 10-bit address, b2, etc.)
3. Program TWI_IADR with b3 b4 b5 b6 b7 b8 b9 b10 (b10 is the LSB of the 10-bit address)

Figure 32-12 shows a byte write to an Atmel AT24LC512 EEPROM. This demonstrates the use of internal addresses to access the device.

Figure 32-12. Internal Address Usage



32.6.3.7 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 32.6.3.14 for detailed flowcharts.

Note that generating a repeated start after a single data read is not supported.

32.6.3.8 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 TWI_CR.

Note: If alternative command is used (ACMEN bit set to 1) DATAL field must be set to 0.

32.6.3.9 Using the Peripheral DMA Controller (PDC) in Master Mode

The use of the PDC significantly reduces the CPU load.

To ensure 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.

If Alternative Command mode is disabled (ACMEN bit set to 0):

1. Initialize the transmit PDC (memory pointers, transfer size - 1).
2. Configure the Master mode (DADR, CKDIV, MREAD = 0, etc.).
3. Start the transfer by setting the PDC TXTEN bit.
4. Wait for the PDC ENDTX flag either by using the polling method or ENDTX interrupt.
5. Disable the PDC by setting the PDC TXTDIS bit.
6. Wait for the TXRDY flag in TWI_SR.
7. Set the STOP command in TWI_CR.
8. Write the last character in TWI_THR.
9. (Only if peripheral clock must be disabled) Wait for the TXCOMP flag to be raised in TWI_SR.

If Alternative Command mode is enabled (ACMEN bit set to 1):

1. Initialize the transmit PDC (memory pointers, transfer size).
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. (Only if peripheral clock must be disabled) Wait for the TXCOMP flag to be raised in TWI_SR.

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.

If Alternative Command mode is disabled (ACMEN bit set to 0):

1. Initialize the receive PDC (memory pointers, transfer size - 2).
2. Configure the Master mode (DADR, CKDIV, MREAD = 1, etc.).
3. Set the PDC RXTEN bit.
4. (Master Only) Write the START bit in 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 to end the transfer.
9. Read the penultimate character in TWI_RHR.

10. Wait for the RXRDY flag in TWI_SR.
11. Read the last character in TWI_RHR.
12. (Only if peripheral clock must be disabled) Wait for the TXCOMP flag to be raised in TWI_SR.

If Alternative Command mode is enabled (ACMEN bit set to 1):

1. Initialize the transmit PDC (memory pointers, transfer size).
2. Configure the Master mode (DADR, CKDIV, etc.).
3. Set the PDC RXTEN bit.
4. (Master Only) Write the START bit in TWI_CR to start the transfer.
5. Wait for the PDC ENDTX Flag either by using the polling method or ENDTX interrupt.
6. Disable the PDC by setting the PDC TXTDIS bit.
7. (Only if peripheral clock must be disabled) Wait for the TXCOMP flag to be raised in TWI_SR.

32.6.3.10 SMBus Mode

SMBus mode is enabled when the SMEN bit is written to one in TWI_CR. SMBus mode operation is similar to I²C operation with the following exceptions:

- Only 7-bit addressing can be used.
- The SMBus standard describes a set of timeout values to ensure progress and throughput on the bus. These timeout values must be programmed into the TWI SMBus Timing Register (TWI_SMBTR).
- Transmissions can optionally include a CRC byte, called Packet Error Check (PEC).
- 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 appropriately configuring TWI_CR.

Packet Error Checking

Each SMBus transfer can optionally end with a CRC byte, called the PEC byte. Writing the PECEN bit in TWI_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 TWI_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. If Alternative Command mode is enabled, only the NPEC bit should be set.

Consider the following transfer:

S, ADR+W, COMMAND_BYTE, ACK, SR, ADR+R, DATA_BYTE, ACK, PEC_BYTE, NACK, P

See Section 32.6.3.14 "Read/Write Flowcharts" for detailed flowcharts.

Timeouts

The TLOWS and TLOWM fields in TWI_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 TWI_SR.

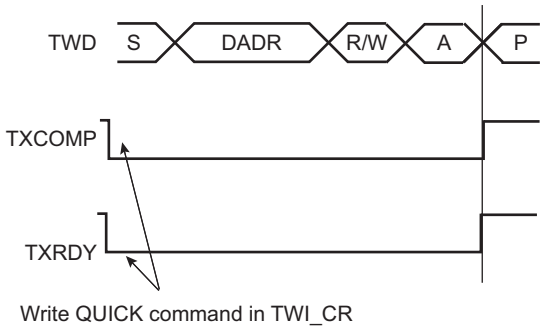
32.6.3.11 SMBUS Quick Command (Master Mode Only)

The TWI interface can perform a quick command:

1. Configure the Master mode (DADR, CKDIV, etc.).
2. Write the MREAD bit in TWI_MMR at the value of the one-bit command to be sent.
3. Start the transfer by setting the QUICK bit in TWI_CR.

Note: If alternative command is used (ACMEN bit set to 1) DATAL field must be set to 0.

Figure 32-13. SMBUS Quick Command



32.6.3.12 Alternative Command

Another way to configure the transfer is to enable the Alternative Command mode with the ACMEN bit of TWI_CR.

In this mode, the transfer is configured through the TWI Alternative Command Register (TWI_ACR). It is possible to define a simple read or write transfer or a combined transfer with a repeated start.

In order to set a simple transfer, the DATAL field and the DIR bit of TWI_ACR must be filled accordingly and the NDATAL field must be cleared. To begin the transfer, either set the START bit in TWI_CR in case of a read transfer, or write in TWI_THR in case of a write transfer.

For a combined transfer linked by a repeated start, the NDATAL field must be filled with the length of the second transfer and the NDIR bit with the corresponding direction.

The PEC and NPEC bits are used to set a PEC bit. In the case of a single transfer with PEC, the PEC bit must be set. In the case of a combined transfer, the NPEC bit must be set.

Note: If Alternative Command mode is used, the field TWI_MMR.IADRSZ must be set to 0.

See Section 32.6.3.14 for detailed flowcharts.

32.6.3.13 Handling Errors in Alternative Command

In case of NACK generated by a slave device or SMBus timeout error, the TWI stops immediately the frame but the PDC transfer may still be active. To prevent a new frame to be restarted with remaining PDC data (transmit), the TWI prevents any start of frame until the LOCK flag is cleared in TWI_SR.

The LOCK bit in TWI_SR indicates the state of the TWI (locked or not locked).

When the TWI is locked, no transfer will begin until the LOCK is cleared using the LOCKCLR bit in TWI_CR and error flags cleared reading TWI_SR.

In case of error, TWI_THR may have been loaded with a new data. The THRCLR bit in TWI_CR can be used to flush TWI_THR. If the THRCLR bit is set, the TXRDY and TXCOMP flags are set.

32.6.3.14 Read/Write Flowcharts

The flowcharts shown in Figure 32-15, Figure 32-16, Figure 32-21, Figure 32-22 and Figure 32-23 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 TWI_IER be configured first.

Figure 32-14. TWI Write Operation with Single Data Byte without Internal Address

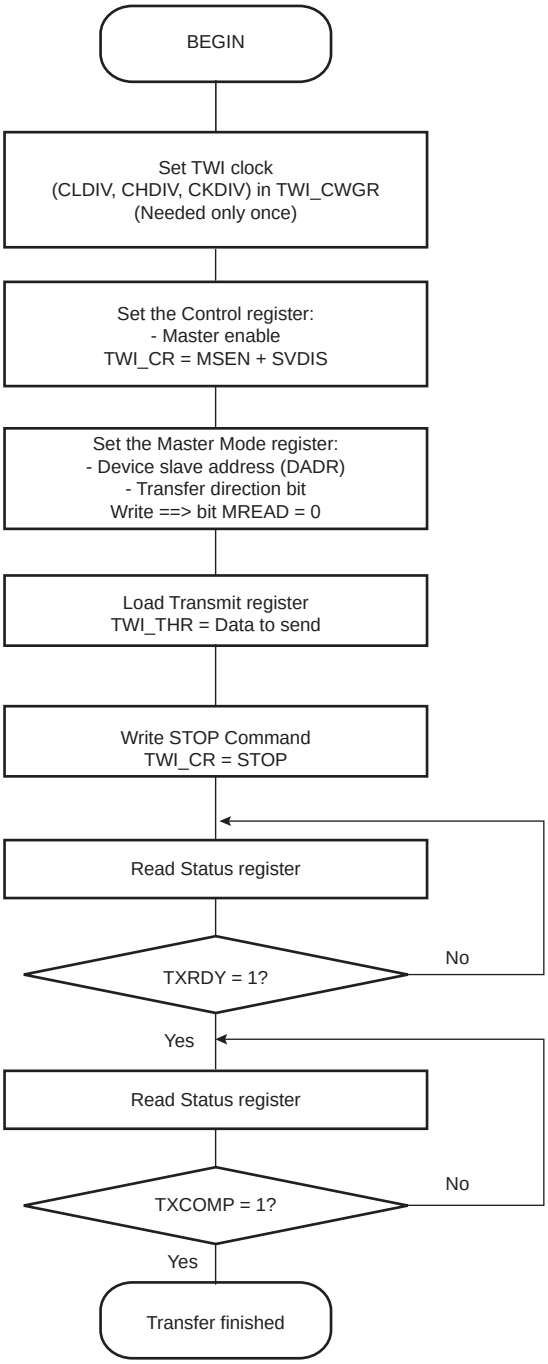


Figure 32-15. TWI Write Operation with Single Data Byte and Internal Address

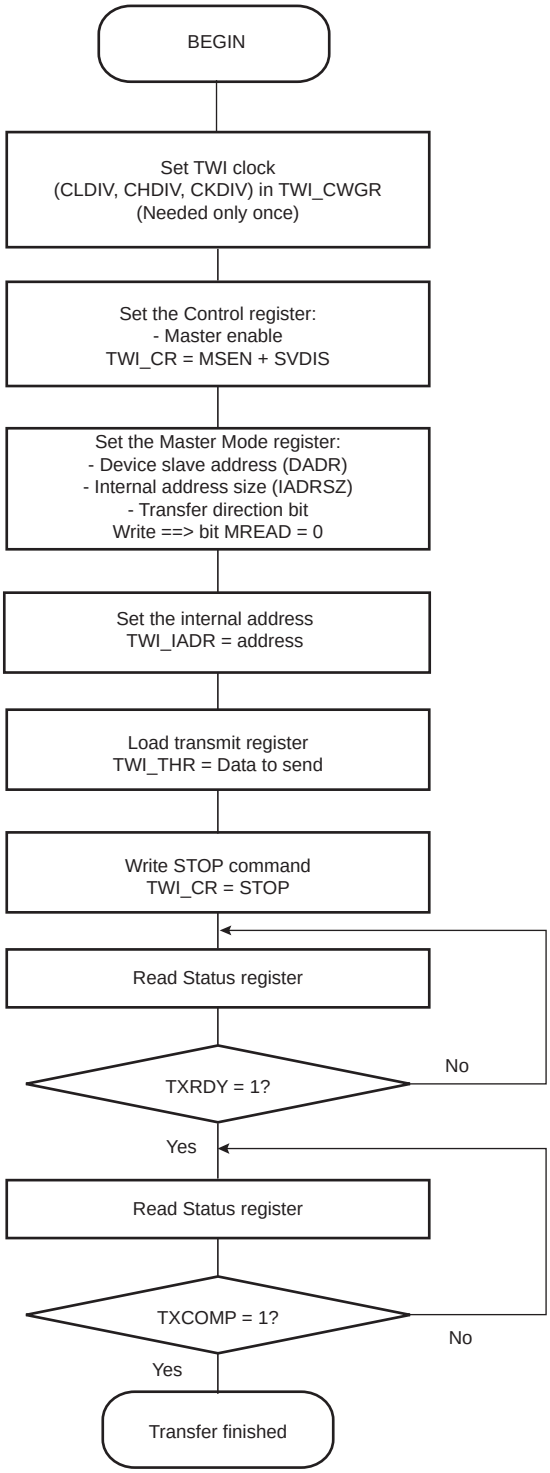


Figure 32-16. TWI Write Operation with Multiple Data Bytes with or without Internal Address

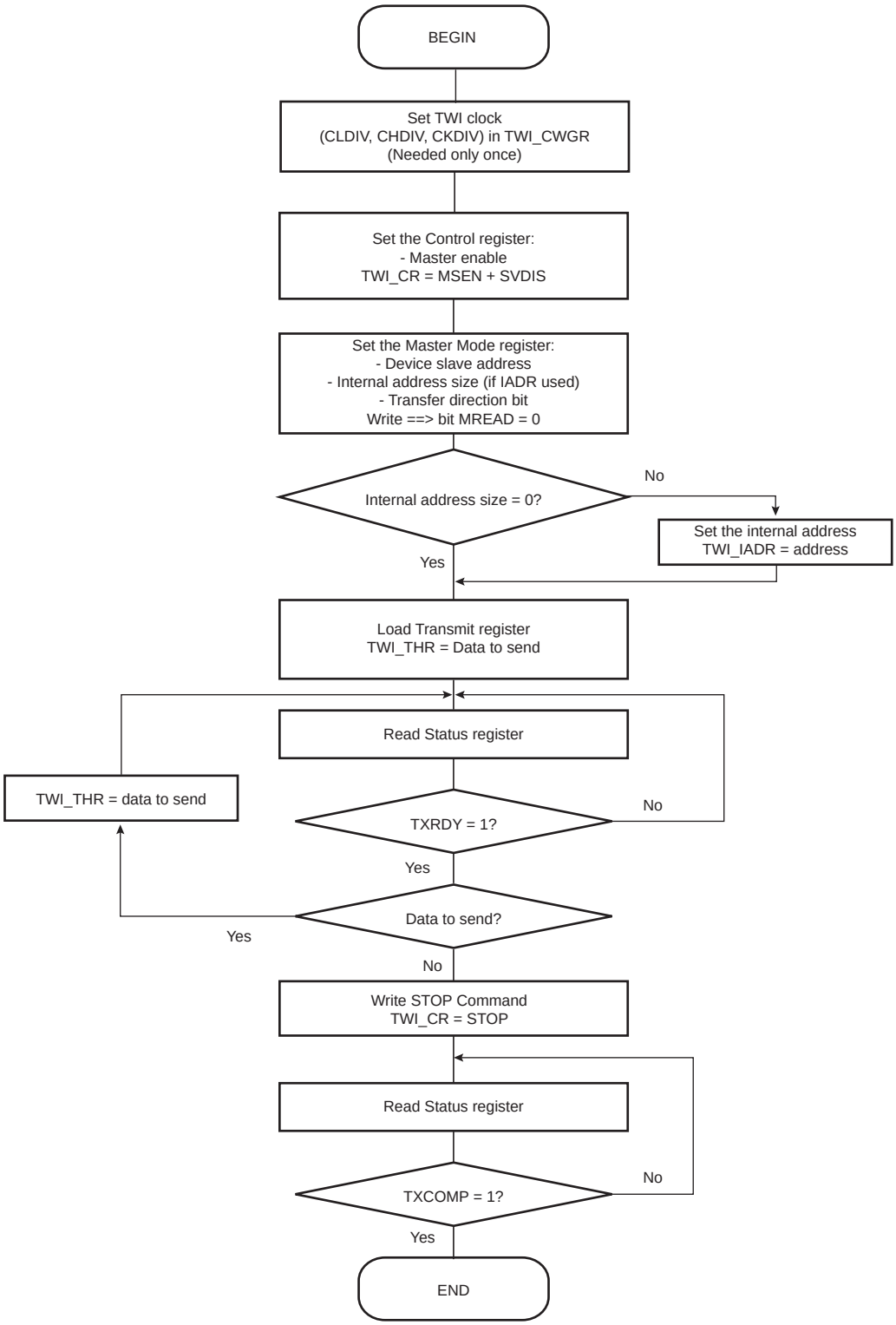


Figure 32-17. SMBus Write Operation with Multiple Data Bytes with or without Internal Address and PEC Sending

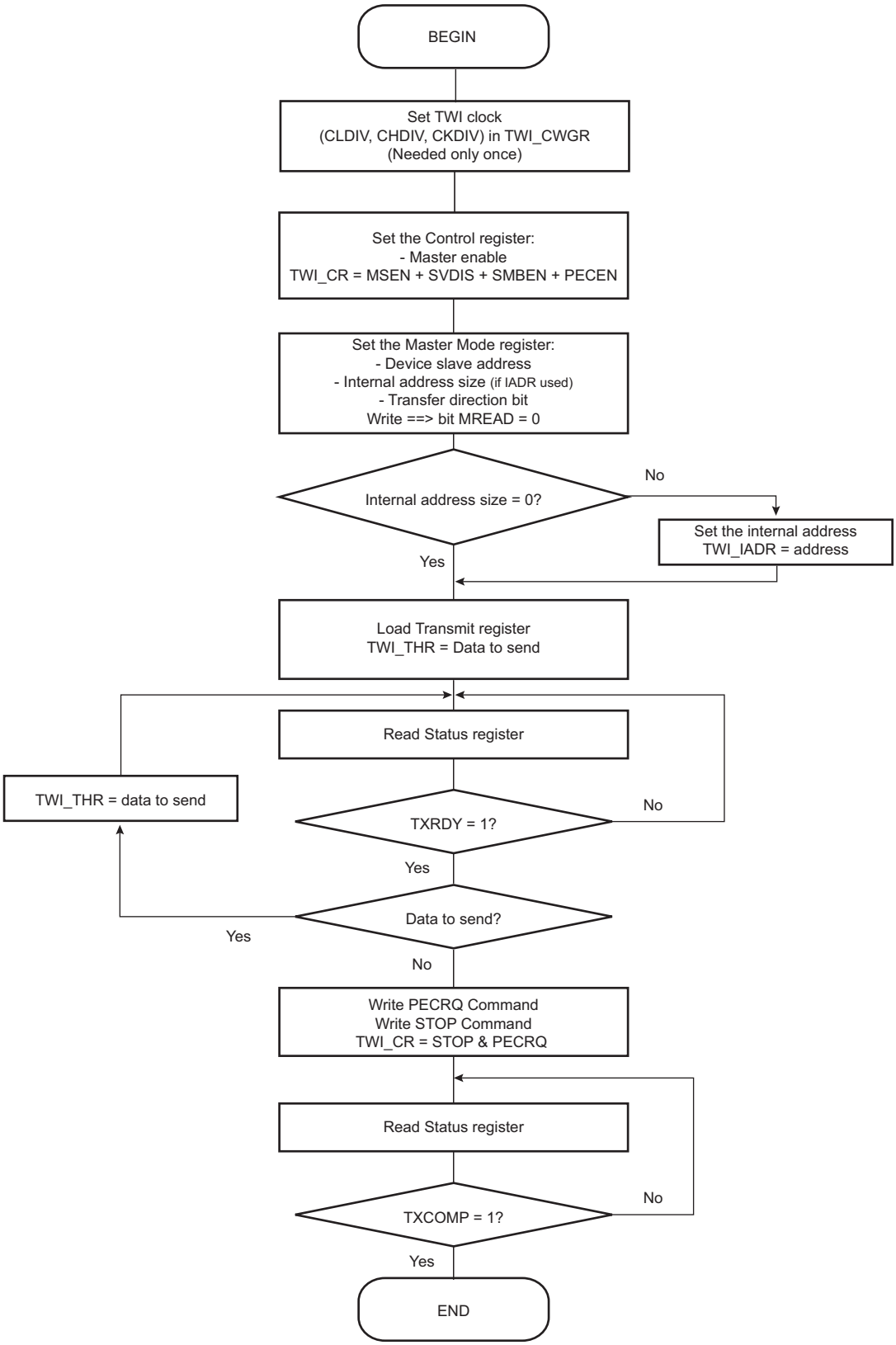


Figure 32-18. SMBus Write Operation with Multiple Data Bytes with PEC and Alternative Command Mode

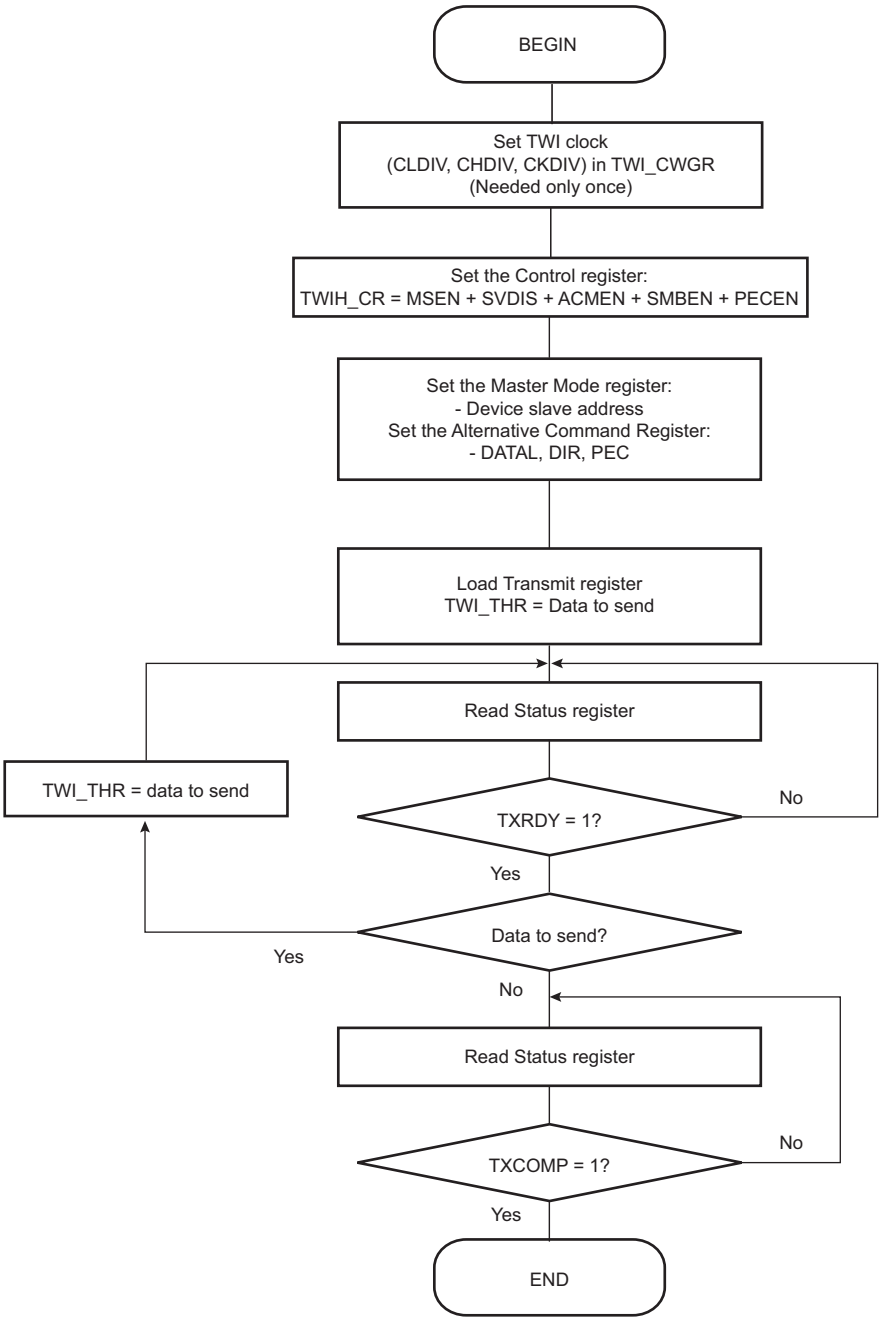


Figure 32-19. TWI Write Operation with Multiple Data Bytes and Read Operation with Multiple Data Bytes (Sr)

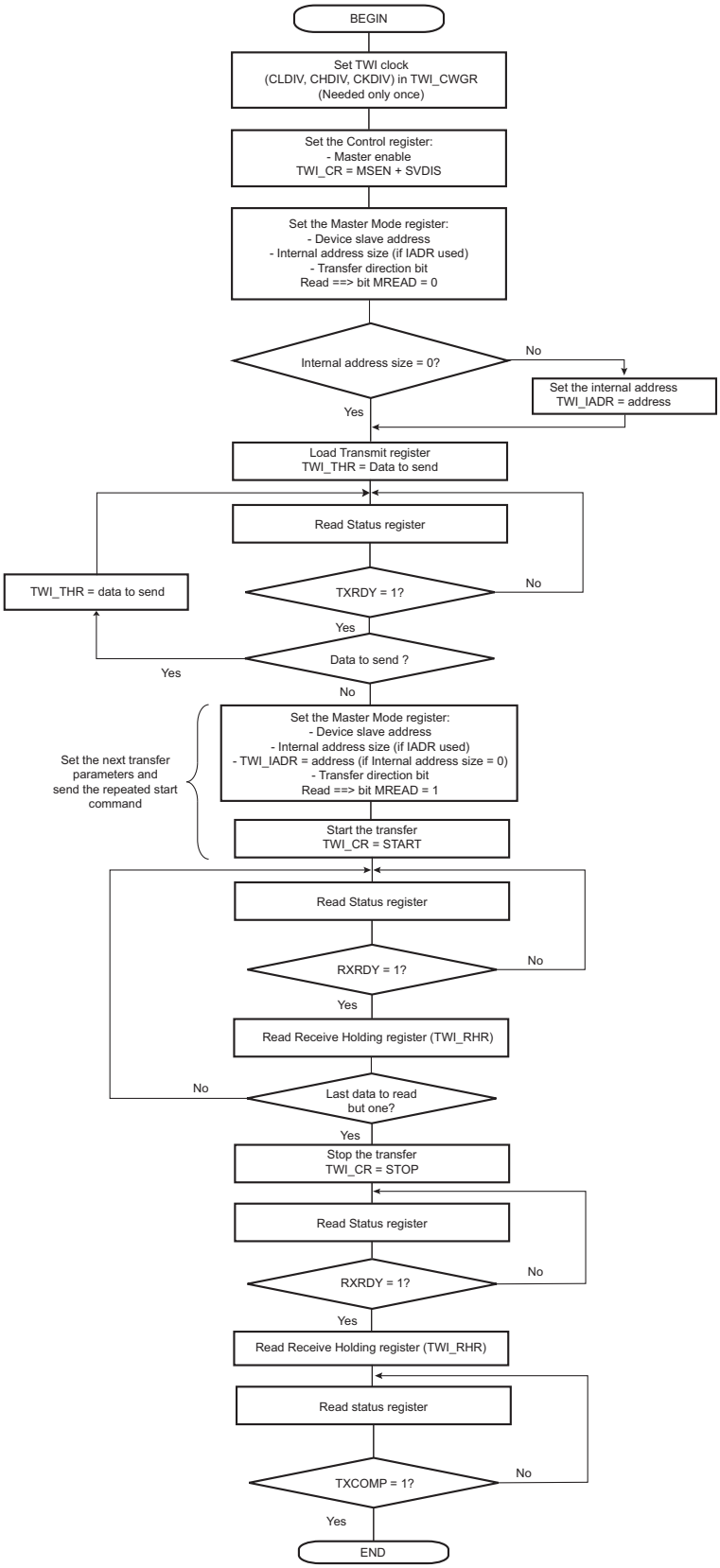


Figure 32-20. TWI Write Operation with Multiple Data Bytes + Read Operation and Alternative Command Mode + PEC

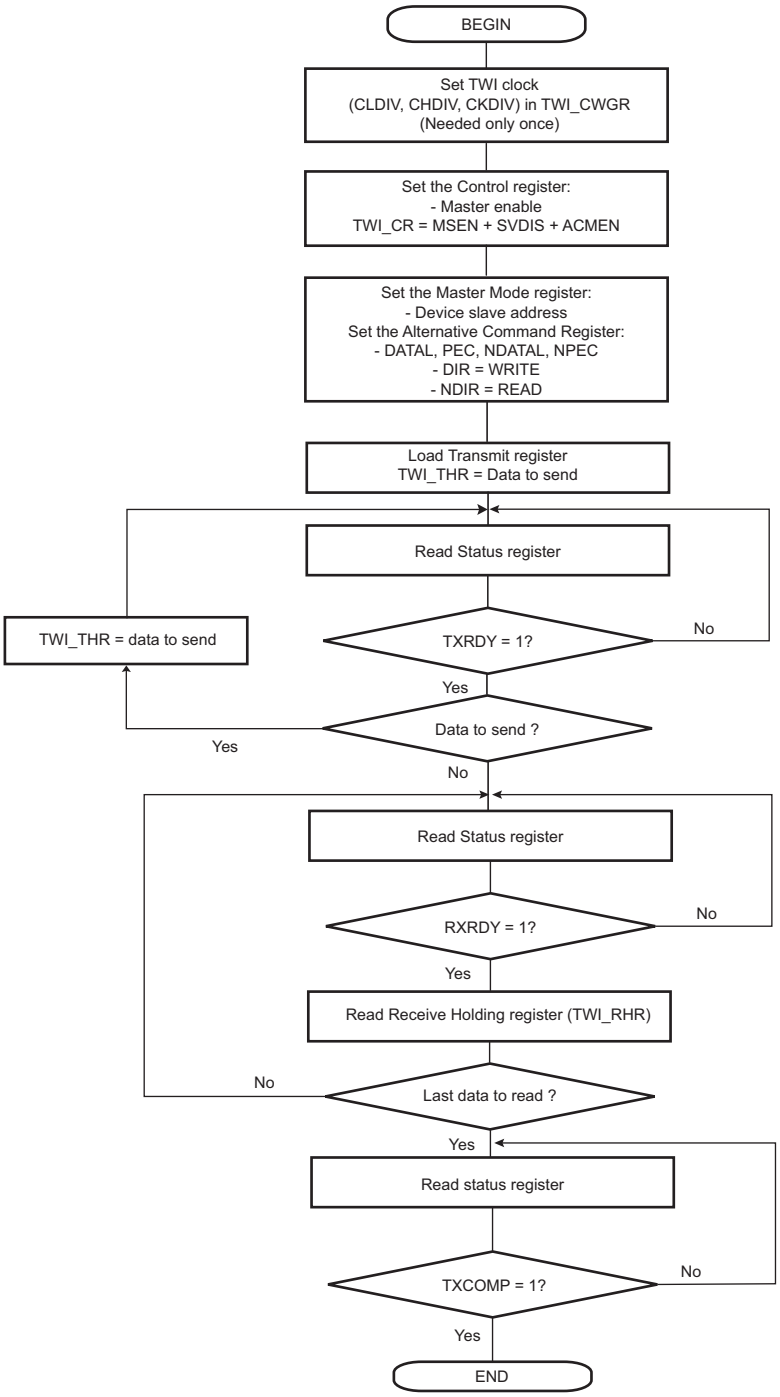


Figure 32-21. TWI Read Operation with Single Data Byte without Internal Address

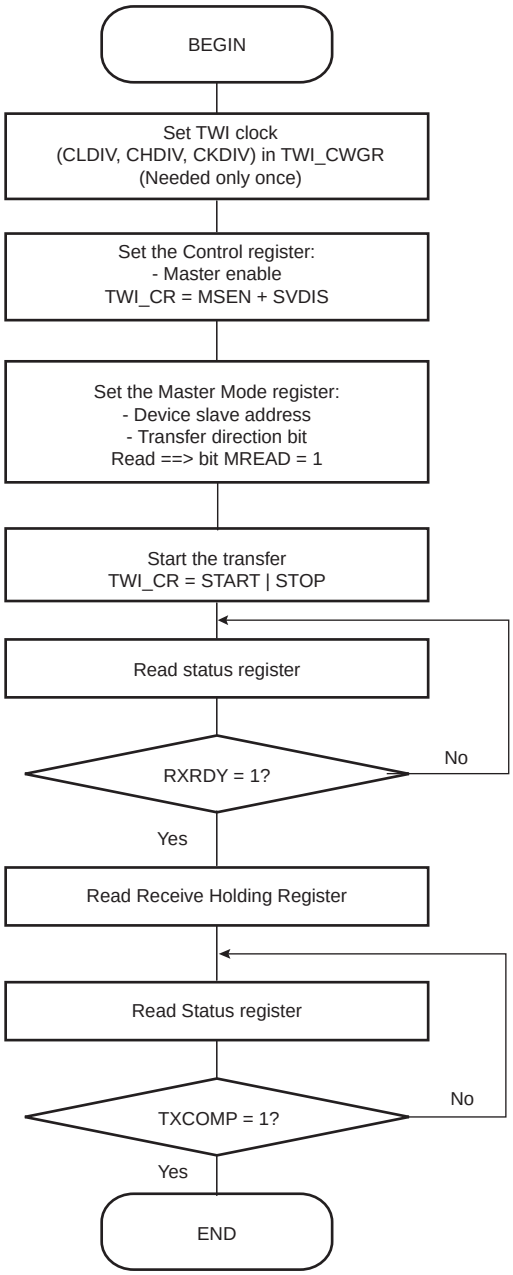


Figure 32-22. TWI Read Operation with Single Data Byte and Internal Address

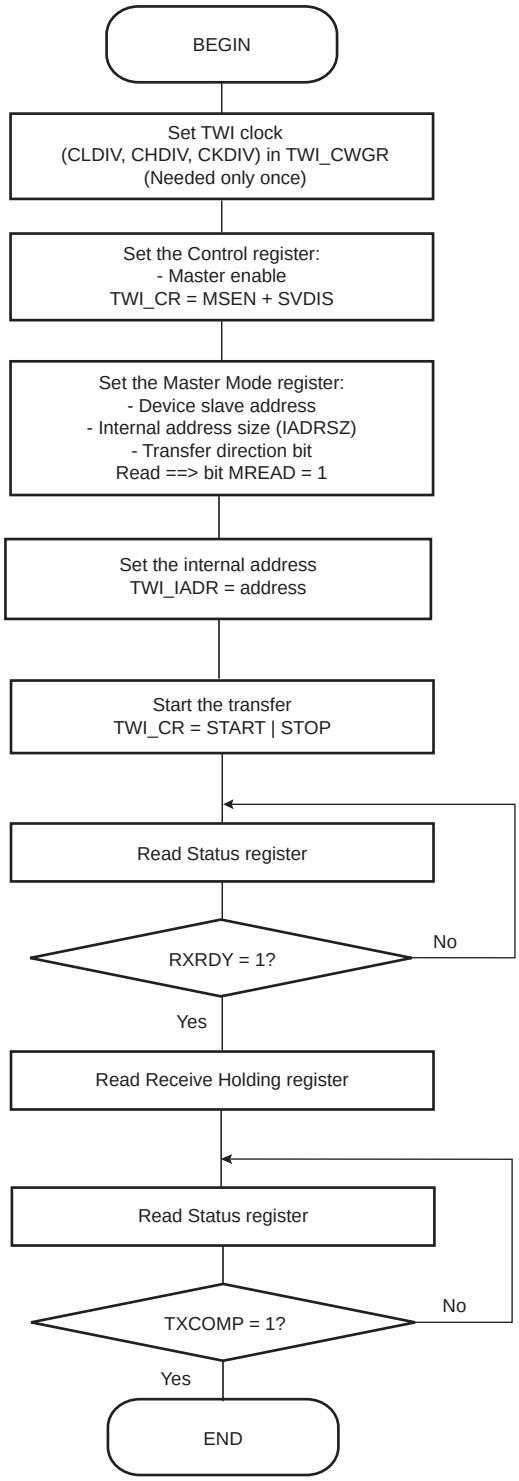


Figure 32-23. TWI Read Operation with Multiple Data Bytes with or without Internal Address

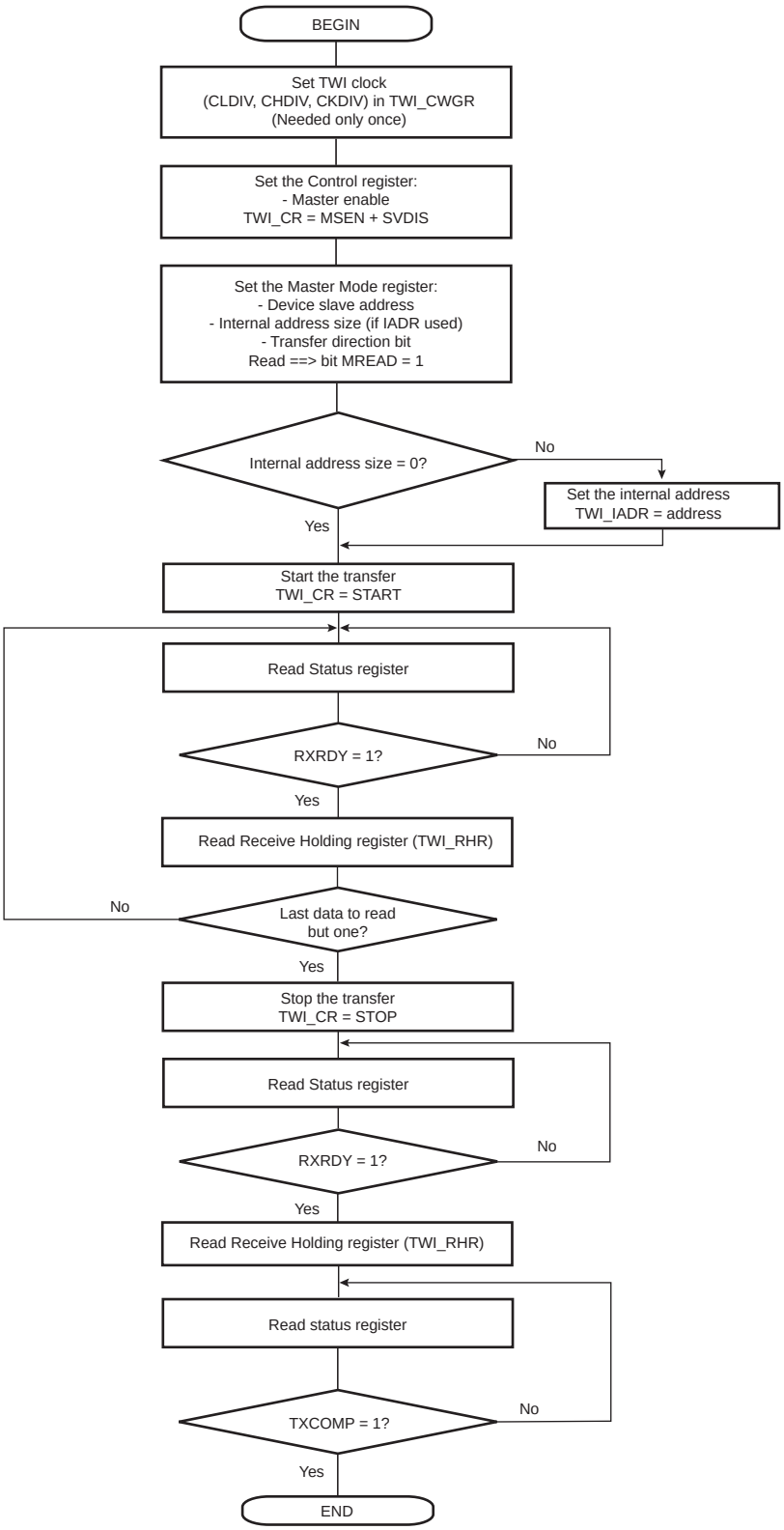


Figure 32-24. TWI Read Operation with Multiple Data Bytes with or without Internal Address with PEC

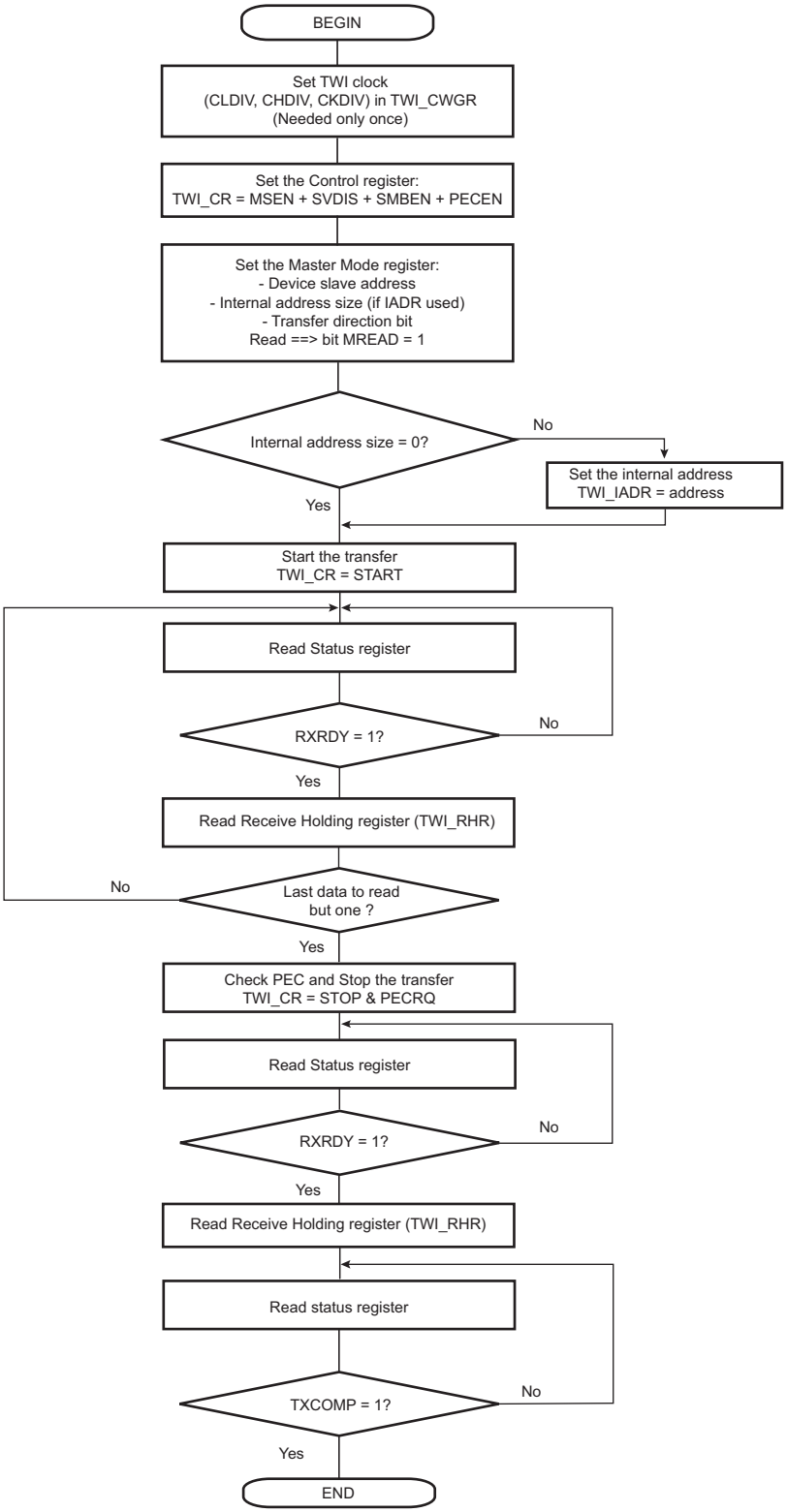


Figure 32-25. TWI Read Operation with Multiple Data Bytes with Alternative Command Mode with PEC

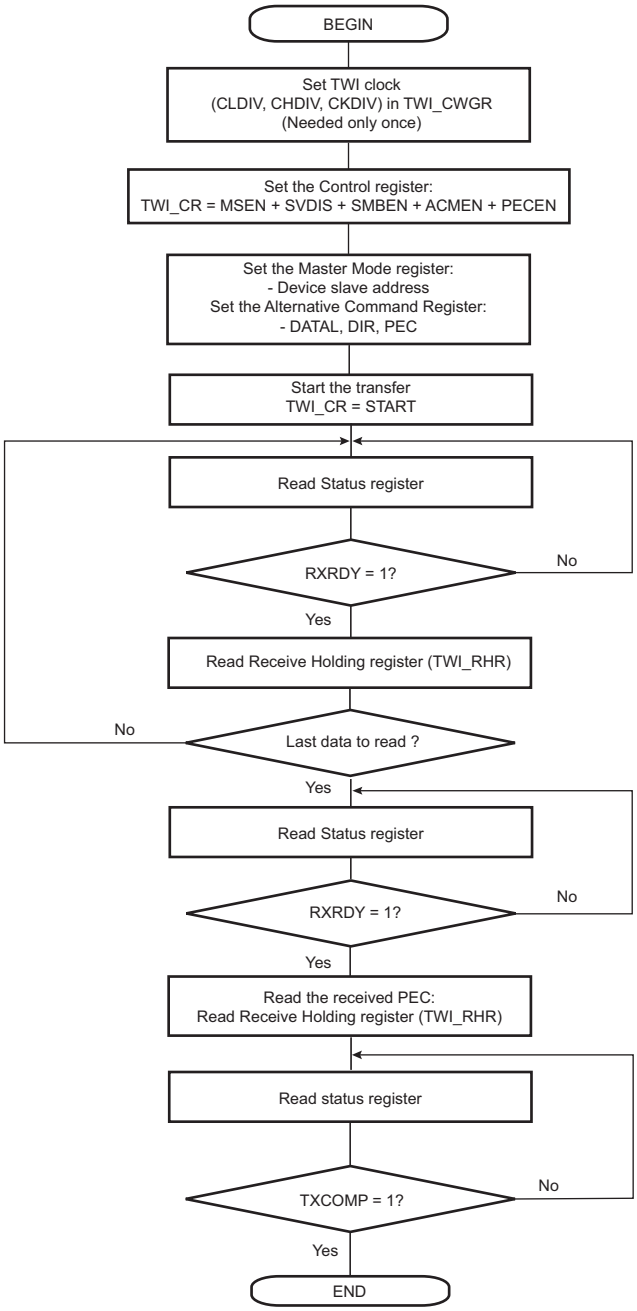


Figure 32-26. TWI Read Operation with Multiple Data Bytes + Write Operation with Multiple Data Bytes (Sr)

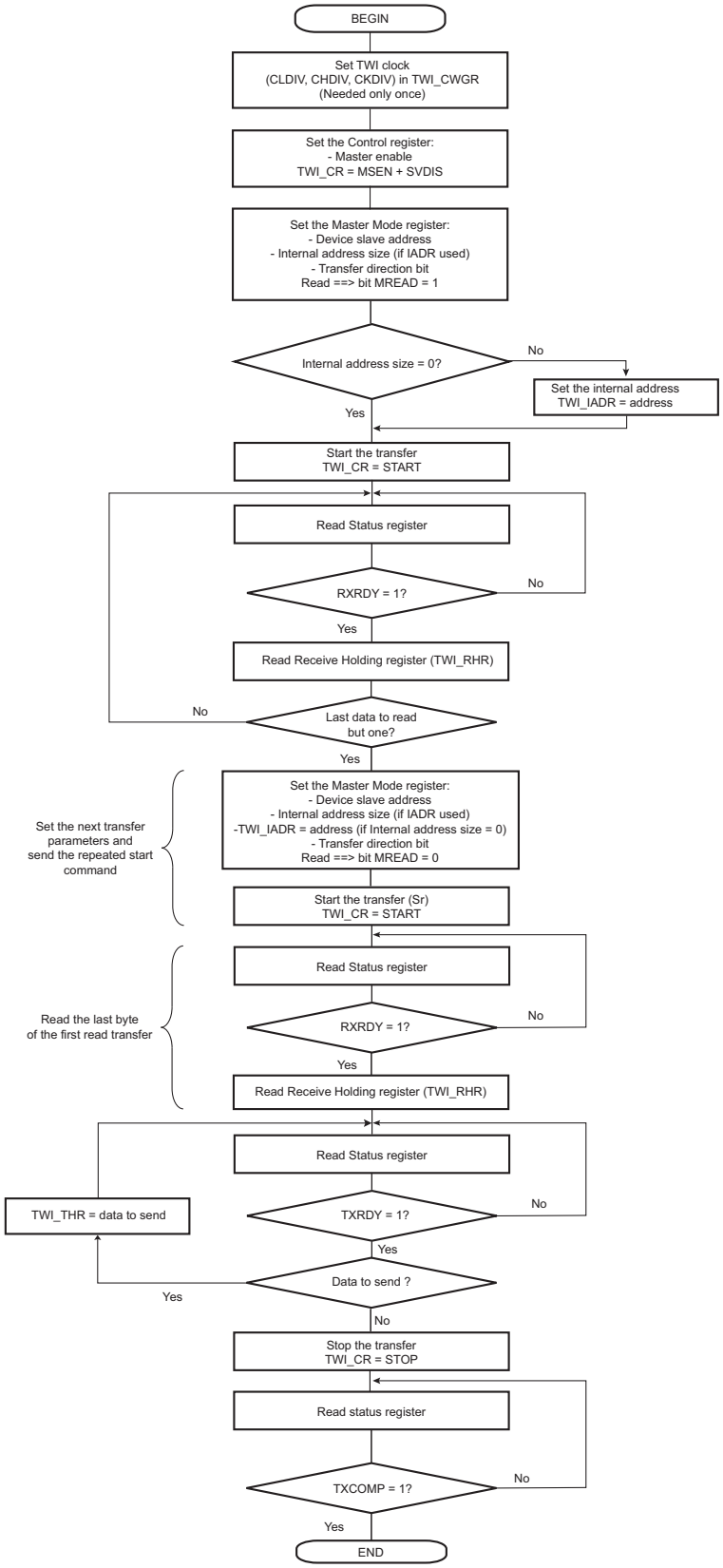
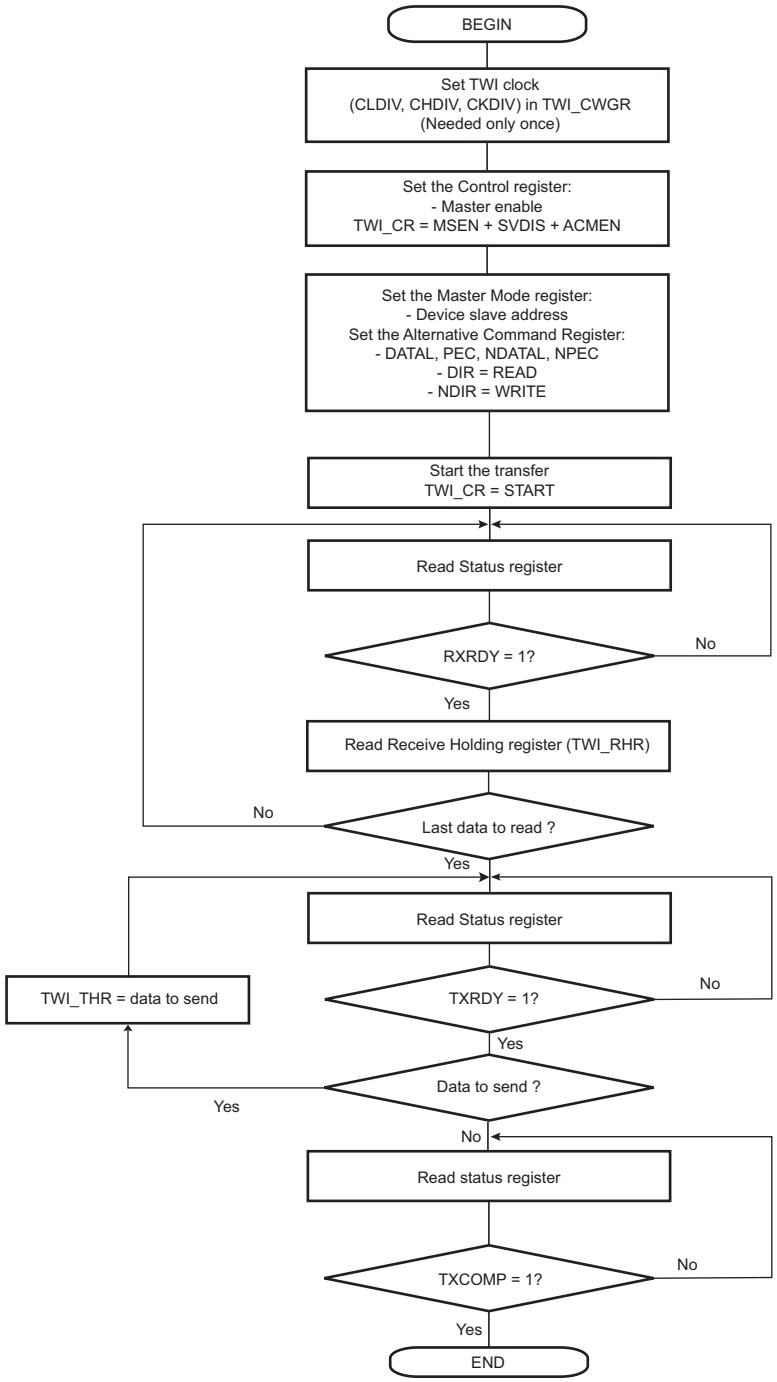


Figure 32-27. TWI Read Operation with Multiple Data Bytes + Write with Alternative Command Mode with PEC



32.6.4 Multi-Master Mode

32.6.4.1 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 32-29.

32.6.4.2 Different Multi-Master Modes

Two Multi-master modes may be distinguished:

- TWI as Master Only—TWI is considered as a master only and will never be addressed.
- TWI as Master or Slave—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 32-28).

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 32-28. User Sends Data While the Bus is Busy

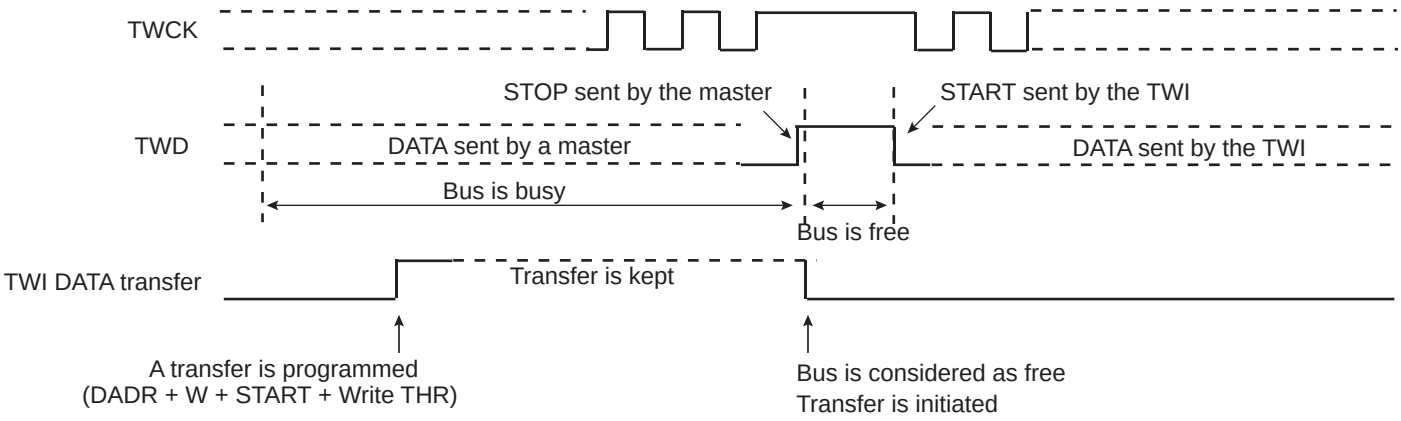
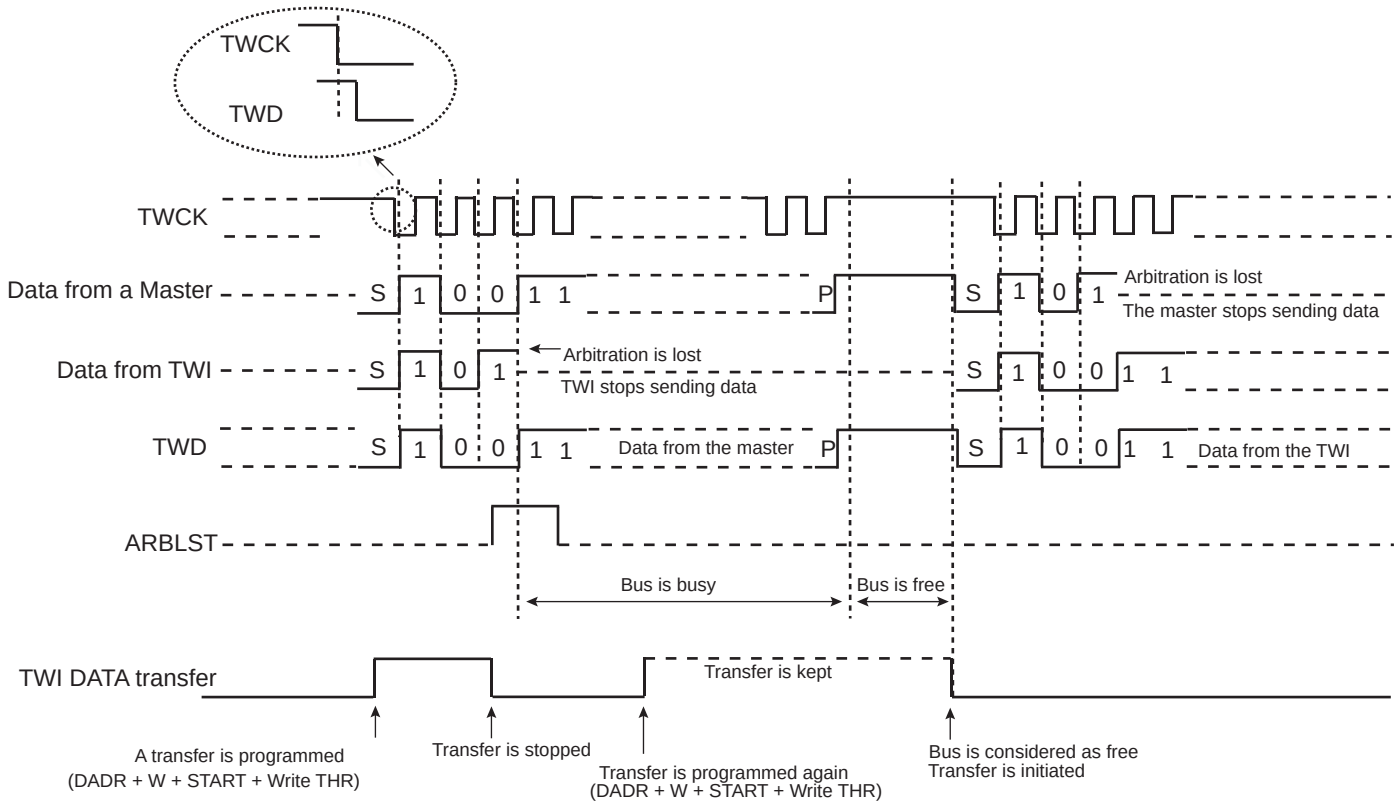
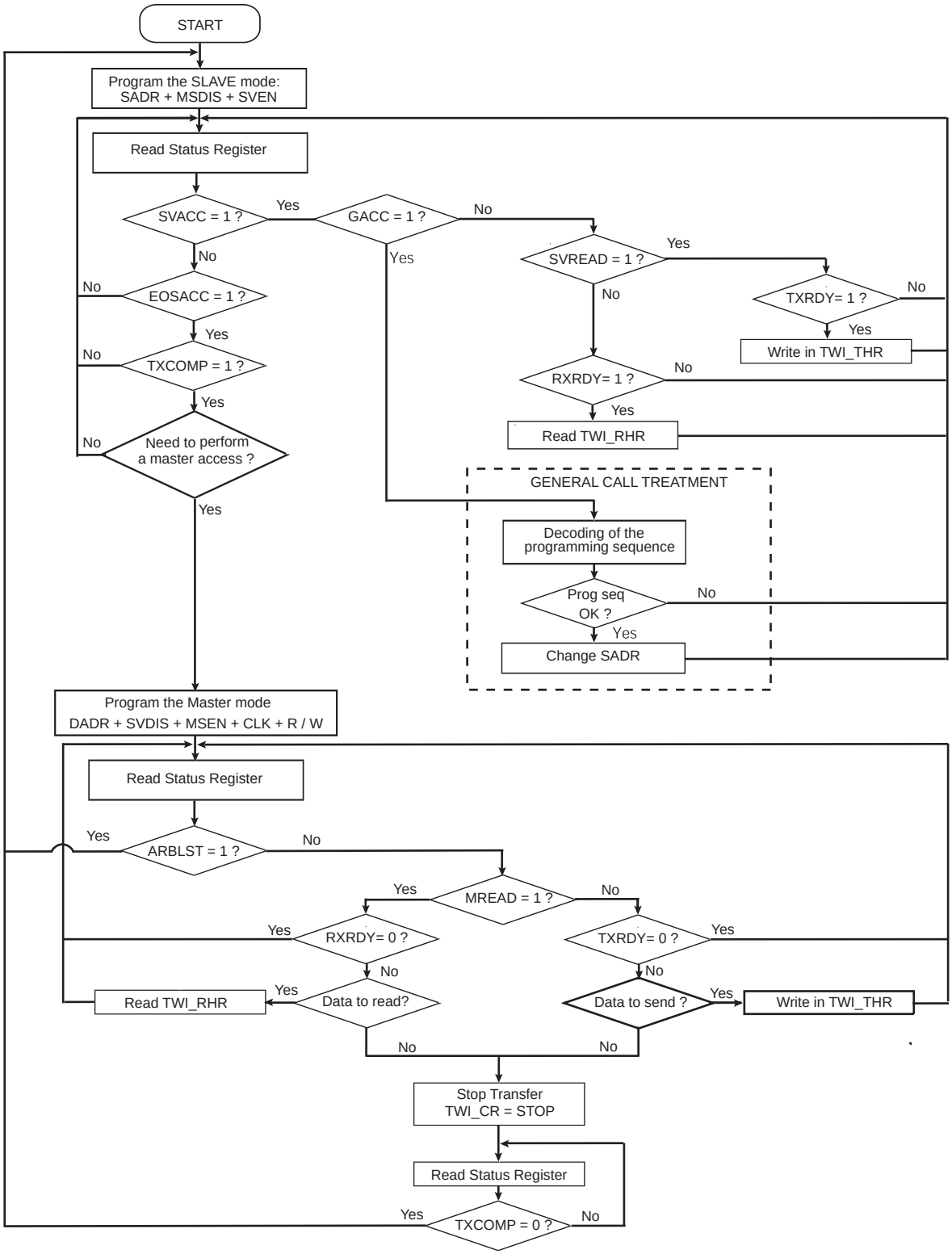


Figure 32-29. Arbitration Cases



The flowchart shown in Figure 32-30 gives an example of read and write operations in Multi-master mode.

Figure 32-30. Multi-Master Flowchart



32.6.5 Slave Modes

32.6.5.1 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).

32.6.5.2 Programming Slave Mode

The following fields must be programmed before entering Slave mode:

1. TWI_SMR.SADR: The slave device address is used in order to be accessed by master devices in Read or Write mode.
2. (Optional) TWI_SMR.MASK can be set to mask some SADR address bits and thus allow multiple address matching.
3. TWI_CR.MSDIS: Disables the Master mode.
4. TWI_CR.SVEN: Enables the Slave mode.

As the device receives the clock, values written in TWI_CWGR are not taken into account.

32.6.5.3 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 TWI_THR 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 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 32-31.

Note: In Slave mode, the TXRDY flag can be cleared by first setting the bit TWI_CR.SVDIS and then setting the bit TWI_CR.SVEN.

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 TWI_RHR. RXRDY is reset when reading 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 32-32.

Clock Stretching Sequence

If TWI_THR or TWI_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 32-34 and Figure 32-35.

Note: Clock stretching can be disabled by configuring the SCLWSDIS bit in the TWI Slave Mode Register (TWI_SMR). In that case, UNRE and OVRE flags will indicate underrun (when TWI_THR is not filled on time) or overrun (when TWI_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 32-33.

32.6.5.4 Data Transfer

Read Operation

The Read mode is defined as a data requirement from the master.

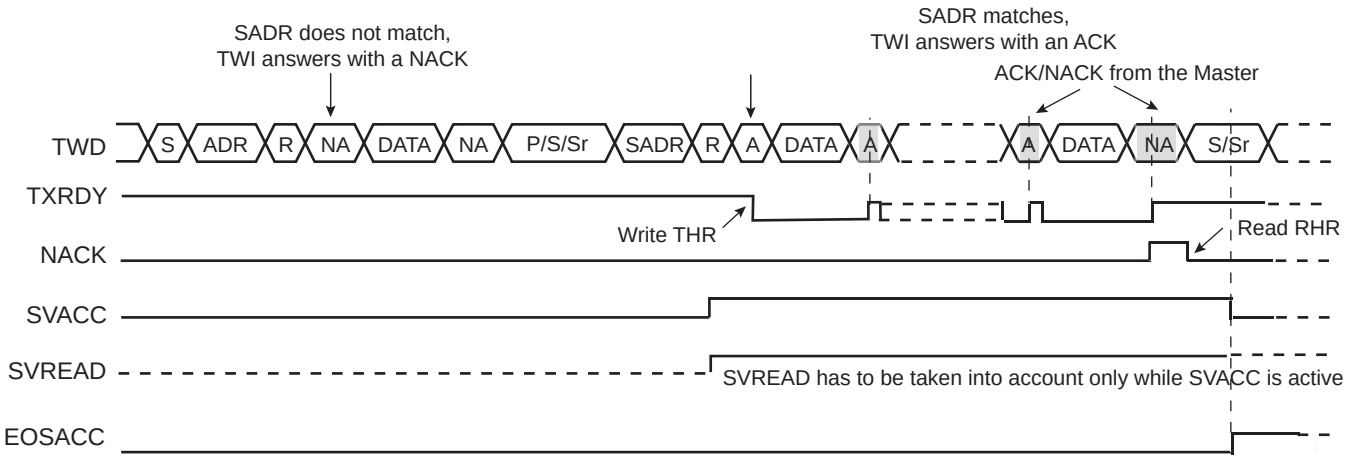
After a START or a REPEATED START condition is detected, the decoding of the address starts. If the slave address (SADR) is decoded, SVACC is set and SVREAD indicates the direction of the transfer.

Until a STOP or REPEATED START condition is detected, TWI continues sending data loaded in TWI_THR.

If a STOP condition or a REPEATED START + an address different from SADR is detected, SVACC is reset.

Figure 32-31 describes the read operation.

Figure 32-31. 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.

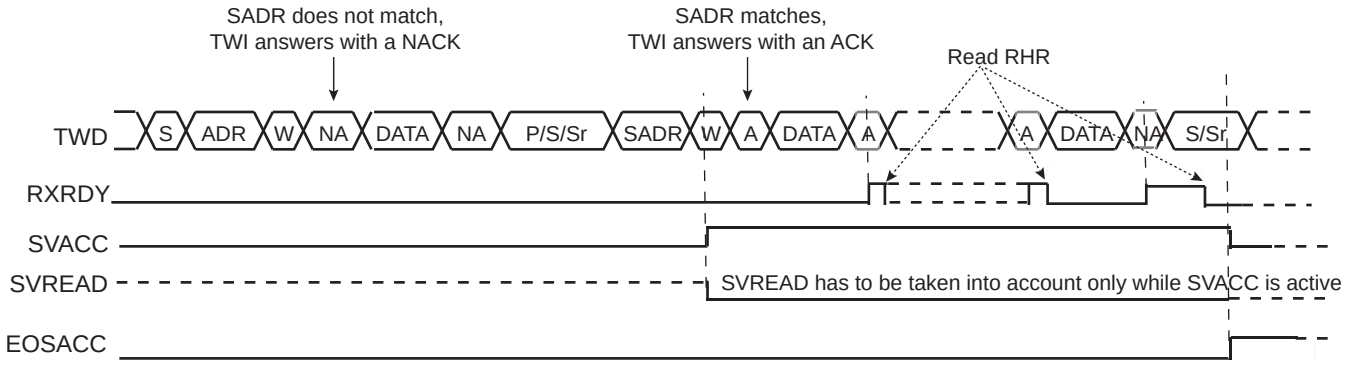
After a START or a REPEATED START, the decoding of the address starts. If the slave address is decoded, SVACC is set and SVREAD indicates the direction of the transfer (SVREAD is low in this case).

Until a STOP or REPEATED START condition is detected, TWI stores the received data in TWI_RHR.

If a STOP condition or a REPEATED START + an address different from SADR is detected, SVACC is reset.

Figure 32-32 describes the write operation.

Figure 32-32. 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 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.

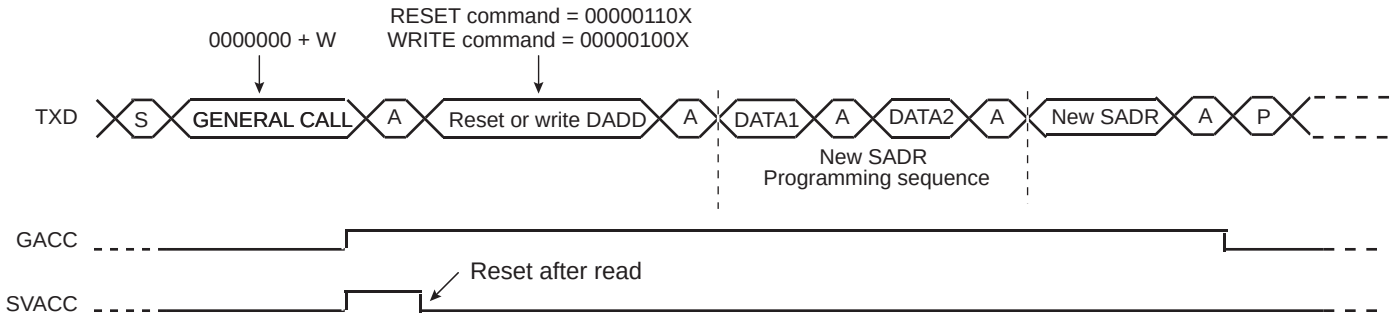
If a GENERAL CALL is detected, GACC is set.

After the detection of general call, it is up to the 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 32-33 describes the general call access.

Figure 32-33. Master Performs a General Call



Note: This method allows to create a user-specific programming sequence by choosing the number of programming bytes. The programming sequence has to be provided to the master.

Clock Stretching

In both Read and Write modes, it may happen that the 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.

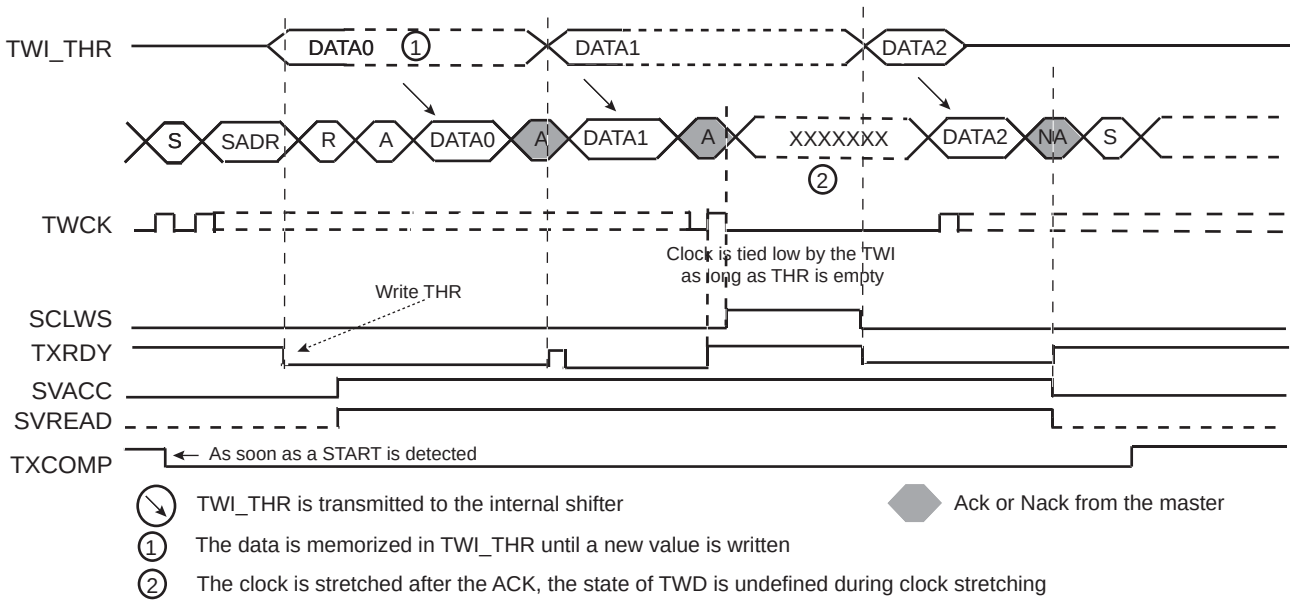
Note: Clock stretching can be disabled by setting the SCLWSDIS bit in TWI_SMR. In that case UNRE and OVRE flags will indicate underrun (when TWI_THR is not filled on time) or overrun (when TWI_RHR is not read on time).

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 32-34 describes the clock stretching in Read mode.

Figure 32-34. Clock Stretching in Read Mode



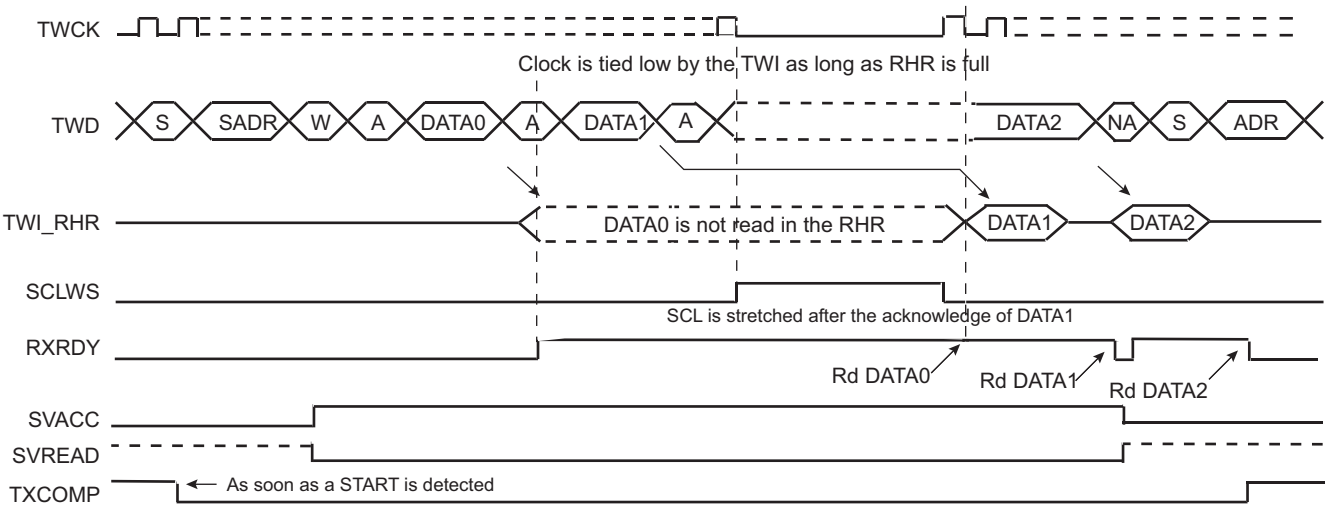
Notes:
1. TXRDY is reset when data has been written in TWI_THR to the internal shifter and set when this data has been acknowledged or non acknowledged.
2. At the end of the read sequence, TXCOMP is set after a STOP or after a REPEATED_START + an address different from SADR.
3. SCLWS is automatically set when the clock stretching mechanism is started.

Clock Stretching in Write Mode

The clock is tied low if the internal shifter and TWI_RHR are full. If a STOP or REPEATED_START condition was not detected, it is tied low until TWI_RHR is read.

Figure 32-35 describes the clock stretching in Write mode.

Figure 32-35. 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.

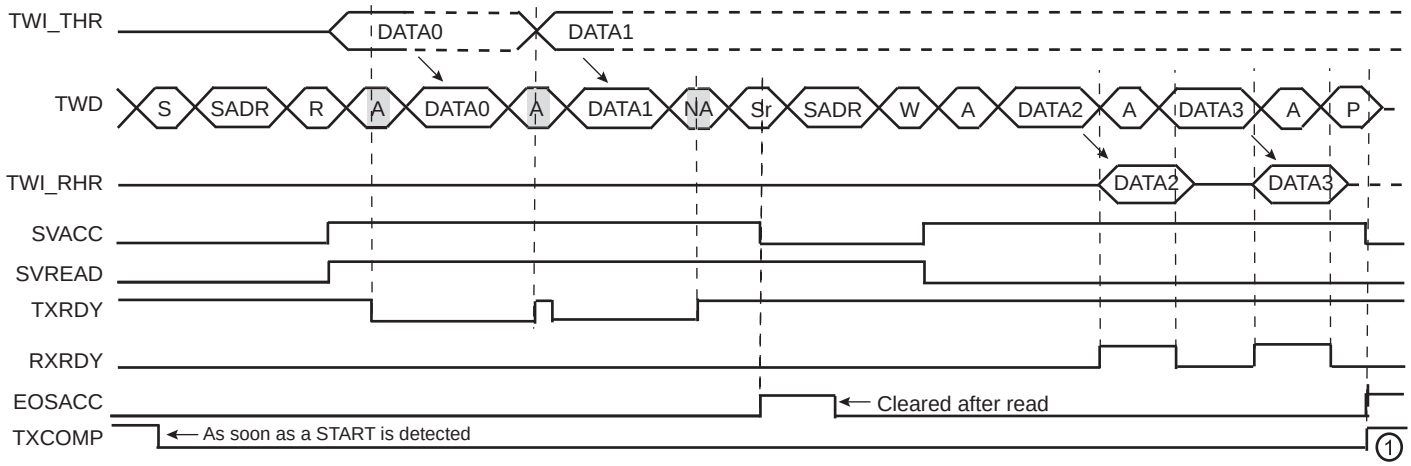
Reversal after a Repeated Start

Reversal of Read to Write

The master initiates the communication by a read command and finishes it by a write command.

Figure 32-36 describes the repeated start and the reversal from Read mode to Write mode.

Figure 32-36. Repeated Start and Reversal from Read Mode to Write Mode

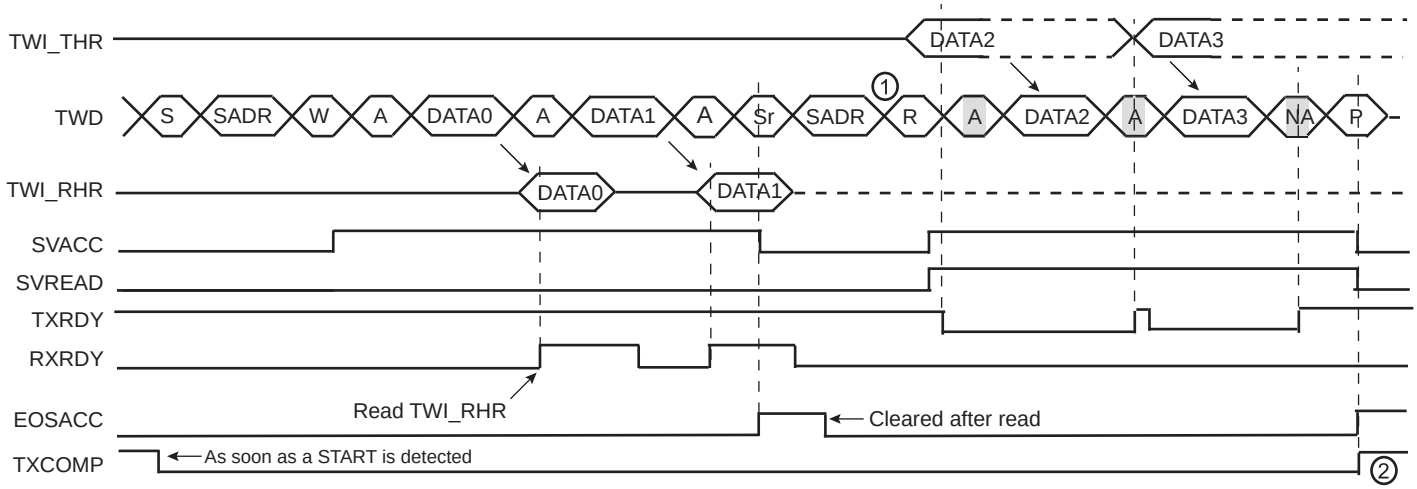


Note: 1. TXCOMP is only set at the end of the transmission because after the repeated start, SADR is detected again.

Reversal of Write to Read

The master initiates the communication by a write command and finishes it by a read command. Figure 32-37 describes the repeated start and the reversal from Write mode to Read mode.

Figure 32-37. Repeated Start and Reversal from Write Mode 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.

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. (Only if peripheral clock must be disabled) Wait for the TXCOMP flag to be raised in TWI_SR.

Data Receive with the PDC in Slave Mode

The following procedure shows an example 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. (Only if peripheral clock must be disabled) Wait for the TXCOMP flag to be raised in TWI_SR.

SMBus Mode

SMBus mode is enabled when SMEN bit is written to one in TWI_CR. SMBus mode operation is similar to I²C operation with the following exceptions:

- Only 7-bit addressing can be used.
- The SMBus standard describes a set of timeout values to ensure progress and throughput on the bus. These timeout values must be programmed into TWI_SMBTR.

- 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 appropriately configuring TWI_CR.

Packet Error Checking

Each SMBus transfer can optionally end with a CRC byte, called the PEC byte. Writing the PECEN bit in TWI_CR to one will send/check the PEC field of the TWI frame 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 TWI_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 32.6.5.8 ”Slave Read/Write Flowcharts” for detailed flowcharts.

Timeouts

TWI_SMBTR configures the SMBus timeout values. If a timeout occurs, the slave will leave the bus. The TOUT bit is also set in TWI_SR.

32.6.5.5 High-Speed Slave Mode

High-speed mode is enabled when the HSEN bit is written to one in TWI_CR. Furthermore, the analog pad filter must be enabled, the PADFEN bit must be written to one in TWI_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 the peripheral clock runs at a minimum of 14 MHz if slave clock stretching is enabled (SCLWSDIS bit at ‘0’). If slave clock stretching is disabled (SCLWSDIS bit at ‘1’), the peripheral clock must run at a minimum of 11 MHz (theoretical value given considering the absence of latency in the system).

Note: When slave clock stretching is disabled, TWI_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 TWI_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 TWI_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 TWI_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.

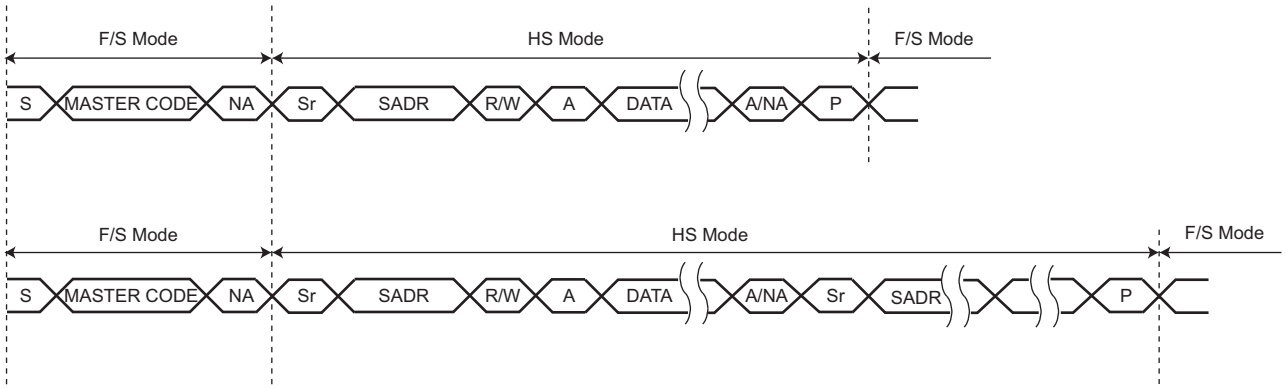
Read/Write Operation

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 32-38. High-Speed Mode Read/Write



Usage

TWI High-speed mode usage is the same as the standard TWI (see Section 32.6.3.14).

32.6.5.6 Alternative Command

In Slave mode, Alternative Command mode is used when SMBus mode is enabled to send or check the PEC byte.

Alternative Command mode is enabled by setting the ACMEN bit of the TWI Control Register and the transfer is configured in TWI_ACR.

For a combined transfer with PEC, only the NPEC bit in TWI_ACR must be set as the PEC byte is sent once at the end of the frame.

See Section 32.6.5.8 ”Slave Read/Write Flowcharts” for detailed flowcharts.

32.6.5.7 TWI Asynchronous and Partial Wakeup (SleepWalking)

The TWI module includes an asynchronous start condition detector, it is capable of waking the device up from a Sleep mode upon an address match (and optionally an additional data match), including Sleep modes where the TWI peripheral clock is stopped.

TWI_RHR must be read prior to enable the asynchronous and partial wakeup.

After detecting the START condition on the bus, the TWI will stretch TWCK until the TWI peripheral clock has started. The time required for starting the TWI peripheral depends on which Sleep mode the device is in. After the TWI peripheral clock has started, the TWI releases its TWCK stretching and receives one byte of data (slave address) on the bus. At this time, only a limited part of the device, including the TWI module, receives a clock, thus saving power. If the address phase causes a TWIS address match (and optionally if the first data byte causes data match as well), the entire device is wakened and normal TWI address matching actions are performed. Normal TWI transfer then follows. If the TWI module is not addressed (or if the optional data match fails), the TWI peripheral clock is automatically stopped and the device returns to its original Sleep mode.

The TWI module has the capability to match on more than one address. The SADR1EN, SADR2EN and SADR3EN bits in TWI_SMR allow to enable address matching on additional addresses which can be configured through SADR1, SADR2 and SADR3 fields in the TWI_SWMR. The SleepWalking matching process can be extended to the first received data byte if DATAMEN bit in TWI_SMR is set, in that case a complete matching includes address matching and first received data matching. The DATAM field in TWI_SWMR allows to configure the data to match on the first received byte.

When the system is in Active mode and the TWI enters Asynchronous Partial Wakeup mode, the flag SVACC must be programmed as the unique source of the TWI interrupt and the data match comparison must be disabled.

When the system exits Wait mode as the result of a matching condition, the SVACC flag is used to determine if the TWI is the source of the exit from Wait mode.

Figure 32-39. Address Match Only (Data Matching Disabled)

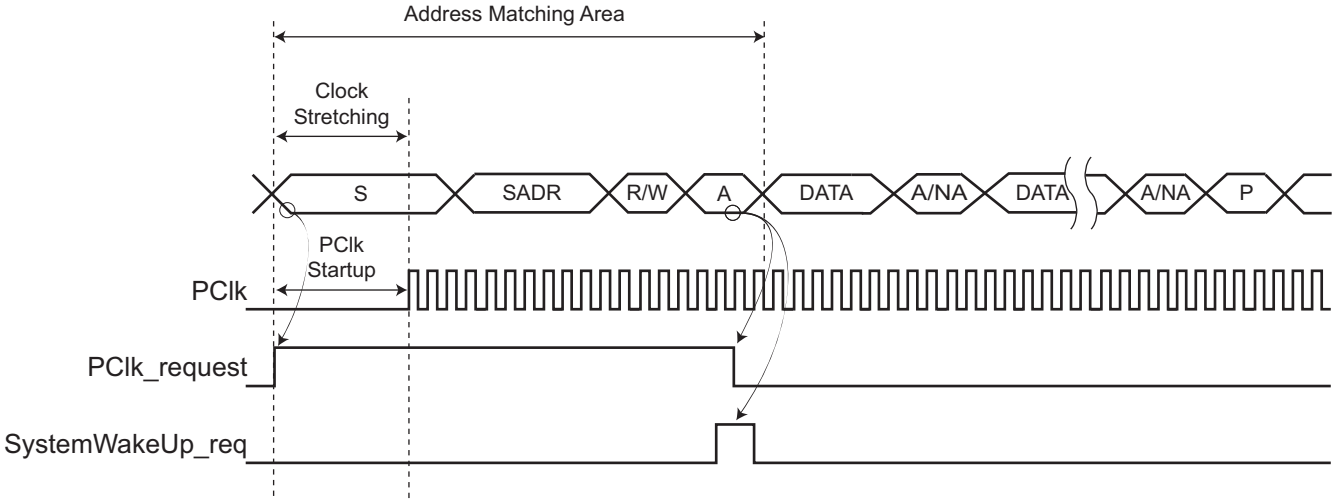


Figure 32-40. No Address Match (Data Matching Disabled)

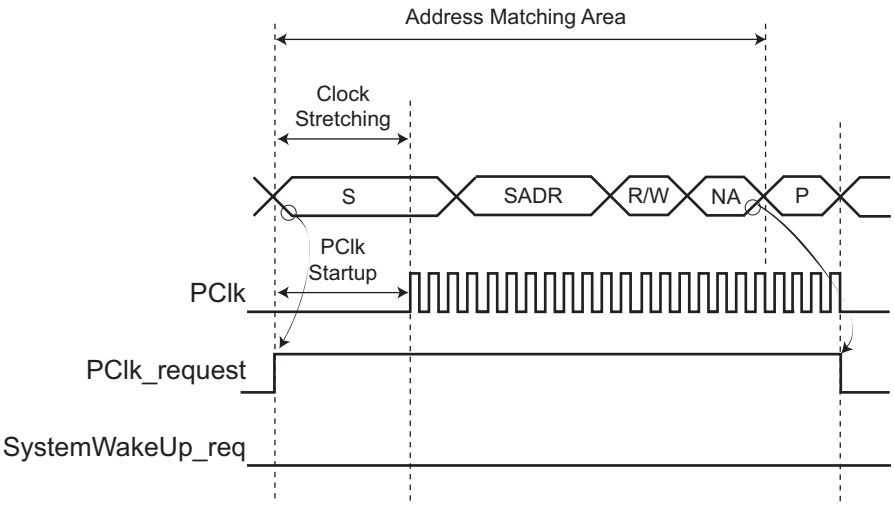


Figure 32-41. Address Match and Data Match (Data Matching Enabled)

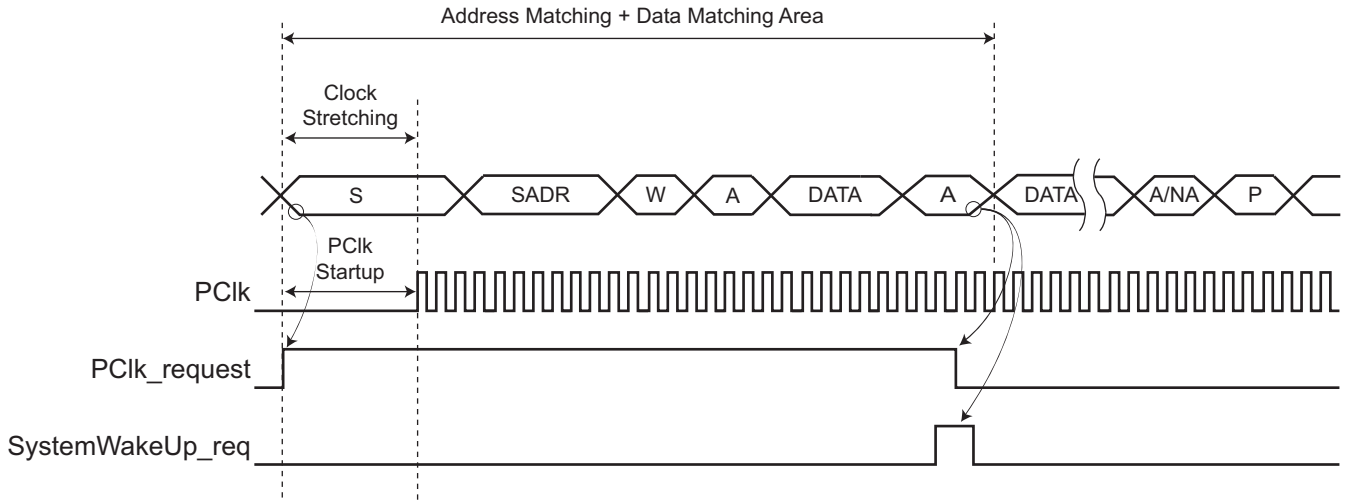
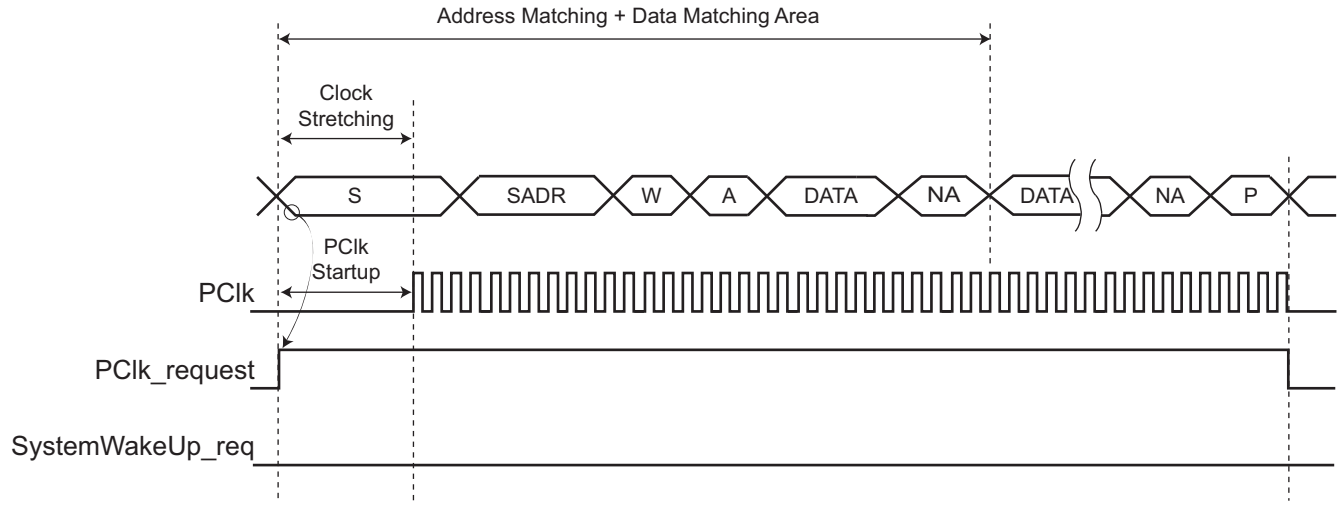


Figure 32-42. Address Match and No Data Match (Data Matching Enabled)



32.6.5.8 Slave Read/Write Flowcharts

The flowchart shown in Figure 32-43 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 TWI_IER be configured first.

Figure 32-43. Read/Write Flowchart in Slave Mode

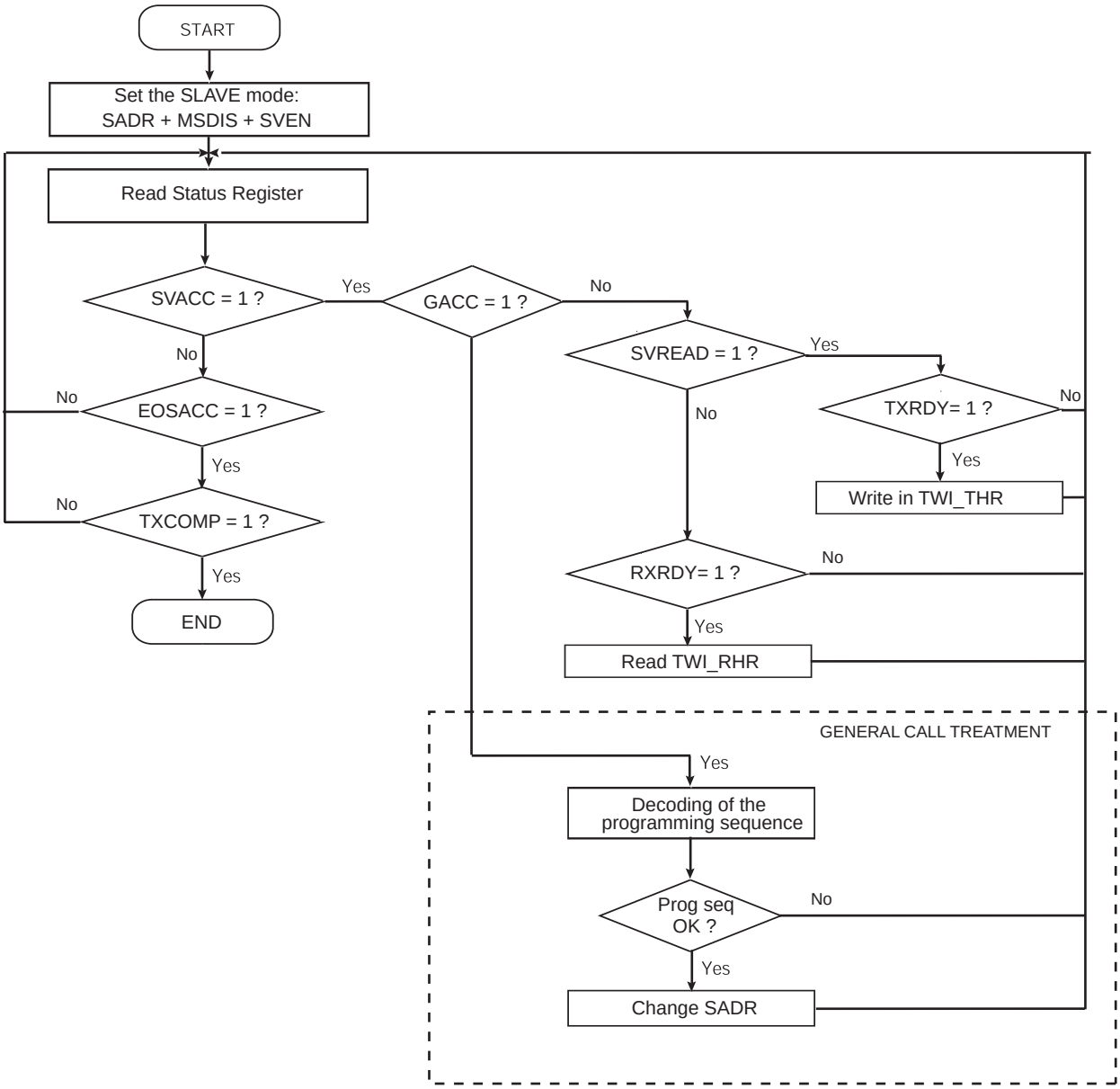


Figure 32-44. Read/Write Flowchart in Slave Mode with SMBus PEC

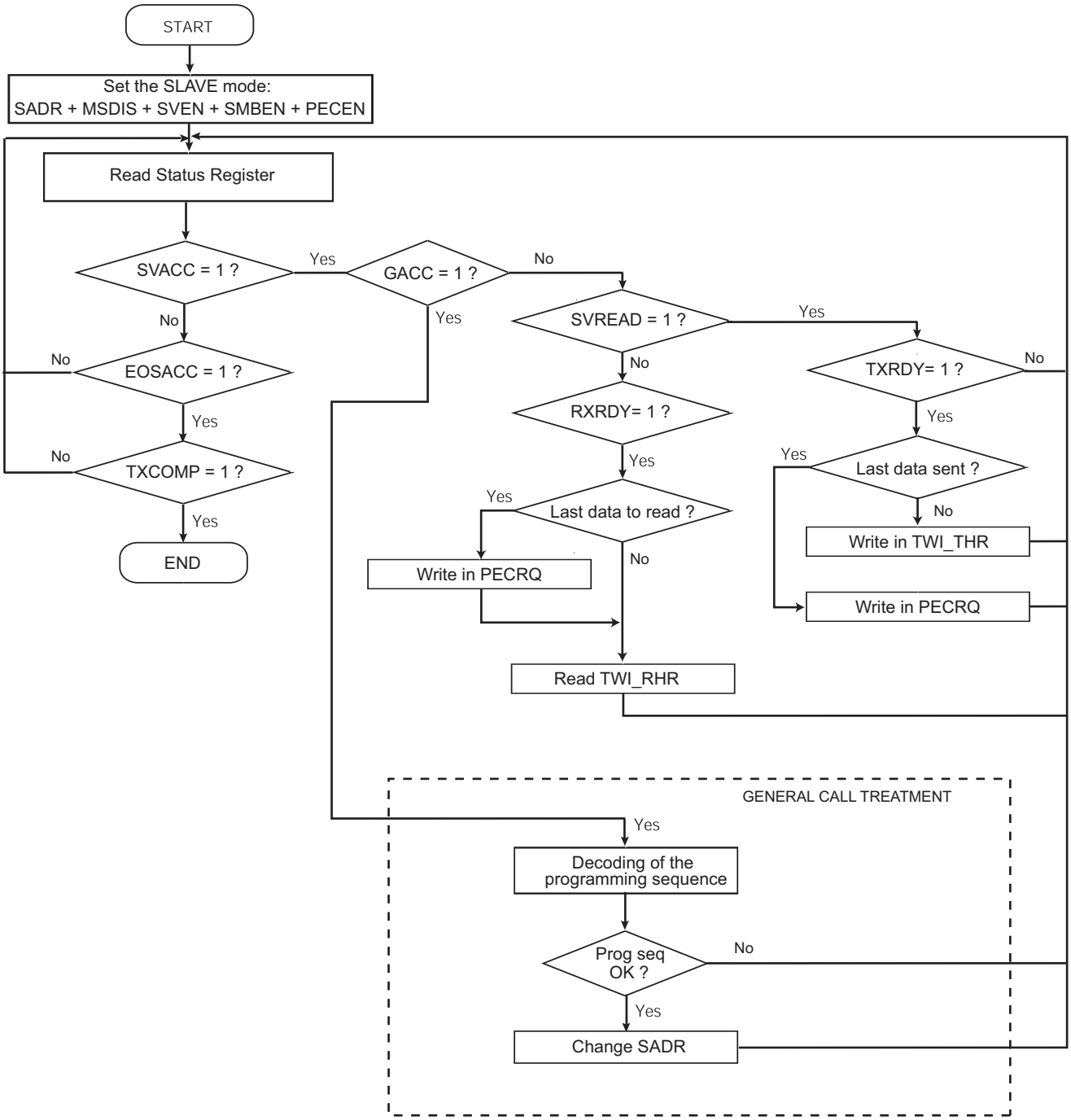
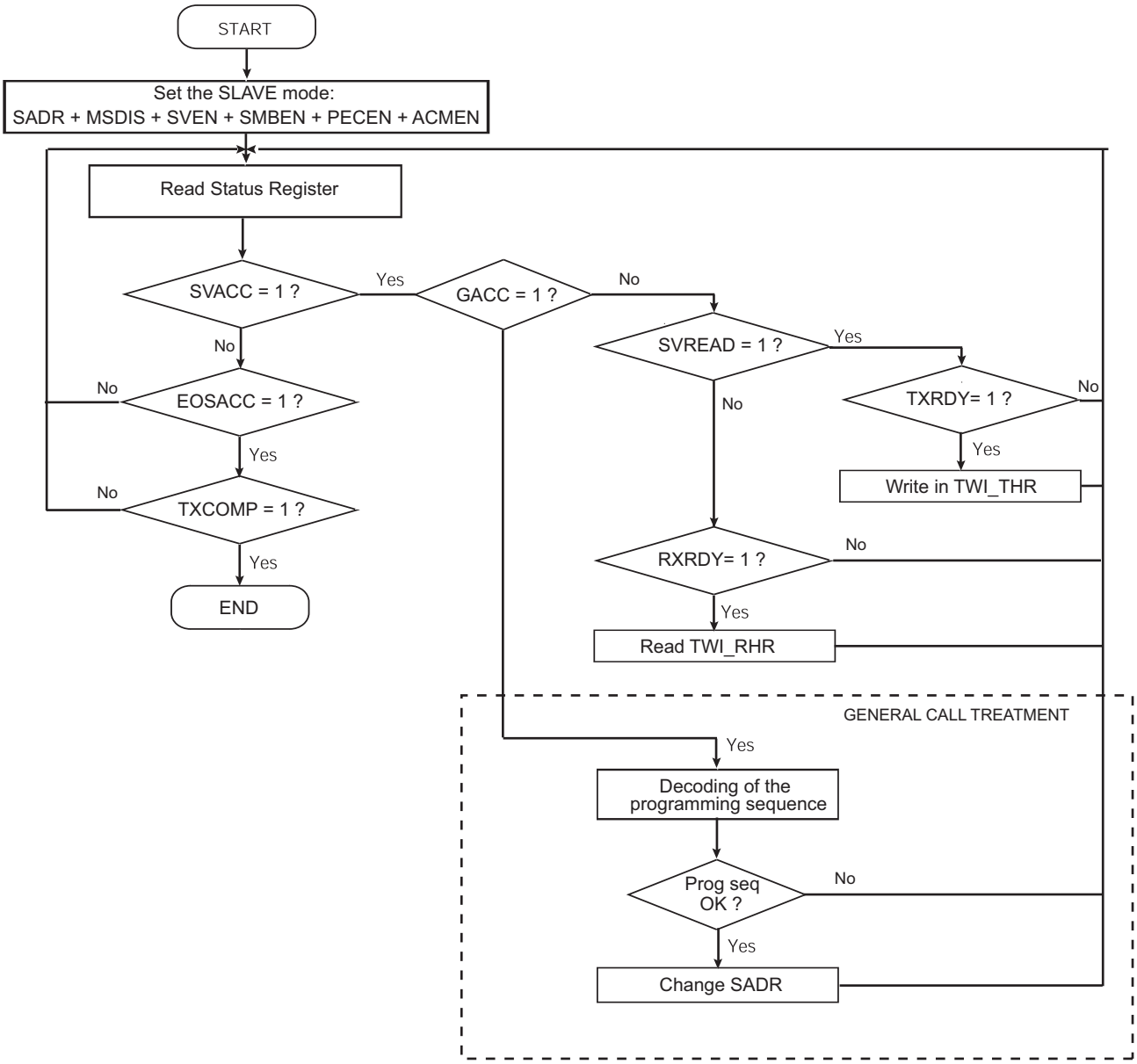


Figure 32-45. Read/Write Flowchart in Slave Mode with SMBus PEC and Alternative Command Mode



32.6.6 TWI Comparison Function on Received Character

The comparison function differs if the asynchronous partial wakeup (Sleepwalking) is enabled or not.

If asynchronous partial wakeup is disabled (see PMC section), the TWI has the capability to extend the address matching on up to three slave addresses. The SADR1EN, SADR2EN and SADR3EN bits in TWI_SMR enable address matching on additional addresses which can be configured through SADR1, SADR2 and SADR3 fields in the TWI_SWMR. The DATAMEN bit had no effect.

The SVACC bit is set when there is a comparison match with the received slave address.

32.6.7 TWI 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 bit 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 TWI_WPSR.

The following registers can be write-protected:

- TWI Slave Mode Register (TWI_SMR)
- TWI Clock Waveform Generator Register (TWI_CWGR)
- TWI SMBus Timing Register (TWI_SMBTR)
- TWI SleepWalking Matching Register (TWI_SWMR)

32.7 Two-wire Interface (TWI) User Interface

Table 32-5. Register Mapping

Offset	Register	Name	Access	Reset
0x000	TWI Control Register	TWI_CR	Write-only	–
0x004	TWI Master Mode Register	TWI_MMR	Read/Write	0x00000000
0x008	TWI Slave Mode Register	TWI_SMR	Read/Write	0x00000000
0x00C	TWI Internal Address Register	TWI_IADR	Read/Write	0x00000000
0x010	TWI Clock Waveform Generator Register	TWI_CWGR	Read/Write	0x00000000
0x014–0x01C	Reserved	–	–	–
0x020	TWI Status Register	TWI_SR	Read-only	0x0300F009
0x024	TWI Interrupt Enable Register	TWI_IER	Write-only	–
0x028	TWI Interrupt Disable Register	TWI_IDR	Write-only	–
0x02C	TWI Interrupt Mask Register	TWI_IMR	Read-only	0x00000000
0x030	TWI Receive Holding Register	TWI_RHR	Read-only	0x00000000
0x034	TWI Transmit Holding Register	TWI_THR	Write-only	–
0x038	TWI SMBus Timing Register	TWI_SMBTR	Read/Write	0x00000000
0x03C	Reserved	–	–	–
0x040	TWI Alternative Command Register	TWI_ACR	Read/Write	0x00000000
0x044	TWI Filter Register	TWI_FILTR	Read/Write	0x00000000
0x048	Reserved	–	–	–
0x04C	TWI SleepWalking Matching Register	TWI_SWMR	Read/Write	0x00000000
0x050–0x0CC	Reserved	–	–	–
0x0D0–0x0E0	Reserved	–	–	–
0x0E4	TWI Write Protection Mode Register	TWI_WPMR	Read/Write	0x00000000
0x0E8	TWI Write Protection Status Register	TWI_WPSR	Read-only	0x00000000
0x0EC–0x0FC	Reserved	–	–	–

32.7.1 TWI Control Register

Name: TWI_CR

Address: 0x4000C600 (0), 0x40020600 (1), 0x40024600 (2), 0x40018600 (3), 0x4001C600 (4), 0x40008600 (5), 0x40040600 (6), 0x40034600 (7)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	LOCKCLR	–	THRCLR
23	22	21	20	19	18	17	16
–	–	–	–	–	–	ACMDIS	ACMEN
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 TWI Master Mode Register (TWI_MMR).

This action is necessary when the TWI peripheral needs 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 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.

- **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.

- **ACMEN: Alternative Command Mode Enable**

0: No effect.

1: Alternative Command mode enabled.

- **ACMDIS: Alternative Command Mode Disable**

0: No effect.

1: Alternative Command mode disabled.

- **THRCLR: Transmit Holding Register Clear**

0: No effect.

1: Clear the Transmit Holding Register and set TXRDY, TXCOMP flags.

- **LOCKCLR: Lock Clear**

0: No effect.

1: Clear the TWI FSM lock.

32.7.2 TWI Master Mode Register

Name: TWI_MMR

Address: 0x4000C604 (0), 0x40020604 (1), 0x40024604 (2), 0x40018604 (3), 0x4001C604 (4), 0x40008604 (5), 0x40040604 (6), 0x40034604 (7)

Access: Read/Write

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

23	22	21	20	19	18	17	16
–	DADR						

15	14	13	12	11	10	9	8
–	–	–	MREAD	–	–	IADRSZ	

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

- **IADRSZ: Internal Device Address Size**

Value	Name	Description
0	NONE	No internal device address
1	1_BYTE	One-byte internal device address
2	2_BYTE	Two-byte internal device address
3	3_BYTE	Three-byte internal device address

- **MREAD: Master Read Direction**

0: Master write direction.

1: Master read direction.

- **DADR: Device Address**

The device address is used to access slave devices in Read or Write mode. Those bits are only used in Master mode.

32.7.3 TWI Slave Mode Register

Name: TWI_SMR

Address: 0x4000C608 (0), 0x40020608 (1), 0x40024608 (2), 0x40018608 (3), 0x4001C608 (4), 0x40008608 (5), 0x40040608 (6), 0x40034608 (7)

Access: Read/Write

31	30	29	28	27	26	25	24
DATAMEN	SADR3EN	SADR2EN	SADR1EN	–	–	–	–
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.

- **SADR1EN: Slave Address 1 Enable**

0: Slave address 1 matching is disabled.

1: Slave address 1 matching is enabled.

- **SADR2EN: Slave Address 2 Enable**

0: Slave address 2 matching is disabled.

1: Slave address 2 matching is enabled.

- **SADR3EN: Slave Address 3 Enable**

0: Slave address 3 matching is disabled.

1: Slave address 3 matching is enabled.

- **DATAMEN: Data Matching Enable**

0: Data matching on first received data is disabled.

1: Data matching on first received data is enabled.

32.7.4 TWI Internal Address Register

Name: TWI_IADR

Address: 0x4000C60C (0), 0x4002060C (1), 0x4002460C (2), 0x4001860C (3), 0x4001C60C (4), 0x4000860C (5), 0x4004060C (6), 0x4003460C (7)

Access: Read/Write

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

- **IADR: Internal Address**

0, 1, 2 or 3 bytes depending on IADRSZ.

32.7.5 TWI Clock Waveform Generator Register

Name: TWI_CWGR

Address: 0x4000C610 (0), 0x40020610 (1), 0x40024610 (2), 0x40018610 (3), 0x4001C610 (4), 0x40008610 (5), 0x40040610 (6), 0x40034610 (7)

Access: Read/Write

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

23	22	21	20	19	18	17	16
–	–	–	BRSRCCLK	–	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 TWI_WPMR.

TWI_CWGR is only used in Master mode.

- **CLDIV: Clock Low Divider**

The TWCK low period is defined as follows:

if BRSRCCLK = 0

$$t_{low} = ((\text{CLDIV} \times 2^{\text{CKDIV}}) + 3) \times t_{\text{peripheral\_clock}}$$

if BRSRCCLK = 1

$$t_{low} = (\text{CLDIV} \times 2^{\text{CKDIV}}) \times t_{\text{PCKx}}$$

- **CHDIV: Clock High Divider**

The TWCK high period is defined as follows:

if BRSRCCLK = 0

$$t_{high} = ((\text{CHDIV} \times 2^{\text{CKDIV}}) + 3) \times t_{\text{peripheral\_clock}}$$

if BRSRCCLK = 1

$$t_{high} = (\text{CHDIV} \times 2^{\text{CKDIV}}) \times t_{\text{PCKx}}$$

- **CKDIV: Clock Divider**

The CKDIV field is used to increase both TWCK high and low periods.

- **BRSRCCLK: Bit Rate Source Clock**

Value	Name	Description
0	PERIPH_CLK	The peripheral clock is the source clock for the bit rate generation.
1	PMC_PCK	PMC PCKx is the source clock for the bit rate generation, thus the bit rate can be independent of the core/peripheral clock.

- **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) $\times$ $t_{peripheral\ clock}$.

32.7.6 TWI Status Register

Name: TWI_SR

Address: 0x4000C620 (0), 0x40020620 (1), 0x40024620 (2), 0x40018620 (3), 0x4001C620 (4), 0x40008620 (5), 0x40040620 (6), 0x40034620 (7)

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	SDA	SCL
23	22	21	20	19	18	17	16
LOCK	–	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 (cleared by writing TWI_THR)**

TXCOMP used in Master mode:

0: During the length of the current frame.

1: When both holding register and internal shifter are empty and STOP condition has been sent.

TXCOMP behavior in Master mode can be seen in Figure 32-6 and in Figure 32-8.

TXCOMP used in Slave mode:

0: As soon as a Start is detected.

1: After a Stop or a Repeated Start + an address different from SADR is detected.

TXCOMP behavior in Slave mode can be seen in Figure 32-34, Figure 32-35, Figure 32-36 and Figure 32-37.

- **RXRDY: Receive Holding Register Ready (cleared when reading TWI_RHR)**

0: No character has been received since the last TWI_RHR read operation.

1: A byte has been received in TWI_RHR since the last read.

RXRDY behavior in Master mode can be seen in Figure 32-8.

RXRDY behavior in Slave mode can be seen in Figure 32-32, Figure 32-35, Figure 32-36 and Figure 32-37.

- **TXRDY: Transmit Holding Register Ready (cleared by writing TWI_THR)**

TXRDY used in Master mode:

0: The transmit holding register has not been transferred into the internal shifter. Set to 0 when writing into TWI_THR.

1: As soon as a data byte is transferred from TWI_THR to the 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 32-4, Figure 32-5, and Figure 32-6.

TXRDY used in Slave mode:

0: As soon as data is written in TWI_THR, until this data has been transmitted and acknowledged (ACK or NACK).

1: Indicates that 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 user must not fill TWI_THR to avoid losing it.

TXRDY behavior in Slave mode can be seen in Figure 32-31, Figure 32-34, Figure 32-36 and Figure 32-37.

- **SVREAD: Slave Read**

This bit is only used in Slave mode. When SVACC is low (no slave access has been detected) SVREAD is irrelevant.

0: Indicates that a write access is performed by a master.

1: Indicates that a read access is performed by a master.

SVREAD behavior can be seen in Figure 32-31, Figure 32-32, Figure 32-36 and Figure 32-37.

- **SVACC: Slave Access**

This bit is only used in Slave mode.

0: TWI is not addressed. SVACC is automatically cleared after a NACK or a STOP condition is detected.

1: Indicates that the address decoding sequence has matched (A master has sent SADR). SVACC remains high until a NACK or a STOP condition is detected.

SVACC behavior can be seen in Figure 32-31, Figure 32-32, Figure 32-36 and Figure 32-37.

- **GACC: General Call Access (cleared on read)**

This bit is only used in Slave mode.

0: No general call has been detected.

1: A general call has been detected. After the detection of general call, if need be, the 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 32-33.

- **OVRE: Overrun Error (cleared on read)**

This bit is only used if clock stretching is disabled.

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.

- **UNRE: Underrun Error (cleared on read)**

This bit is only used if clock stretching is disabled.

0: TWI_THR has been filled on time.

1: TWI_THR has not been filled on time.

- **NACK: Not Acknowledged (cleared on read)**

NACK used in Master mode:

0: Each data byte has been correctly received by the far-end side TWI slave component.

1: A data 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 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 (cleared on read)**

This bit is only used in Master mode.

0: Arbitration won.

1: Arbitration lost. Another master of the TWI bus has won the multi-master arbitration. TXCOMP is set at the same time.

- **SCLWS: Clock Wait State**

This bit is only used in Slave mode.

0: The clock is not stretched.

1: The clock is stretched. The TWI_THR / TWI_RHR buffer is not filled / emptied before the emission / reception of a new character.

SCLWS behavior can be seen in Figure 32-34 and Figure 32-35.

- **EOSACC: End Of Slave Access (cleared 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 32-36 and Figure 32-37.

- **ENDRX: End of RX Buffer (cleared by writing TWI_RCR or TWI_RNCR)**

0: The Receive Counter Register has not reached 0 since the last write in TWI_RCR or TWI_RNCR(1).

1: The Receive Counter Register has reached 0 since the last write in TWI_RCR or TWI_RNCR(1).

- **ENDTX: End of TX Buffer (cleared by writing TWI_TCR or TWI_TNCR)**

0: The Transmit Counter Register has not reached 0 since the last write in TWI_TCR or TWI_TNCR(1).

1: The Transmit Counter Register has reached 0 since the last write in TWI_TCR or TWI_TNCR(1).

- **RXBUFF: RX Buffer Full (cleared by writing TWI_RCR or TWI_RNCR)**

0: TWI_RCR or TWI_RNCR have a value other than 0(1).

1: Both TWI_RCR and TWI_RNCR have a value of 0(1).

- **TXBUFE: TX Buffer Empty (cleared by writing TWI_TCR or TWI_TNCR)**

0: TWI_TCR or TWI_TNCR have a value other than 0(1).

1: Both TWI_TCR and TWI_TNCR have a value of 0(1).

Note: 1. TWI_RCR, TWI_RNCR, TWI_TCR and TWI_TNCR are PDC registers.

- **MCACK: Master Code Acknowledge (cleared on read)**

MACK used in Slave mode:

0: No Master Code has been received.

1: A Master Code has been received.

- **TOUT: Timeout Error (cleared on read)**

0: No SMBus timeout occurred.

1: SMBus timeout occurred.

- **PECERR: PEC Error (cleared on read)**

0: No SMBus PEC error occurred.

1: An SMBus PEC error occurred.

- **SMBDAM: SMBus Default Address Match (cleared on read)**

0: No SMBus Default Address received.

1: An SMBus Default Address was received.

- **SMBHHM: SMBus Host Header Address Match (cleared on read)**

0: No SMBus Host Header Address received.

1: An SMBus Host Header Address was received.

- **LOCK: TWI Lock Due to Frame Errors**

0: The TWI is not locked.

1: The TWI is locked due to frame errors (see “Handling Errors in Alternative Command” on page 794).

- **SCL: Serial Clock (TWCK) Line Value**

0: SCL line sampled value is ‘0’.

1: SCL line sampled value is ‘1.’

- **SDA: Serial Data (TWD) Line Value**

0: SDA line sampled value is ‘0’.

1: SDA line sampled value is ‘1’.

32.7.7 TWI SMBus Timing Register

Name: TWI_SMBTR

Address: 0x4000C638 (0), 0x40020638 (1), 0x40024638 (2), 0x40018638 (3), 0x4001C638 (4), 0x40008638 (5), 0x40040638 (6), 0x40034638 (7)

Access: Read/Write

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.

32.7.8 TWI Alternative Command Register

Name: TWI_ACR

Address: 0x4000C640 (0), 0x40020640 (1), 0x40024640 (2), 0x40018640 (3), 0x4001C640 (4), 0x40008640 (5), 0x40040640 (6), 0x40034640 (7)

Access: Read/Write

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

23	22	21	20	19	18	17	16
NDATAL							

15	14	13	12	11	10	9	8
–	–	–	–	–	–	PEC	DIR

7	6	5	4	3	2	1	0
DATAL							

- **DATAL: Data Length**

0: No data to send (see Section 32.6.3.12 ”Alternative Command”).

1–255: Number of bytes to send during the transfer.

- **DIR: Transfer Direction**

0: Write direction.

1: Read direction.

- **PEC: PEC Request (SMBus Mode only)**

0: The transfer does not use a PEC byte.

1: The transfer uses a PEC byte.

- **NDATAL: Next Data Length**

0: No data to send (see Section 32.6.3.12 ”Alternative Command”).

1–255: Number of bytes to send for the next transfer.

- **NDIR: Next Transfer Direction**

0: Write direction.

1: Read direction.

- **NPEC: Next PEC Request (SMBus Mode only)**

0: The next transfer does not use a PEC byte.

1: The next transfer uses a PEC byte.

32.7.9 TWI Filter Register

Name: TWI_FILTR

Address: 0x4000C644 (0), 0x40020644 (1), 0x40024644 (2), 0x40018644 (3), 0x4001C644 (4), 0x40008644 (5), 0x40040644 (6), 0x40034644 (7)

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
–	–	–	–	–	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 peripheral clock 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 Configuration**

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 peripheral clock cycles.

32.7.10 TWI Interrupt Enable Register

Name: TWI_IER

Address: 0x4000C624 (0), 0x40020624 (1), 0x40024624 (2), 0x40018624 (3), 0x4001C624 (4), 0x40008624 (5), 0x40040624 (6), 0x40034624 (7)

Access: Write-only

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

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**
- **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**

32.7.11 TWI Interrupt Disable Register

Name: TWI_IDR

Address: 0x4000C628 (0), 0x40020628 (1), 0x40024628 (2), 0x40018628 (3), 0x4001C628 (4), 0x40008628 (5), 0x40040628 (6), 0x40034628 (7)

Access: Write-only

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

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**
- **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**

32.7.12 TWI Interrupt Mask Register

Name: TWI_IMR

Address: 0x4000C62C (0), 0x4002062C (1), 0x4002462C (2), 0x4001862C (3), 0x4001C62C (4), 0x4000862C (5), 0x4004062C (6), 0x4003462C (7)

Access: Read-only

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

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**
- **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**

32.7.13 TWI Receive Holding Register

Name: TWI_RHR

Address: 0x4000C630 (0), 0x40020630 (1), 0x40024630 (2), 0x40018630 (3), 0x4001C630 (4), 0x40008630 (5), 0x40040630 (6), 0x40034630 (7)

Access: Read-only

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
RXDATA							

- **RXDATA: Master or Slave Receive Holding Data**

32.7.14 TWI SleepWalking Matching Register

Name: TWI_SWMR

Address: 0x4000C64C (0), 0x4002064C (1), 0x4002464C (2), 0x4001864C (3), 0x4001C64C (4), 0x4000864C (5), 0x4004064C (6), 0x4003464C (7)

Access: Read/Write

31	30	29	28	27	26	25	24
DATAM							

23	22	21	20	19	18	17	16
–	SADR3						

15	14	13	12	11	10	9	8
–	SADR2						

7	6	5	4	3	2	1	0
–	SADR1						

- **SADR1: Slave Address 1**

Slave address 1. The TWI module will match on this additional address if SADR1EN bit is enabled.

- **SADR2: Slave Address 2**

Slave address 2. The TWI module will match on this additional address if SADR2EN bit is enabled.

- **SADR3: Slave Address 3**

Slave address 3. The TWI module will match on this additional address if SADR3EN bit is enabled.

- **DATAM: Data Match**

The TWI module will extend the SleepWalking matching process to the first received data comparing it with DATAM if DATAMEN bit is enabled.

32.7.15 TWI Transmit Holding Register

Name: TWI_THR

Address: 0x4000C634 (0), 0x40020634 (1), 0x40024634 (2), 0x40018634 (3), 0x4001C634 (4), 0x40008634 (5), 0x40040634 (6), 0x40034634 (7)

Access: Write-only

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
TXDATA							

- **TXDATA: Master or Slave Transmit Holding Data**

32.7.16 TWI Write Protection Mode Register

Name: TWI_WPMR

Address: 0x4000C6E4 (0), 0x400206E4 (1), 0x400246E4 (2), 0x400186E4 (3), 0x4001C6E4 (4), 0x400086E4 (5), 0x400406E4 (6), 0x400346E4 (7)

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 32.6.7 ”TWI 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

32.7.17 TWI Write Protection Status Register

Name: TWI_WPSR

Address: 0x4000C6E8 (0), 0x400206E8 (1), 0x400246E8 (2), 0x400186E8 (3), 0x4001C6E8 (4), 0x400086E8 (5), 0x400406E8 (6), 0x400346E8 (7)

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 TWI_WPSR.

1: A Write Protection Violation has occurred since the last read of 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.

33. Inter-IC Sound Controller (I2SC)

33.1 Description

The Inter-IC Sound Controller (I2SC) provides a 5-wire, bidirectional, synchronous, digital audio link to external audio devices: I2SDI, I2SDO, I2SWS, I2SCK, and I2SMCK pins.

The I2SC is compliant with the Inter-IC Sound (I^2S) bus specification.

The I2SC consists of a receiver, a transmitter and a common clock generator that can be enabled separately to provide Master, Slave or Controller modes with receiver and/or transmitter active.

Peripheral DMA Controller (PDC) channels, separate for the receiver and for the transmitter, allow a continuous high bit rate data transfer without processor intervention to the following:

- Audio CODECs in Master, Slave, or Controller mode
- Stereo DAC or ADC through a dedicated I^2S serial interface

The I2SC can use either a single PDC channel for both audio channels or one PDC channel per audio channel.

The 8- and 16-bit compact stereo format reduces the required PDC bandwidth by transferring the left and right samples within the same data word.

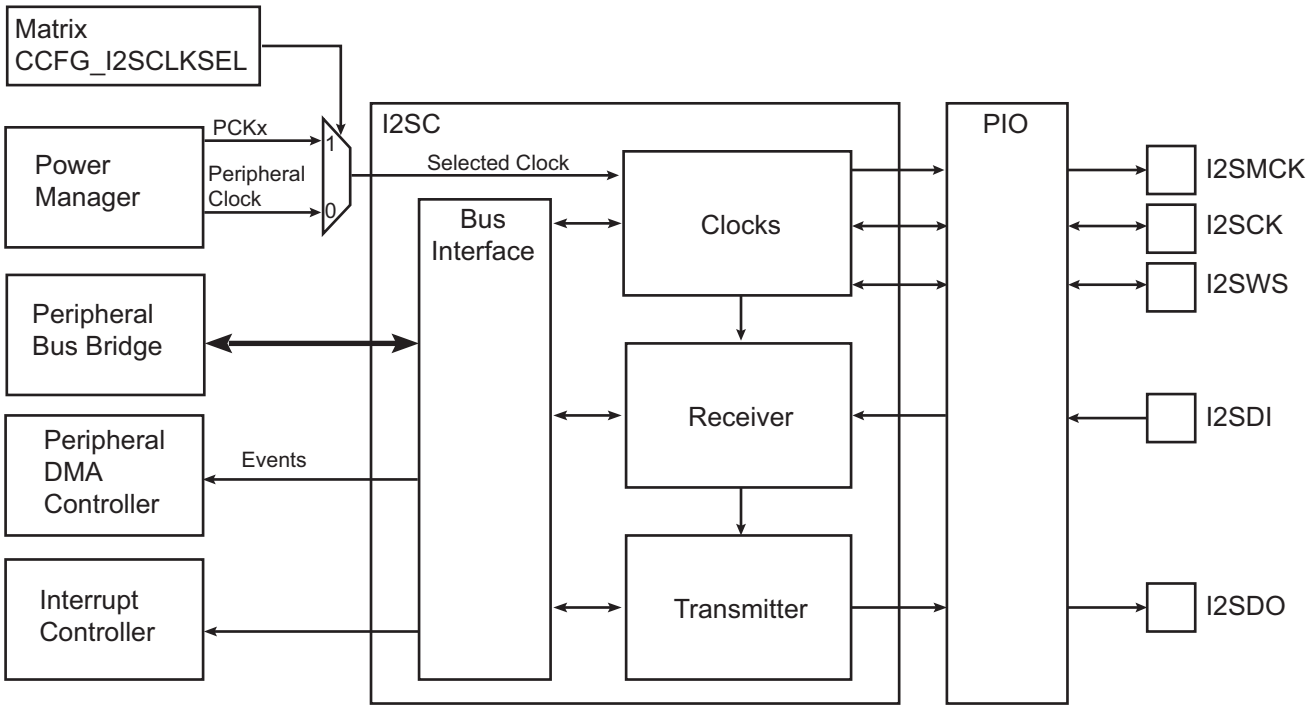
In Master mode, the I2SC can produce a 32 f_s to 1024 f_s master clock that provides an over-sampling clock to an external audio codec or digital signal processor (DSP).

33.2 Embedded Characteristics

- Compliant with Inter-IC Sound (I^2S) Bus Specification
- Master, Slave, and Controller Modes
  - Slave: Data Received/Transmitted
  - Master: Data Received/Transmitted And Clocks Generated
  - Controller: Clocks Generated
- Individual Enable and Disable of Receiver, Transmitter and Clocks
- Configurable Clock Generator Common to Receiver and Transmitter
  - Suitable for a Wide Range of Sample Frequencies (f_s), Including 32 kHz, 44.1 kHz, 48 kHz, 88.2 kHz, 96 kHz, and 192 kHz
  - 32 f_s to 1024 f_s Master Clock Generated for External Oversampling Data Converters
- Support for Multiple Data Formats
  - 32-, 24-, 20-, 18-, 16-, and 8-bit Mono or Stereo Format
  - 16- and 8-bit Compact Stereo Format, with Left and Right Samples Packed in the Same Word to Reduce Data Transfers
- PDC Interfaces the Receiver and Transmitter to Reduce Processor Overhead
  - One PDC Channel for Both Audio Channels, or
  - One PDC Channel Per Audio Channel
- Smart Holding Registers Management to Avoid Audio Channels Mix After Overrun or Underrun

33.3 Block Diagram

Figure 33-1. I2SC Block Diagram



33.4 I/O Lines Description

Table 33-1. I/O Lines Description

Pin Name	Pin Description	Type
I2SMCK	Master Clock	Output
I2SCK	Serial Clock	Input/Output
I2SWS	I^2S Word Select	Input/Output
I2SDI	Serial Data Input	Input
I2SDO	Serial Data Output	Output

33.5 Product Dependencies

To use the I2SC, other parts of the system must be configured correctly, as described below.

33.5.1 I/O Lines

The I2SC pins may be multiplexed with I/O Controller lines. The user must first program the PIO Controller to assign the required I2SC pins to their peripheral function. If the I2SC I/O lines are not used by the application, they can be used for other purposes by the PIO Controller. The user must enable the I2SC inputs and outputs that are used.

Table 33-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
I2SC0	I2SC0_CK	PA0	A
I2SC0	I2SC0_DI0	PA2	B
I2SC0	I2SC0_DO0	PA3	B
I2SC0	I2SC0_DO0	PA17	A
I2SC0	I2SC0_MCK	PA4	B
I2SC0	I2SC0_MCK	PA18	A
I2SC0	I2SC0_WS	PA1	A
I2SC1	I2SC1_CK	PA19	B
I2SC1	I2SC1_DI0	PA22	B
I2SC1	I2SC1_DO0	PA23	A
I2SC1	I2SC1_DO0	PA25	B
I2SC1	I2SC1_MCK	PA24	A
I2SC1	I2SC1_MCK	PA26	B
I2SC1	I2SC1_WS	PA20	B

33.5.2 Power Management

If the CPU enters a Sleep mode that disables clocks used by the I2SC, the I2SC stops functioning and resumes operation after the system wakes up from Sleep mode.

33.5.3 Clocks

The clock for the I2SC bus interface is generated by the Power Management Controller (PMC). I2SC must be disabled before disabling the clock to avoid freezing the I2SC in an undefined state.

33.5.4 Peripheral DMA Controller

The I2SC interfaces to the Peripheral DMA Controller (PDC). Using the I2SC DMA functionality requires the PDC to be programmed first.

33.5.5 Interrupt Sources

The I2SC interrupt line is connected to the Interrupt Controller. Using the I2SC interrupt requires the Interrupt Controller to be programmed first.

Table 33-3. Peripheral IDs

Instance	ID
I2SC0	16
I2SC1	17

33.6 Functional Description

33.6.1 Initialization

The I2SC features a receiver, a transmitter and a clock generator for Master and Controller modes. Receiver and transmitter share the same serial clock and word select.

Before enabling the I2SC, the selected configuration must be written to the I2SC Mode Register (I2SC_MR) and to the I2S Clock Source Selection register (CCFG_I2SCLKSEL) described in the MATRIX section. If the I2SC_MR.IMCKMODE bit is set, the I2SC_MR.IMCKFS field must be configured as described in Section 33.6.5 ”Serial Clock and Word Select Generation”.

Once the I2SC_MR has been written, the I2SC clock generator, receiver, and transmitter can be enabled by writing a ’1’ to the CKEN, RXEN, and TXEN bits in the Control Register (I2SC_CR). The clock generator can be enabled alone in Controller mode to output clocks to the I2SMCK, I2SCK, and I2SWS pins. The clock generator must also be enabled if the receiver or the transmitter is enabled.

The clock generator, receiver, and transmitter can be disabled independently by writing a ’1’ to I2SC_CR.CXDIS, I2SC_CR.RXDIS and/or I2SC_CR.TXDIS, respectively. Once requested to stop, they stop only when the transmission of the pending frame transmission is completed.

33.6.2 Basic Operation

The receiver can be operated by reading the Receiver Holding Register (I2SC_RHR), whenever the Receive Ready (RXRDY) bit in the Status Register (I2SC_SR) is set. Successive values read from RHR correspond to the samples from the left and right audio channels for the successive frames.

The transmitter can be operated by writing to the Transmitter Holding Register (I2SC_THR), whenever the Transmit Ready (TXRDY) bit in the I2SC_SR is set. Successive values written to THR correspond to the samples from the left and right audio channels for the successive frames.

The RXRDY and TXRDY bits can be polled by reading the I2SC_SR.

The I2SC processor load can be reduced by enabling interrupt-driven operation. The RXRDY and/or TXRDY interrupt requests can be enabled by writing a ’1’ to the corresponding bit in the Interrupt Enable Register (I2SC_IER). The interrupt service routine associated to the I2SC interrupt request is executed whenever the Receive Ready or the Transmit Ready status bit is set.

33.6.3 Master, Controller and Slave Modes

In Master and Controller modes, the I2SC provides the master clock, the serial clock and the word select. I2SMCK, I2SCK, and I2SWS pins are outputs.

In Controller mode, the I2SC receiver and transmitter are disabled. Only the clocks are enabled and used by an external receiver and/or transmitter.

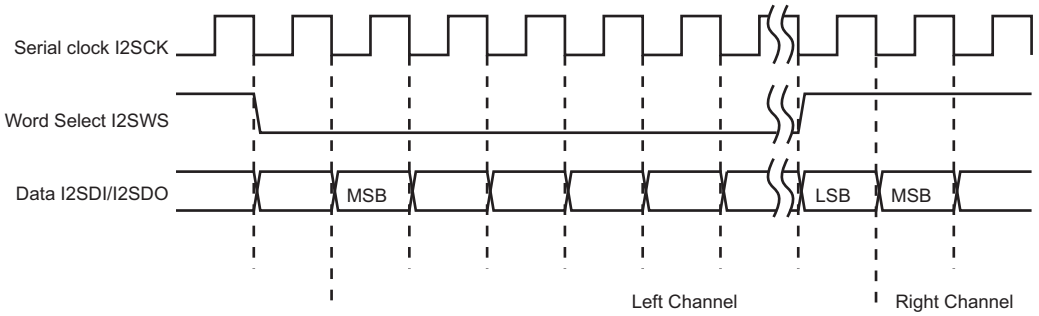
In Slave mode, the I2SC receives the serial clock and the word select from an external master. I2SCK and I2SWS pins are inputs.

The mode is selected by writing the MODE field in the I2SC_MR. Since the MODE field changes the direction of the I2SWS and I2SSCK pins, the I2SC_MR must be written when the I2SC is stopped.

33.6.4 I^2S Reception and Transmission Sequence

As specified in the I^2S protocol, data bits are left-justified in the word select time slot, with the MSB transmitted first, starting one clock period after the transition on the word select line.

Figure 33-2. I²S Reception and Transmission Sequence



Data bits are sent on the falling edge of the serial clock and sampled on the rising edge of the serial clock. the word select line indicates the channel in transmission, a low level for the left channel and a high level for the right channel.

The length of transmitted words can be chosen among 8, 16, 18, 20, 24, and 32 bits by writing the I2SC_MR.DATALENGTH field.

If the time slot allows for more data bits than written in the I2SC_MR.DATALENGTH field, zeroes are appended to the transmitted data word or extra received bits are discarded.

33.6.5 Serial Clock and Word Select Generation

The generation of clocks in the I2SC is described in Figure 33-3 ”I2SC Clock Generation”.

In Slave mode, the serial clock and word select clock are driven by an external master. I2SCK and I2SWS pins are inputs.

In Master mode, the user can configure the master clock, serial clock, and word select clock through the I2SC_MR. I2SMCK, I2SCK, and I2SWS pins are outputs and MCK is used to derive the I2SC clocks.
In Master mode, if the Peripheral clock frequency is higher than 96 MHz, the PCKx clock from PMC must be selected as I2SC input clock by writing a ’1’ in the CLKSELx bit of the CCFG_I2CLKSEL register located in Matrix (See Figure 33-3 ”I2SC Clock Generation”).

Audio codecs connected to the I2SC pins may require a master clock (I2SMCK) signal with a frequency multiple of the audio sample frequency (f_s), such as $256f_s$. When the I2SC is in Master mode, writing a ’1’ to I2SC_MR.IMCKMODE outputs MCK as master clock to the I2SMCK pin, and divides MCK to create the internal bit clock, output on the I2SCK pin. The clock division factor is defined by writing to I2SC_MR.IMCKFS and I2SC_MR.DATALENGTH, as described in the I2SC_MR.IMCKFS field description.

The master clock (I2SMCK) frequency is [2×16 × (IMCKFS + 1)] / (IMCKDIV + 1) times the sample frequency (f_s), i.e., I2SWS frequency.

Example: If the sampling rate is 44.1 kHz with an I2S master clock (I2SMCK) ratio of 256, the core frequency must be an integer multiple of 11.2896 MHz. Assuming an integer multiple of 4, the IMCKDIV field must be configured to 4; the field IMCKFS must then be set to 31.

The serial clock (I2SCK) frequency is 2 × Slot Length times the sample frequency (f_s), where Slot Length is defined in Table 33-4.

Table 33-4. Slot Length

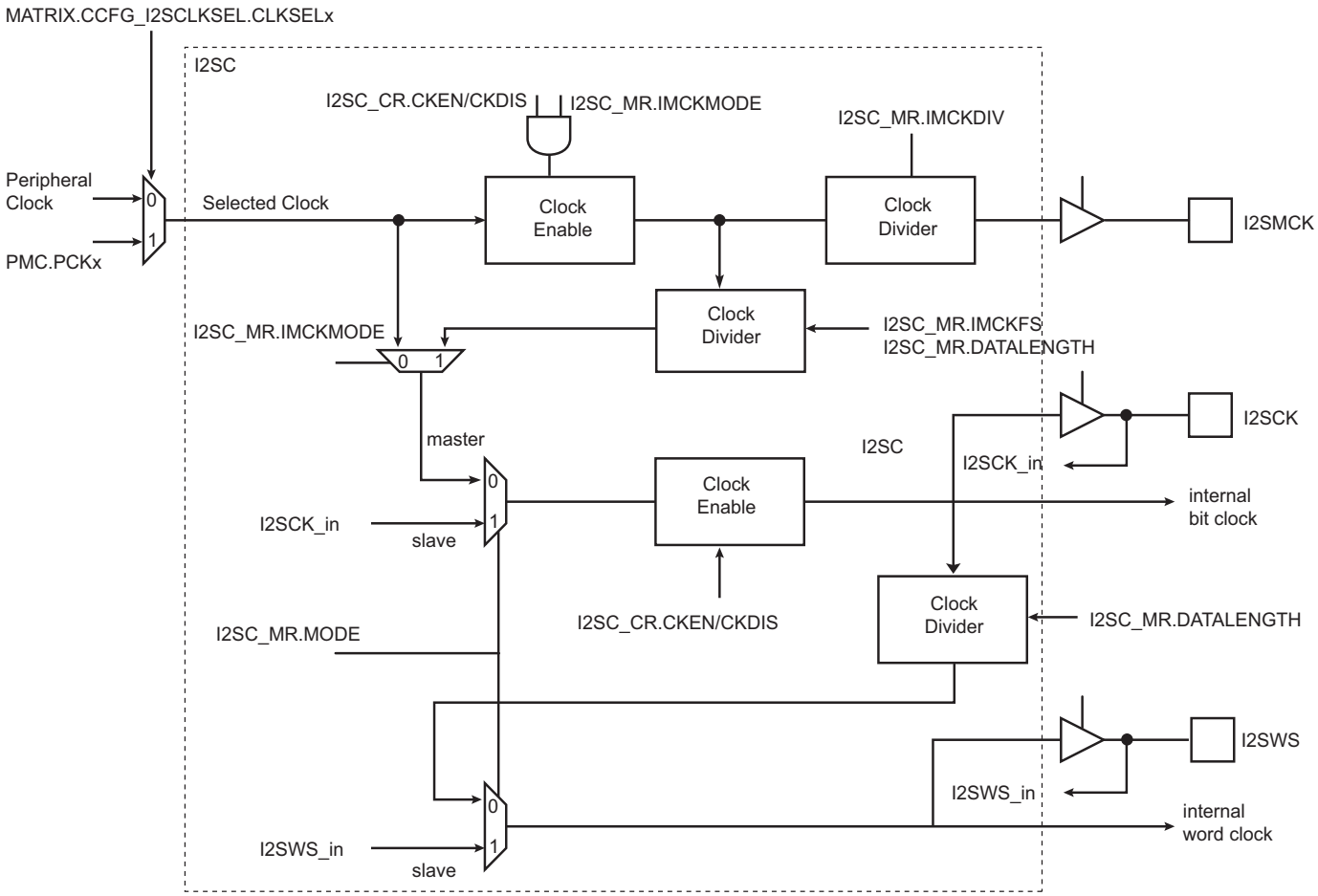
I2SC_MR.DATALENGTH	Word Length	Slot Length
0	32 bits	32
1	24 bits	32 if I2SC_MR.IWS = 0 24 if I2SC_MR.IWS = 1
2	20 bits	
3	18 bits	
4	16 bits	16
5	16 bits compact stereo	
6	8 bits	8
7	8 bits compact stereo	

Warning: I2SC_MR.IMCKMODE must be written to ’1’ if the master clock frequency is strictly higher than the serial clock.

If a master clock output is not required, the MCK clock is used as I2SCK by clearing I2SC_MR.IMCKMODE. Alternatively, if the frequency of the MCK clock used is a multiple of the required I2SCK frequency, the I2SMCK to I2SCK divider can be used with the ratio defined by writing the I2SC_MR.IMCKFS field.

The I2SWS pin is used as word select as described in Section 33.6.4 ”I2S Reception and Transmission Sequence”.

Figure 33-3. I2SC Clock Generation



33.6.6 Mono

When the Transmit Mono bit (TXMONO) in I2SC_MR is set, data written to the left channel is duplicated to the right output channel.

When the Receive Mono bit (RXMONO) in I2SC_MR is set, data received from the left channel is duplicated to the right channel.

33.6.7 Holding Registers

The I2SC user interface includes a Receive Holding Register (I2SC_RHR) and a Transmit Holding Register (I2SC_THR). These registers are used to access audio samples for both audio channels.

When a new data word is available in I2SC_RHR, the Receive Ready bit (RXRDY) in I2SC_SR is set. Reading I2SC_RHR clears this bit.

A receive overrun condition occurs if a new data word becomes available before the previous data word has been read from I2SC_RHR. In this case, the Receive Overrun bit in I2SC_SR and bit i of the RXORCH field in I2SC_SR are set, where i is the current receive channel number.

When I2SC_THR is empty, the Transmit Ready bit (TXRDY) in I2SC_SR is set. Writing to I2SC_THR clears this bit.

A transmit underrun condition occurs if a new data word needs to be transmitted before it has been written to I2SC_THR. In this case, the Transmit Underrun (TXUR) bit and bit i of the TXORCH field in I2SC_SR are set, where i is the current transmit channel number. If the TXSAME bit in I2SC_MR is ’0’, then a zero data word is

transmitted in case of underrun. If I2SC_MR.TXSAME is ’1’, then the previous data word for the current transmit channel number is transmitted.

Data words are right-justified in I2SC_RHR and I2SC_THR. For the 16-bit compact stereo data format, the left sample uses bits 15:0 and the right sample uses bits 31:16 of the same data word. For the 8-bit compact stereo data format, the left sample uses bits 7:0 and the right sample uses bits 15:8 of the same data word.

33.6.8 Peripheral DMA Controller Operation

All receiver audio channels can be assigned to a single PDC channel or individual audio channels can be assigned to one PDC channel per audio channel. The same channel assignment choice applies to the transmitter audio channels.

Channel assignment is selected by writing to the I2SC_MR.RXDMA and I2SC_MR.TXDMA bits. If a single PDC channel is selected, all data samples use I2SC receiver or transmitter PDC channel 0.

The PDC reads from the I2SC_RHR and writes to the I2SC_THR for both audio channels successively.

The PDC transfers may use 32-bit word, 16-bit halfword, or 8-bit byte depending on the value of the I2SC_MR.DATALENGTH field.

33.6.9 Loop-back Mode

For debug purposes, the I2SC can be configured to loop back the transmitter to the Receiver. Writing a ’1’ to the I2SC_MR.LOOP bit internally connects I2SDO to I2SDI, so that the transmitted data is also received. Writing a ’0’ to I2SC_MR.LOOP restores the normal behavior with independent Receiver and Transmitter. As for other changes to the Receiver or Transmitter configuration, the I2SC Receiver and Transmitter must be disabled before writing to I2SC_MR to update I2SC_MR.LOOP.

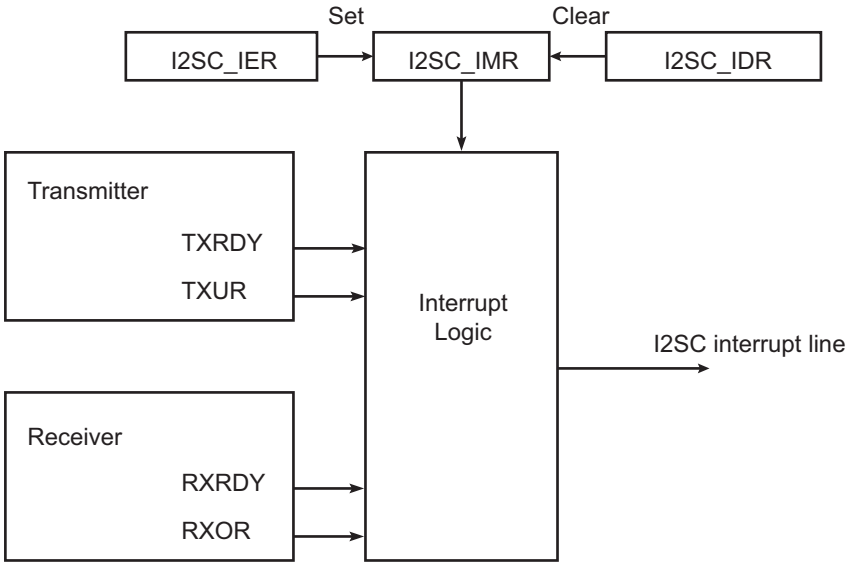
33.6.10 Interrupts

An I2SC interrupt request can be triggered whenever one or several of the following bits are set in I2SC_SR: Receive Ready (RXRDY), Receive Overrun (RXOR), Transmit Ready (TXRDY) or Transmit Underrun (TXUR).

The interrupt request is generated if the corresponding bit in the Interrupt Mask Register (I2SC_IMR) is set. Bits in I2SC_IMR are set by writing a ’1’ to the corresponding bit in I2SC_IER and cleared by writing a ’1’ to the corresponding bit in the Interrupt Disable Register (I2SC_IDR). The interrupt request remains active until the corresponding bit in I2SC_SR is cleared by writing a ’1’ to the corresponding bit in the Status Clear Register (I2SC_SCR).

For debug purposes, interrupt requests can be simulated by writing a ’1’ to the corresponding bit in the Status Set Register (I2SC_SSR).

Figure 33-4. Interrupt Block Diagram



33.7 I2SC Application Examples

The I2SC supports several serial communication modes used in audio or high-speed serial links. Examples of standard applications are shown in the following figures. All serial link applications supported by the I2SC are not listed here.

Figure 33-5. Slave Transmitter I2SC Application Example

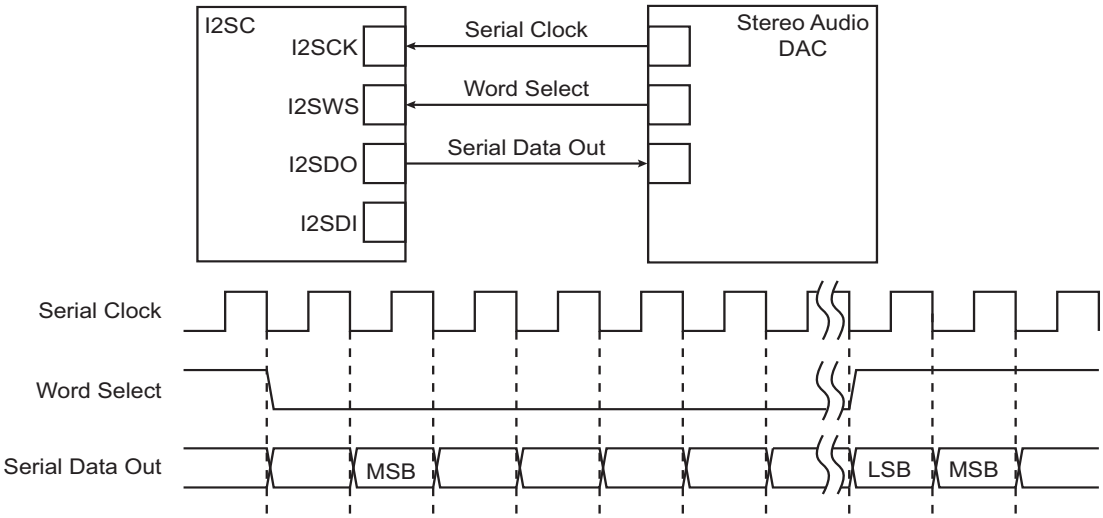


Figure 33-6. Dual Microphone Application Block Diagram

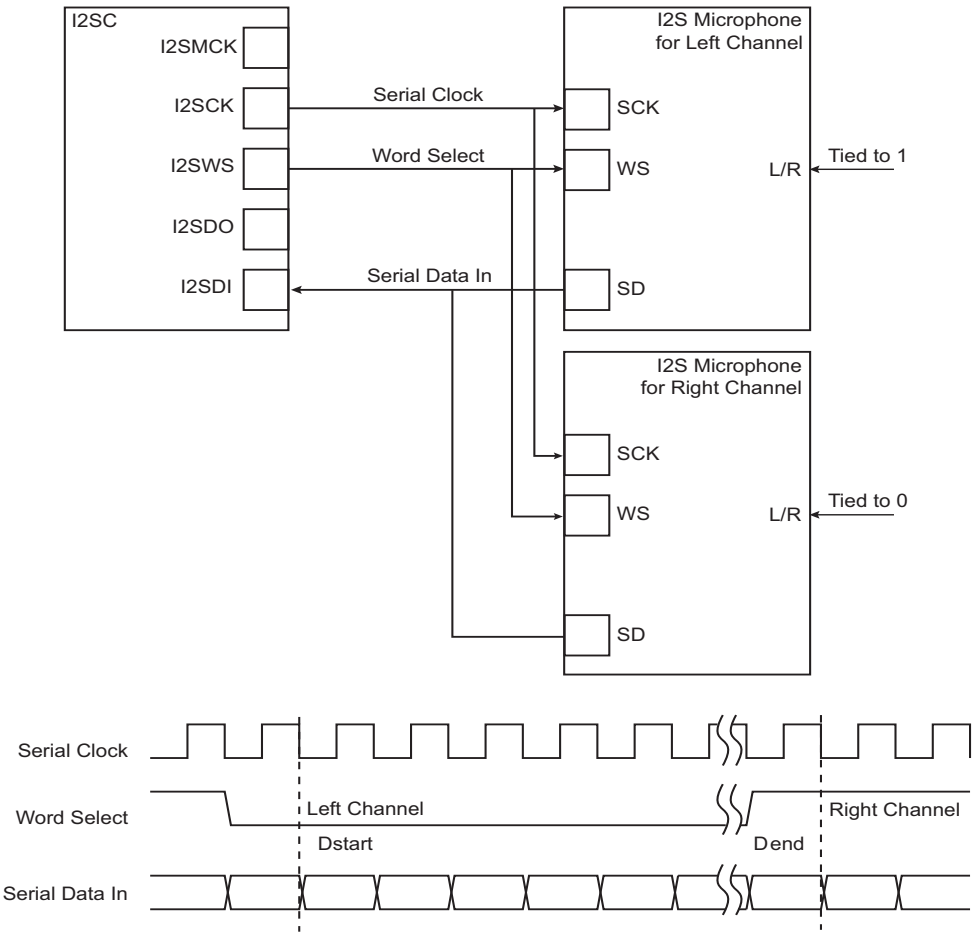
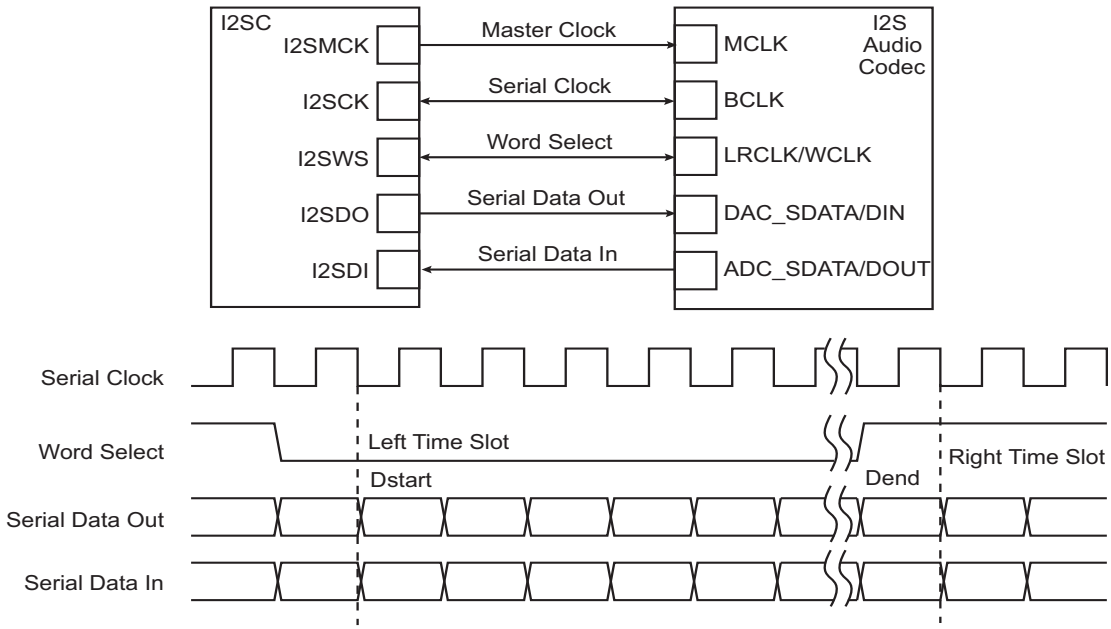


Figure 33-7. Codec Application Block Diagram



33.8 Inter-IC Sound Controller (I2SC) User Interface

Table 33-5. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	I2SC_CR	Write-only	–
0x04	Mode Register	I2SC_MR	Read/Write	0x00000000
0x08	Status Register	I2SC_SR	Read-only	0x00000000
0x0C	Status Clear Register	I2SC_SCR	Write-only	–
0x10	Status Set Register	I2SC_SSR	Write-only	–
0x14	Interrupt Enable Register	I2SC_IER	Write-only	–
0x18	Interrupt Disable Register	I2SC_IDR	Write-only	–
0x1C	Interrupt Mask Register	I2SC_IMR	Read-only	0x00000000
0x20	Receiver Holding Register	I2SC_RHR	Read-only	0x00000000
0x24	Transmitter Holding Register	I2SC_THR	Write-only	–
0x28–0xFC	Reserved	–	–	–
0x100–0x124	Reserved for PDC registers for left side	–	–	–
0x128–0x1FC	Reserved	–	–	–
0x200–0x224	Reserved for PDC registers for right side	–	–	–

33.8.1 Inter-IC Sound Controller Control Register

Name: I2SC_CR

Address: 0x40000000 (0), 0x40004000 (1)

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
SWRST	–	TXDIS	TXEN	CKDIS	CKEN	RXDIS	RXEN

- **RXEN: Receiver Enable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit enables the I2SC receiver, if RXDIS is not one. Bit I2SC_SR.RXEN is set when the receiver is activated.

- **RXDIS: Receiver Disable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit disables the I2SC receiver. Bit I2SC_SR.RXEN is cleared when the receiver is stopped.

- **CKEN: Clocks Enable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit enables the I2SC clocks generation, if CKDIS is not one.

- **CKDIS: Clocks Disable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a zone to this bit disables the I2SC clock generation.

- **TXEN: Transmitter Enable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit enables the I2SC transmitter, if TXDIS is not one. Bit I2SC_SR.TXEN is set when the Transmitter is started.

- **TXDIS: Transmitter Disable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit disables the I2SC transmitter. Bit I2SC_SR.TXEN is cleared when the Transmitter is stopped.

- **SWRST: Software Reset**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit resets all the registers in the I2SC. The I2SC is disabled after the reset.

33.8.2 Inter-IC Sound Controller Mode Register

Name: I2SC_MR

Address: 0x40000004 (0), 0x40004004 (1)

Access: Read/Write

31	30	29	28	27	26	25	24
IWS	IMCKMODE	IMCKFS					
23	22	21	20	19	18	17	16
–	–	IMCKDIV					
15	14	13	12	11	10	9	8
–	TXSAME	TXDMA	TXMONO	–	RXLOOP	RXDMA	RXMONO
7	6	5	4	3	2	1	0
FORMAT		–	DATALENGTH			–	MODE

The I2SC_MR must be written when the I2SC is stopped. The proper sequence is to write to I2SC_MR, then write to I2SC_CR to enable the I2SC or to disable the I2SC before writing a new value to I2SC_MR.

- **MODE: Inter-IC Sound Controller Mode**

Value	Name	Description
0	SLAVE	I2SCK and i2SWS pin inputs used as bit clock and word select/frame synchronization.
1	MASTER	Bit clock and word select/frame synchronization generated by I2SC from MCK and output to I2SCK and I2SWS pins. MCK is output as master clock on I2SMCK if I2SC_MR.IMCKMODE is set.

- **DATALENGTH: Data Word Length**

Value	Name	Description
0	32_BITS	Data length is set to 32 bits
1	24_BITS	Data length is set to 24 bits
2	20_BITS	Data length is set to 20 bits
3	18_BITS	Data length is set to 18 bits
4	16_BITS	Data length is set to 16 bits
5	16_BITS_COMPACT	Data length is set to 16-bit compact stereo. Left sample in bits 15:0 and right sample in bits 31:16 of same word.
6	8_BITS	Data length is set to 8 bits
7	8_BITS_COMPACT	Data length is set to 8-bit compact stereo. Left sample in bits 7:0 and right sample in bits 15:8 of the same word.

- **FORMAT: Data Format**

Value	Name	Description
0	I2S	I^2S format, stereo with I2SWS low for left channel, and MSB of sample starting one I2SCK period after I2SWS edge
1	LJ	Left-justified format, stereo with I2SWS high for left channel, and MSB of sample starting on I2SWS edge
2	–	Reserved
3	–	Reserved

- **RXDMA: Single or Multiple PDC Channels for Receiver**

0: The receiver uses only one PDC channel for all audio channels.

1: The receiver uses one PDC channel per audio channel.

- **RXMONO: Receive Mono**

0: Stereo

1: Mono, with left audio samples duplicated to right audio channel by the I2SC.

- **RXLOOP: Loop-back Test Mode**

0: Normal mode

1: I2SDO output of I2SC is internally connected to I2SDI input.

- **TXMONO: Transmit Mono**

0: Stereo

1: Mono, with left audio samples duplicated to right audio channel by the I2SC.

- **TXDMA: Single or Multiple PDC Channels for Transmitter**

0: The transmitter uses only one PDC channel for all audio channels.

1: The transmitter uses one PDC channel per audio channel.

- **TXSAME: Transmit Data when Underrun**

0: Zero sample transmitted when underrun.

1: Previous sample transmitted when underrun

- **IMCKDIV: Selected Clock to I2SC Master Clock Ratio**

I2SMCK Master clock output frequency is Selected Clock divided by (IMCKDIV + 1). Refer to the IMCKFS field description.

Notes: 1. This field is write-only. Always read as ‘0’.
2. Do not write a ‘0’ to this field.

- **IMCKFS: Master Clock to f_s Ratio**

Master clock frequency is [2 x 16 × (IMCKFS + 1)] / (IMCKDIV + 1) times the sample rate, i.e., I2SWS frequency.

Value	Name	Description
0	M2SF32	Sample frequency ratio set to 32
1	M2SF64	Sample frequency ratio set to 64
2	M2SF96	Sample frequency ratio set to 96
3	M2SF128	Sample frequency ratio set to 128
5	M2SF192	Sample frequency ratio set to 192
7	M2SF256	Sample frequency ratio set to 256
11	M2SF384	Sample frequency ratio set to 384
15	M2SF512	Sample frequency ratio set to 512
23	M2SF768	Sample frequency ratio set to 768
31	M2SF1024	Sample frequency ratio set to 1024
47	M2SF1536	Sample frequency ratio set to 1536
63	M2SF2048	Sample frequency ratio set to 2048

- **IMCKMODE: Master Clock Mode**

0: No master clock generated (Selected Clock drives I2SCK output).

1: Master clock generated (internally generated clock is used as I2SMCK output).

Warning: If I2SMCK frequency is the same as I2SCK, IMCKMODE must be cleared. Refer to Section 33.6.5 ”Serial Clock and Word Select Generation” and Table 33-4 “Slot Length”.

- **IWS: I2SWS Slot Width**

0: I2SWS slot is 32 bits wide for DATALENGTH = 18/20/24 bits.

1: I2SWS slot is 24 bits wide for DATALENGTH = 18/20/24 bits.

Refer to Table 33-4 “Slot Length”.

33.8.3 Inter-IC Sound Controller Status Register

Name: I2SC_SR

Address: 0x40000008 (0), 0x40004008 (1)

Access: Read-only

31	30	29	28	27	26	25	24
TXBUFE	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	TXURCH		RXBUFF	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	RXORCH	
7	6	5	4	3	2	1	0
ENDTX	TXUR	TXRDY	TXEN	ENDRX	RXOR	RXRDY	RXEN

- **RXEN: Receiver Enabled**

0: This bit is cleared when the receiver is disabled, following a RXDIS or SWRST request in I2SC_CR.

1: This bit is set when the receiver is enabled, following a RXEN request in I2SC_CR.

- **RXRDY: Receive Ready**

0: This bit is cleared when I2SC_RHR is read.

1: This bit is set when received data is present in I2SC_RHR.

- **RXOR: Receive Overrun**

0: This bit is cleared when the corresponding bit in I2SC_SCR is written to ’1’.

1: This bit is set when an overrun error occurs on I2SC_RHR or when the corresponding bit in I2SC_SSR is written to ’1’.

- **ENDRX: End of Receiver Transfer**

0: This bit is set when PDC has completed the receive transfer.

1: This bit is cleared when a new receive transfer is programmed into the PDC.

- **TXEN: Transmitter Enabled**

0: This bit is cleared when the transmitter is disabled, following a I2SC_CR.TXDIS or I2SC_CR.SWRST request.

1: This bit is set when the transmitter is enabled, following a I2SC_CR.TXEN request.

- **TXRDY: Transmit Ready**

0: This bit is cleared when data is written to I2SC_THR.

1: This bit is set when I2SC_THR is empty and can be written with new data to be transmitted.

- **TXUR: Transmit Underrun**

0: This bit is cleared when the corresponding bit in I2SC_SCR is written to ’1’.

1: This bit is set when an underrun error occurs on I2SC_THR or when the corresponding bit in I2SC_SSR is written to ’1’.

- **ENDTX: End of Transmitter Transfer**

0: This bit is set when the PDC has completed the transmit transfer.

1: This bit is cleared when a new transmit transfer is programmed into the PDC.

- **RXORCH: Receive Overrun Channel**

This field is cleared when I2SC_SCR.RXOR is written to ’1’.

Bit i of this field is set when a receive overrun error occurred in channel i (i = 0 for first channel of the frame).

- **RXBUFF: Receive Buffer Full**

0: This bit is set when received data is present in I2SC_RHR.

1: This bit is cleared when I2SC_RHR is read and no more received data is present.

- **TXURCH: Transmit Underrun Channel**

0: This field is cleared when I2SC_SCR.TXUR is written to ’1’.

1: Bit i of this field is set when a transmit underrun error occurred in channel i (i = 0 for first channel of the frame).

- **TXBUFE: Transmit Buffer Empty**

0: This bit is set when I2SC_THR is empty and can be written with new data to be transmitted.

1: This bit is cleared when data is written to I2SC_THR and is being transmitted.

33.8.4 Inter-IC Sound Controller Status Clear Register

Name: I2SC_SCR

Address: 0x4000000C (0), 0x4000400C (1)

Access: Write-only

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

- **RXOR: Receive Overrun Status Clear**

Writing a ’0’ to this bit has no effect.

Writing a ’1’ to this bit clears the status bit.

- **TXUR: Transmit Underrun Status Clear**

Writing a ’0’ to this bit has no effect.

Writing a ’1’ to this bit clears the status bit.

- **RXORCH: Receive Overrun Per Channel Status Clear**

Writing a ’0’ has no effect.

Writing a ’1’ to any bit in this field clears the corresponding bit in the I2SC_SR and the corresponding interrupt request.

- **TXURCH: Transmit Underrun Per Channel Status Clear**

Writing a ’0’ has no effect.

Writing a ’1’ to any bit in this field clears the corresponding bit in the I2SC_SR and the corresponding interrupt request.

33.8.5 Inter-IC Sound Controller Status Set Register

Name: I2SC_SSR

Address: 0x40000010 (0), 0x40004010 (1)

Access: Write-only

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

- **RXOR: Receive Overrun Status Set**

Writing a ’0’ to this bit has no effect.

Writing a ’1’ to this bit sets the status bit.

- **TXUR: Transmit Underrun Status Set**

Writing a ’0’ to this bit has no effect.

Writing a ’1’ to this bit sets the status bit.

- **RXORCH: Receive Overrun Per Channel Status Set**

Writing a ’0’ has no effect.

Writing a ’1’ to any bit in this field sets the corresponding bit in I2SC_SR and the corresponding interrupt request.

- **TXURCH: Transmit Underrun Per Channel Status Set**

Writing a ’0’ has no effect.

Writing a ’1’ to any bit in this field sets the corresponding bit in I2SC_SR and the corresponding interrupt request.

33.8.6 Inter-IC Sound Controller Interrupt Enable Register

Name: I2SC_IER

Address: 0x40000014 (0), 0x40004014 (1)

Access: Write-only

31	30	29	28	27	26	25	24
TXEMPTY	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	RXFULL	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
ENDTX	TXUR	TXRDY	–	ENDRX	RXOR	RXRDY	–

- **RXRDY: Receiver Ready Interrupt Enable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit sets the corresponding bit in I2SC_IMR.

- **RXOR: Receiver Overrun Interrupt Enable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit sets the corresponding bit in I2SC_IMR.

- **ENDRX: End of Reception Interrupt Enable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit sets the corresponding bit in the I2SC_IMR.

- **TXRDY: Transmit Ready Interrupt Enable**

0: Writing a ’0’ to this bit as no effect.

1: Writing a ’1’ to this bit sets the corresponding bit in I2SC_IMR.

- **TXUR: Transmit Underflow Interrupt Enable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit sets the corresponding bit in I2SC_IMR.

- **ENDTX: End of Transmission Interrupt Enable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit sets the corresponding bit in I2SC_IMR.

- **RXFULL: Receive Buffer Full Interrupt Enable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit sets the corresponding bit in I2SC_IMR.

- **TXEMPTY: Transmit Buffer Empty Interrupt Enable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit sets the corresponding bit in I2SC_IMR.

33.8.7 Inter-IC Sound Controller Interrupt Disable Register

Name: I2SC_IDR

Address: 0x40000018 (0), 0x40004018 (1)

Access: Write-only

31	30	29	28	27	26	25	24
TXEMPTY	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	RXFULL	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
ENDTX	TXUR	TXRDY	–	ENDRX	RXOR	RXRDY	–

- **RXRDY: Receiver Ready Interrupt Disable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit clears the corresponding bit in I2SC_IMR.

- **RXOR: Receiver Overrun Interrupt Disable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit clears the corresponding bit in I2SC_IMR.

- **ENDRX: End of Reception Interrupt Disable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit clears the corresponding bit in I2SC_IMR.

- **TXRDY: Transmit Ready Interrupt Disable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit clears the corresponding bit in I2SC_IMR.

- **TXUR: Transmit Underflow Interrupt Disable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit clears the corresponding bit in I2SC_IMR.

- **ENDTX: End of Transmission Interrupt Disable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit clears the corresponding bit in I2SC_IMR.

- **RXFULL: Receive Buffer Full Interrupt Disable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit clears the corresponding bit in I2SC_IMR.

- **TXEMPTY: Transmit Buffer Empty Interrupt Disable**

0: Writing a ’0’ to this bit has no effect.

1: Writing a ’1’ to this bit clears the corresponding bit in I2SC_IMR.

33.8.8 Inter-IC Sound Controller Interrupt Mask Register

Name: I2SC_IMR

Address: 0x4000001C (0), 0x4000401C (1)

Access: Write-only

31	30	29	28	27	26	25	24
TXEMPTY	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	RXFULL	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
ENDTX	TXUR	TXRDY	–	ENDRX	RXOR	RXRDY	–

- **RXRDY: Receiver Ready Interrupt Disable**

0: The corresponding interrupt is disabled. This bit is cleared when the corresponding bit in I2SC_IDR is written to ’1’.

1: The corresponding interrupt is enabled. This bit is set when the corresponding bit in I2SC_IER is written to ’1’.

- **RXOR: Receiver Overrun Interrupt Disable**

0: The corresponding interrupt is disabled. This bit is cleared when the corresponding bit in I2SC_IDR is written to ’1’.

1: The corresponding interrupt is enabled. This bit is set when the corresponding bit in I2SC_IER is written to ’1’.

- **ENDRX: End of Reception Interrupt Disable**

0: The corresponding interrupt is disabled. This bit is cleared when the corresponding bit in I2SC_IDR is written to ’1’.

1: The corresponding interrupt is enabled. This bit is set when the corresponding bit in I2SC_IER is written to ’1’.

- **TXRDY: Transmit Ready Interrupt Disable**

0: The corresponding interrupt is disabled. This bit is cleared when the corresponding bit in I2SC_IDR is written to ’1’.

1: The corresponding interrupt is enabled. This bit is set when the corresponding bit in I2SC_IER is written to ’1’.

- **TXUR: Transmit Underflow Interrupt Disable**

0: The corresponding interrupt is disabled. This bit is cleared when the corresponding bit in I2SC_IDR is written to ’1’.

1: The corresponding interrupt is enabled. This bit is set when the corresponding bit in I2SC_IER is written to ’1’.

- **ENDTX: End of Transmission Interrupt Disable**

0: The corresponding interrupt is disabled. This bit is cleared when the corresponding bit in I2SC_IDR is written to ’1’.

1: The corresponding interrupt is enabled. This bit is set when the corresponding bit in I2SC_IER is written to ’1’.

- **RXFULL: Receive Buffer Full Interrupt Disable**

0: The corresponding interrupt is disabled. This bit is cleared when the corresponding bit in I2SC_IDR is written to ’1’.

1: The corresponding interrupt is enabled. This bit is set when the corresponding bit in I2SC_IER is written to ’1’.

- **TXEMPTY: Transmit Buffer Empty Interrupt Disable**

0: The corresponding interrupt is disabled. This bit is cleared when the corresponding bit in I2SC_IDR is written to ’1’.

1: The corresponding interrupt is enabled. This bit is set when the corresponding bit in I2SC_IER is written to ’1’.

33.8.9 Inter-IC Sound Controller Receiver Holding Register

Name: I2SC_RHR

Address: 0x40000020 (0), 0x40004020 (1)

Access: Read-only

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

- **RHR: Receiver Holding Register**

This field is set by hardware to the last received data word. If I2SC_MR.DATALENGTH specifies fewer than 32 bits, data is right justified in the RHR field.

33.8.10 Inter-IC Sound Controller Transmitter Holding Register

Name: I2SC_THR

Address: 0x40000024 (0), 0x40004024 (1)

Access: Write-only

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

- **THR: Transmitter Holding Register**

Next data word to be transmitted after the current word if TXRDY is not set. If I2SC_MR.DATALENGTH specifies fewer than 32 bits, data is right-justified in the THR field.

34. Pulse Density Modulation Interface Controller (PDMIC)

34.1 Description

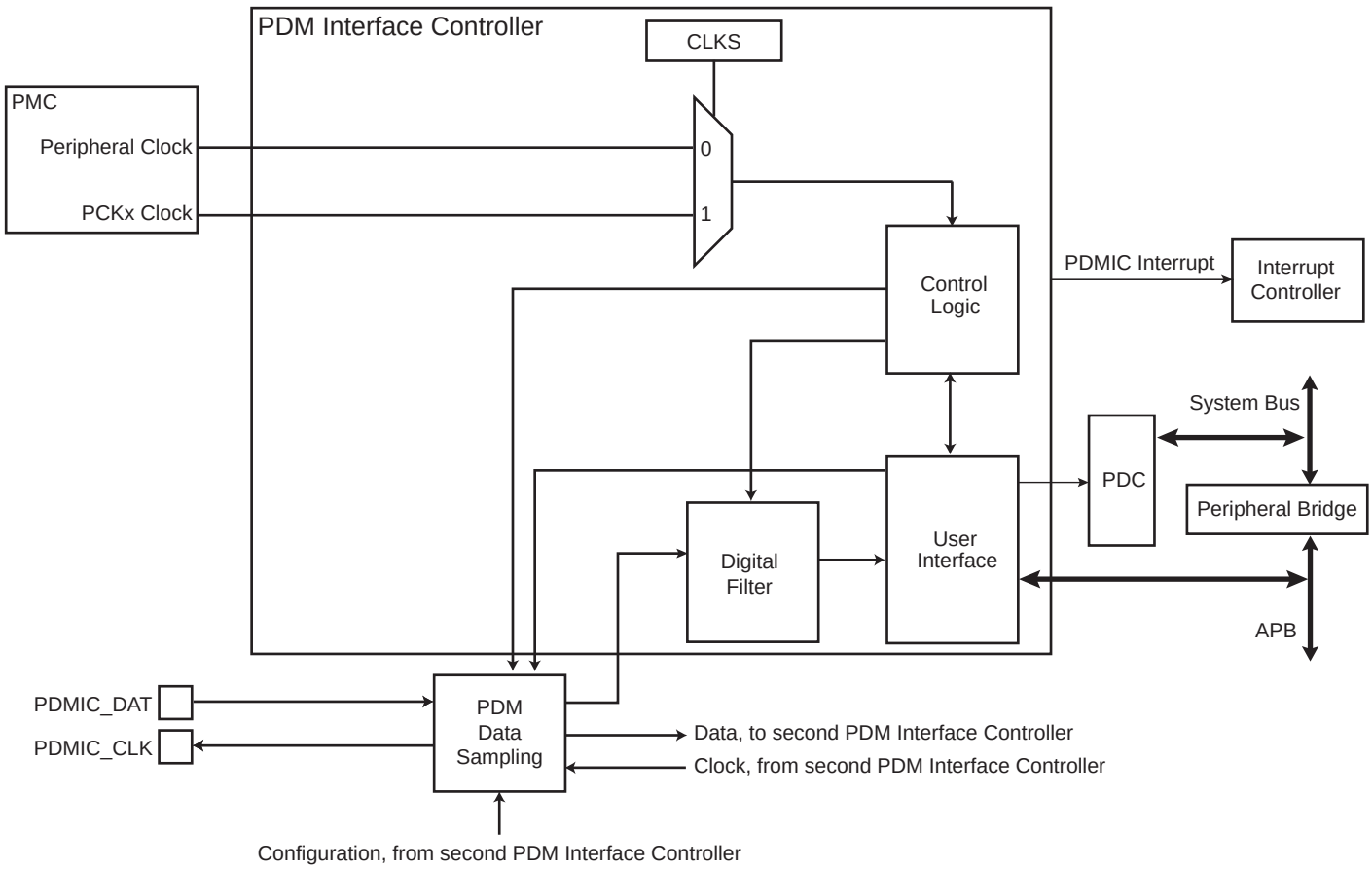
The Pulse Density Modulation Interface Controller (PDMIC) is a PDM interface controller and decoder that support both mono and stereo PDM formats. It integrates a clock generator driving the PDM microphones and embeds filters which decimate the incoming bitstream to obtain most common audio rates.

34.2 Embedded Characteristics

- Multiplexed PDM Input Support
- 16-bit Resolution
- PDC Support
- Up to 4 Conversions Stored
- PDM Clock Source can be Independent from Core Clock
- Register Write Protection

34.3 Block Diagram

Figure 34-1. PDMIC Block Diagram



34.4 Signal Description

Table 34-1. PDMIC Pin Description

Pin Name	Description	Type
PDMIC_CLK	Pulse Density Modulation Bitstream Sampling Clock	Output
PDMIC_DAT	Pulse Density Modulation Multiplexed Data	Input

34.5 Product Dependencies

34.5.1 I/O Lines

The pins used for interfacing the PDMIC are multiplexed with PIO lines. The programmer must first program the PIO controllers to assign the peripheral functions to PDMIC pins.

Table 34-2. I/O Lines

Instance	Signal	I/O Line	Peripheral
PDMIC0	PDMIC0_CLK	PA10	B
PDMIC0	PDMIC0_DAT	PA9	B

34.5.2 Power Management

The PDMIC is not continuously clocked. The user must first enable the PDMIC peripheral clock in the Power Management Controller (PMC) before using the controller.

34.5.3 Interrupt Sources

The PDMIC interrupt line is connected on one of the internal sources of the Interrupt Controller. Using the PDMIC interrupt requires the Interrupt Controller to be programmed first.

Table 34-3. Peripheral IDs

Instance	ID
PDMIC0	13
PDMIC1	18

34.6 Functional Description

34.6.1 PDM Interface

34.6.1.1 Description

The PDM clock (PDMIC_CLK) is used to sample the PDM bitstream.

The PDMIC_CLK frequency range is between peripheral clock/2 and peripheral clock/256 or between PCKx clock/2 and PCKx clock/256, depending on the selected clock source.

The PCKx clock frequency must always be at least three times lower than the peripheral clock frequency.

The field PRESCAL in the Mode Register (PDMIC_MR) must be programmed in order to provide a PDMIC_CLK frequency compliant with the microphone parameters.

34.6.1.2 Start-up Sequence

To start processing the bitstream coming from the PDM interface, follow the steps below:

1. Clear all bits in the Control Register (PDMIC_CR) or compute a soft reset using the SWRST bit of PDMIC_CR.
2. Configure the PRESCAL field in PDMIC_MR according to the microphone specifications.
3. Enable the PDM mode and start the conversions using the ENPDM bit in PDMIC_CR.

34.6.1.3 Restrictions

The external PDM data sampling module is shared with two PDMIC modules. When two PDM microphones are connected on the PDMIC_DAT line, both PDMICs must be enabled and the steps below must be followed:

1. Clear all bits in PDMIC_CR or compute a soft reset using the SWRST bit of PDMIC_CR in both PDMIC modules.
2. Configure the PMC to provide the same clock to both PDMIC modules (clock frequencies should be the same).
3. Configure the PRESCAL field in PDMIC_MR according to the microphone specifications (the PRESCAL value should be the same in both PDMIC).
4. Enable the PDM mode and start the conversions using the ENPDM bit in PDMIC_CR in the PDMIC0.
5. Enable the PDM mode and start the conversions using the ENPDM bit in PDMIC_CR in the PDMIC1.

To stop the conversions, follow the steps below:

1. Disable the PDM mode or compute a software reset in PDMIC1.
2. Disable the PDM mode or compute a software reset in PDMIC0.

The bitstream sampled on the rising edge of PDMIC_CLK is routed to PDMIC0 and the bitstream sampled on the falling edge of PDMIC_CLK is routed to PDMIC1.

34.6.2 Digital Signal Processing (Digital Filter)

34.6.2.1 Description

The PDMIC includes a DSP section containing a decimation filter, a droop compensation filter, a sixth-order low pass filter, a first-order high pass filter and an offset and gain compensation stage. A block diagram of the DSP section is represented in Figure 34-2 “DSP Block Diagram”.

Data processed by the filtering section are two’s complement signals defined on 24 bits.

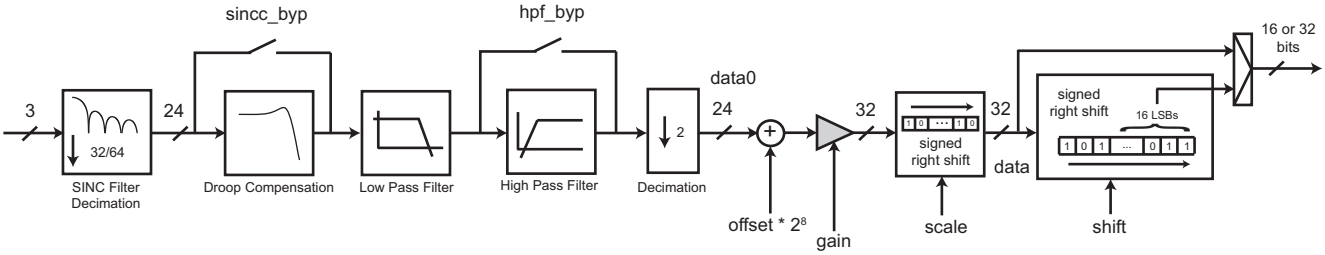
The filtering of the decimation stage is performed by a fourth-order sinc-based filter whose zeros are placed in order to minimize aliasing effects of the decimation. The decimation ratio of this filter is either 32 or 64. The droop induced by this filter can be compensated by the droop compensation stage.

The sixth-order low pass filter is used to decimate the sinc filter output by a ratio of 2.

An optional first-order high pass filter is implemented in order to eliminate the DC component of the incoming signal.

The overall decimation ratio of this DSP section is either 64 or 128. This fits an audio sampling rate of 48 kHz with a PDM microphone sampling frequency of either 3.072 or 6.144 MHz. The frequency response of the filters optimizes the gain flatness between 0 and 20 kHz (when the droop compensation filter is implemented and the high pass filter is bypassed) and highly reduces the aliasing effects of the decimation.

Figure 34-2. DSP Block Diagram



34.6.2.2 Decimation Filter

The sigma-delta architecture of the PDM microphone implies a filtering and a decimation of the bitstream at the output of the microphone bitstream. The decimation filter decimates the bitstream by either 32 or 64. To perform this operation, a fourth-order sinc filter with an Over-Sampling Ratio (OSR) of 32 or 64 is implemented with the following transfer function:

$$H(z) = \frac{1}{OSR^4}\left(\sum_{i=0}^{OSR-1} z^{-i}\right)^4$$

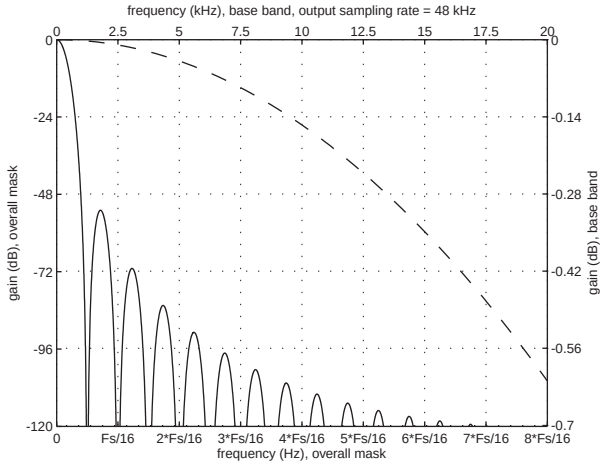
The DC gain of this filter is unity and does not depend on its OSR. However, as it generates a fourth-order zero at *Fs/OSR* frequency multiples (Fs being the sampling frequency of the microphone), the frequency response of the decimation filter depends on the OSR parameter. See Section 34.6.2.3 “Droop Compensation” for frequency plots.

Its non-flat frequency response can be compensated over the 0 to 20 kHz band by using the droop compensation filter when the decimated frequency is set to 48 kHz. See Section 34.6.2.3 “Droop Compensation”.

If the decimated sampling rate is modified, the frequency response of this filter is scaled proportionally to the new frequency.

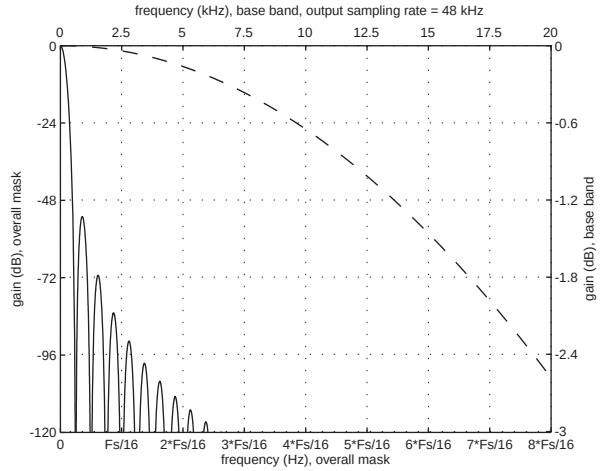
In Figure 34-3 and Figure 34-4, Fs is the sampling rate of the PDM microphone.

Figure 34-3. Spectral mask of an OSR = 32, Fs = 6.144 MHz, Fourth-Order Sinc Filter: Overall Response (continuous line) and 0 to 20 kHz Bandwidth Response (dashed line)



The zeros of this filter are located at multiples of Fs/32

Figure 34-4. Spectral Mask of an OSR = 64, Fs = 3.072 MHz, Fourth-order Sinc Filter: Overall Response (continuous line) and 0 to 20 kHz Bandwidth Response (dashed line)



The zeros of this filter are located at multiples of Fs/64.

34.6.2.3 Droop Compensation

The droop effect introduced by the sinc filter can be compensated in the 0 to 20 kHz by the droop compensation filter (see Figure 34-5). This is a second-order IIR filter which is applied on the signal output by the sinc. The default coefficients of the droop compensation filter are computed to optimize the droop of the sinc filter with the decimated frequency equal to 48 kHz.

This filter compensates the droop of the sinc filter regardless of the OSR value.

If the decimated sampling rate is modified, the frequency response of this filter is scaled proportionally to the new frequency.

This filter can be bypassed by setting the SINBYP bit in the PDMIC DSP Configuration Register 0 (PDMIC_DSPR0).

Figure 34-5. Droop Compensation Filter Overall Frequency Response

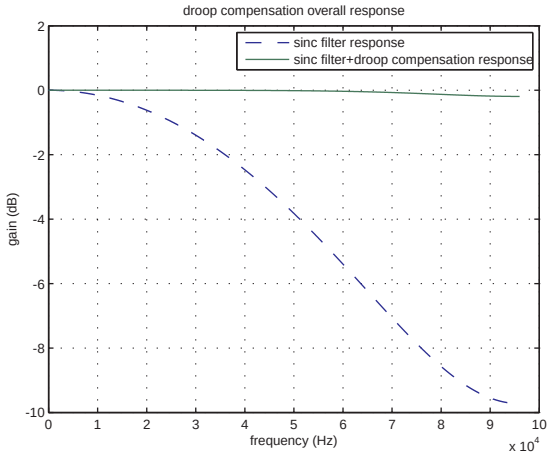
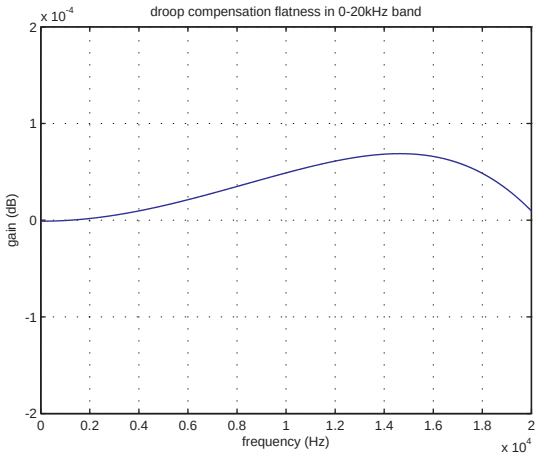


Figure 34-6. Droop Compensation Filter 0 to 20 kHz Band Flatness



34.6.2.4 Low Pass Filter

The PDMIC includes a sixth-order IIR filter that performs a low pass transfer function and decimates by 2 the output of the sinc filter. The coefficients are computed for a decimated sampling rate of 48 kHz and optimize the 0 to 20 kHz band flatness while rejecting the aliasing of the PDM microphone by at least 60 dB in the 28 to 48 kHz band.

If the decimated sampling rate is modified, the frequency response of this filter is scaled proportionally to the new frequency.

Figure 34-7 and Figure 34-8 are drawn for an output sampling frequency of 48 kHz.

Figure 34-7. Low Pass Filter Spectral Mask

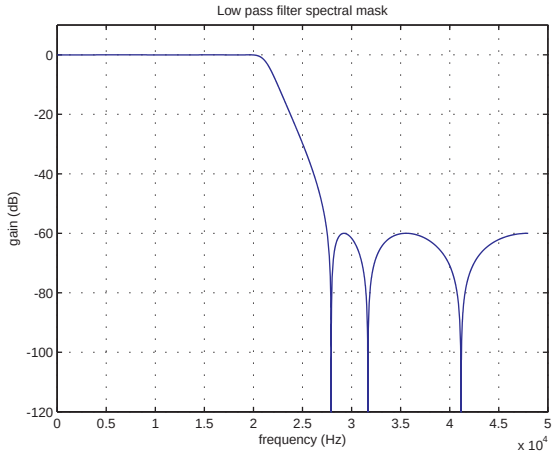
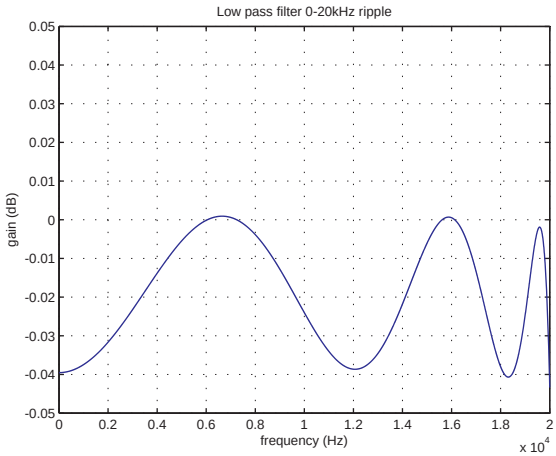


Figure 34-8. Low Pass Filter Ripple in the 0 to 20 kHz Band



34.6.2.5 High Pass Filter

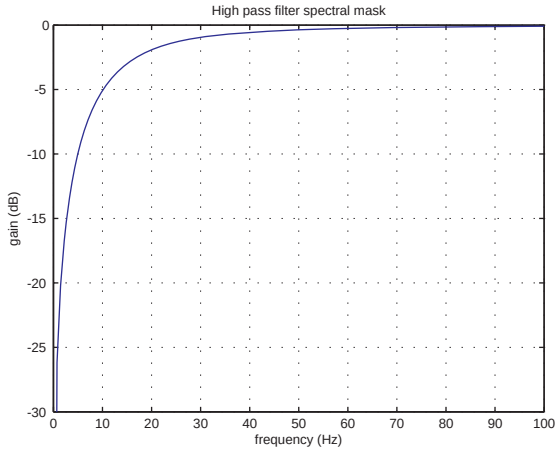
The PDMIC includes an optional first-order IIR filter performing a high pass transfer function after the low pass filter and before the decimation. The coefficients are computed for a decimated sampling rate of 48 kHz to obtain a -3dB cutoff frequency at 15 Hz.

If the decimated sampling rate is modified, the frequency response of this filter is scaled proportionally to the new frequency.

This filter can be bypassed by setting the HPFBYP bit in PDMIC_DSPR0 (see Section 34.7.8 “PDMIC DSP Configuration Register 0”).

Figure 34-9 is drawn for an output sampling frequency of 48 kHz.

Figure 34-9. High Pass Filter Spectral Mask in the 0 to 100 Hz Band



34.6.2.6 Gain and Offset Compensation

An offset, a gain, a scaling factor and a shift can be applied to a converted PDM microphone value using the following operation:

$$data = \frac{(data_0 + offset \times 2^8) \times dgain}{2^{scale + shift + 8}}$$

where:

- $data_0$ is a signed integer defined on 24 bits. It is the output of the filtering channel.
- *offset* is a signed integer defined on 16 bits (see PDMIC DSP Configuration Register 1). It is multiplied by 2^8 to have the same weight as $data_0$.
- *dgain* is an unsigned integer defined on 15 bits (see PDMIC DSP Configuration Register 1). Only the 32 MSBs of the multiplication operation are used for scaling and shifting operations. dgain defaults to 0 after reset, which forces CDR to 0. It must be programmed to a non-zero value to read non-zero data into the PDMIC_CDR register.
- *scale* is an unsigned integer defined on 4 bits (see PDMIC DSP Configuration Register 0). It shifts the multiplication operation result by *scale* bits to the right. Maximum allowed value is 15.
- *shift* is an unsigned integer defined on 4 bits (see PDMIC DSP Configuration Register 0). It shifts the multiplication operation result by *shift* bits to the right. Maximum allowed value is 15.

If the data transfer is configured in 32-bit mode (see PDMIC DSP Configuration Register 0), the 2^{shift} division is not performed and the 32-bit result of the remaining operation is sent.

If the data transfer is configured in 16-bit mode, the 2^{shift} division is performed. The result is then saturated to be within $\pm(2^{15}-1)$ and the 16 LSBs of this saturation operation are sent to the controller as the result of the PDM microphone conversion.

Default parameters are defined to output a 16-bit result whatever the data transfer configuration may be.

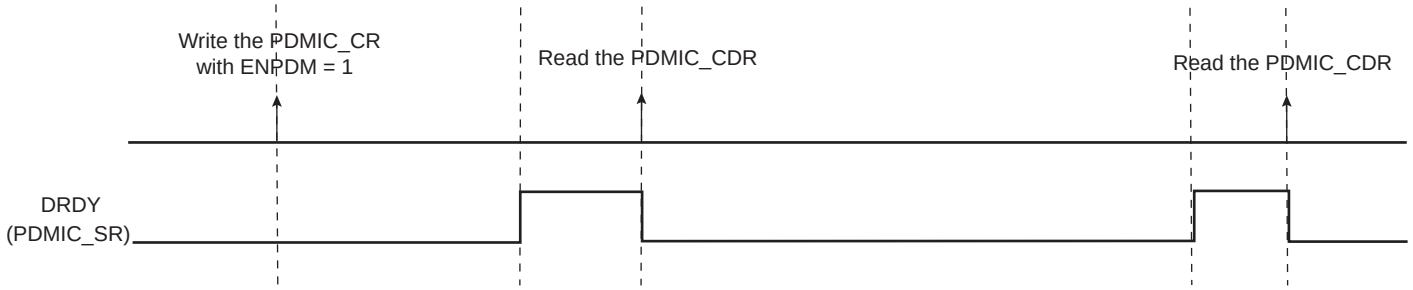
34.6.3 Conversion Results

When a conversion is completed, the resulting 16-bit digital value is stored in the PDMIC Converted Data Register (PDMIC_CDR).

The DRDY bit in the Interrupt Status Register (PDMIC_ISR) is set. In the case of a connected PDC channel, DRDY rising triggers a data transfer request. In any case, DRDY can trigger an interrupt.

Reading PDMIC_CDR clears the DRDY flag.

Figure 34-10. DRDY Flag Behavior



If PDMIC_CDR is not read before further incoming data is converted, the Overrun Error (OVRE) flag is set in PDMIC_ISR. Likewise, new data converted when DRDY is high sets the OVRE bit (Overrun Error) in PDMIC_ISR. In case of overrun, the newly converted data is lost.

The OVRE flag is automatically cleared when PDMIC_ISR is read.

34.6.4 Register Write Protection

To prevent any single software error from corrupting PDMIC behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the PDMIC Write Protection Mode Register (PDMIC_WPMR).

If a write access to a write-protected register is detected, the WPVS flag in the PDMIC Write Protection Status Register (PDMIC_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 PDMIC_WPSR.

The following registers can be write-protected:

- PDMIC Mode Register
- PDMIC DSP Configuration Register 0
- PDMIC DSP Configuration Register 1

34.7 Pulse Density Modulation Interface Controller (PDMIC) User Interface

Table 34-4. Register Mapping

Offset[1]	Register	Name	Access	Reset
0x00	Control Register	PDMIC_CR	Read/Write	0x00000000
0x04	Mode Register	PDMIC_MR	Read/Write	0x00F00000
0x08–0x10	Reserved	–	–	–
0x14	Converted Data Register	PDMIC_CDR	Read-only	0x00000000
0x18	Interrupt Enable Register	PDMIC_IER	Write-only	–
0x1C	Interrupt Disable Register	PDMIC_IDR	Write-only	–
0x20	Interrupt Mask Register	PDMIC_IMR	Read-only	0x00000000
0x24	Interrupt Status Register	PDMIC_ISR	Read-only	0x00000000
0x28–0x54	Reserved	–	–	–
0x58	DSP Configuration Register 0	PDMIC_DSPR0	Read/Write	0x00000000
0x5C	DSP Configuration Register 1	PDMIC_DSPR1	Read/Write	0x00000001
0x60–0xE0	Reserved	–	–	–
0xE4	Write Protection Mode Register	PDMIC_WPMR	Read/Write	0x00000000
0xE8	Write Protection Status Register	PDMIC_WPSR	Read-only	0x00000000
0xEC–0xFC	Reserved	–	–	–
0x100–0x124	Reserved for PDC Registers	–	–	–

Notes: 1. If an offset is not listed in the table, it must be considered as “reserved”.

34.7.1 PDMIC Control Register

Name: PDMIC_CR

Address: 0x4002C000 (0), 0x40030000 (1)

Access: Read/Write

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

- **SWRST: Software Reset**

0: No effect.

1: Resets the PDMIC, simulating a hardware reset.

Warning: The read value of this bit is always 0.

- **ENPDM: Enable PDM**

0: Disables the PDM and stops the conversions.

1: Enables the PDM and starts the conversions.

34.7.2 PDMIC Mode Register

Name: PDMIC_MR

Address: 0x4002C004 (0), 0x40030004 (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
–	PRESCAL						
7	6	5	4	3	2	1	0
–	–	–	CLKS	–	–	–	–

This register can only be written if the WPEN bit is cleared in the PDMIC Write Protection Mode Register.

- **CLKS: Clock Source Selection**

0: Peripheral clock selected

1: PCKx clock selected (This clock source can be independent of the processor clock.)

- **PRESCAL: Prescaler Rate Selection**

PRESCAL determines the frequency of the PDM bitstream sampling clock (PDMIC_CLK):

$$PRESCAL = \frac{SELCK}{2 \times f_{PDMIC\_CLK}} - 1$$

where SELCK is either $f_{peripheral\ clock}$ or $f_{PCKx\ clock}$ depending on the value of bit CLKS ($f_{peripheral\ clock}$ or $f_{PCKx\ clock}$ is the clock frequency in Hz).

34.7.3 PDMIC Converted Data Register

Name: PDMIC_CDR

Address: 0x4002C014 (0), 0x40030014 (1)

Access: Read-only

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

- **DATA: Data Converted**

The filtered output data is placed into this register at the end of a conversion and remains until it is read.

34.7.4 PDMIC Interrupt Enable Register

Name: PDMIC_IER

Address: 0x4002C018 (0), 0x40030018 (1)

Access: Write-only

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

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **DRDY: Data Ready Interrupt Enable**
- **OVRE: Overrun Error Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable**
- **RXBUFF: Receive Buffer Full Interrupt Enable**

34.7.5 PDMIC Interrupt Disable Register

Name: PDMIC_IDR

Address: 0x4002C01C (0), 0x4003001C (1)

Access: Write-only

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

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Disables the corresponding interrupt.

- **DRDY: Data Ready Interrupt Disable**
- **OVRE: General Overrun Error Interrupt Disable**
- **ENDRX: End of Receive Buffer Interrupt Disable**
- **RXBUFF: Receive Buffer Full Interrupt Disable**

34.7.6 PDMIC Interrupt Mask Register

Name: PDMIC_IMR

Address: 0x4002C020 (0), 0x40030020 (1)

Access: Read-only

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

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.

- **DRDY: Data Ready Interrupt Mask**
- **OVRE: General Overrun Error Interrupt Mask**
- **ENDRX: End of Receive Buffer Interrupt Mask**
- **RXBUFF: Receive Buffer Full Interrupt Mask**

34.7.7 PDMIC Interrupt Status Register

Name: PDMIC_ISR

Address: 0x4002C024 (0), 0x40030024 (1)

Access: Read-only

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

- **FIFOCNT: FIFO Count**

Number of conversions available in the FIFO (not a source of interrupt).

- **DRDY: Data Ready (cleared by reading PDMIC_CDR)**

0: No data has been converted since the last read of PDMIC_CDR.

1: At least one data has been converted and is available in PDMIC_CDR.

- **OVRE: Overrun Error (cleared on read)**

0: No overrun error has occurred since the last read of PDMIC_ISR.

1: At least one overrun error has occurred since the last read of PDMIC_ISR.

- **ENDRX: End of RX Buffer (cleared by writing PDMIC_RCR or PDMIC_RNCR)**

0: The Receive Counter register has not reached 0 since the last write in PDMIC_RCR or PDMIC_RNCR.

1: The Receive Counter register has reached 0 since the last write in PDMIC_RCR or PDMIC_RNCR.

Note: PDMIC_RCR and PDMIC_RNCR are located in the Peripheral DMA Controller (PDC).

- **RXBUFF: RX Buffer Full (cleared by writing PDMIC_RCR or PDMIC_RNCR)**

0: PDMIC_RCR or PDMIC_RNCR has a value other than 0.

1: Both PDMIC_RCR and PDMIC_RNCR have a value of 0.

Note: PDMIC_RCR and PDMIC_RNCR are located in the Peripheral DMA Controller (PDC).

34.7.8 PDMIC DSP Configuration Register 0

Name: PDMIC_DSPR0

Address: 0x4002C058 (0), 0x40030058 (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
SHIFT				SCALE			
7	6	5	4	3	2	1	0
–	OSR			SIZE	SINBYP	HPFBYP	–

This register can only be written if the WPEN bit is cleared in the PDMIC Write Protection Mode Register.

- **HPFBYP: High-Pass Filter Bypass**

0: High-pass filter enabled.

1: Bypasses the high-pass filter.

- **SINBYP: SINCC Filter Bypass**

0: Droop compensation filter enabled.

1: Bypasses the droop compensation filter.

- **SIZE: Data Size**

0: Converted data size is 16 bits.

1: Converted data size is 32 bits.

- **OSR: Global Oversampling Ratio**

Value	Name	Description
0	128	Global Oversampling ratio is 128 (SINC filter oversampling ratio is 64)
1	64	Global Oversampling ratio is 64 (SINC filter oversampling ratio is 32)

Note: Values not listed are reserved.

- **SCALE: Data Scale**

Shifts the multiplication operation result by SCALE bits to the right.

- **SHIFT: Data Shift**

Shifts the scaled result by SHIFT bits to the right.

34.7.9 PDMIC DSP Configuration Register 1

Name: PDMIC_DSPR1

Address: 0x4002C05C (0), 0x4003005C (1)

Access: Read/Write

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

This register can only be written if the WPEN bit is cleared in the PDMIC Write Protection Mode Register.

- **DGAIN: Gain Correction**

Gain correction to apply to the final result.

- **OFFSET: Offset Correction**

Offset correction to apply to the final result.

DGAIN and OFFSET values can be determined using the formula in Section 34.6.2.6 “Gain and Offset Compensation”.

34.7.10 PDMIC Write Protection Mode Register

Name: PDMIC_WPMR

Address: 0x4002C0E4 (0), 0x400300E4 (1)

Access: Read/Write

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

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x414443 (“ADC” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x414443 (“ADC” in ASCII).

See Section 34.6.4 “Register Write Protection” for the list of registers that can be write-protected.

- **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.

34.7.11 PDMIC Write Protection Status Register

Name: PDMIC_WPSR

Address: 0x4002C0E8 (0), 0x400300E8 (1)

Access: Read-only

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

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of PDMIC_WPSR.

1: A write protection violation has occurred since the last read of PDMIC_WPSR. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protection Violation Source**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

35. Cyclic Redundancy Check Calculation Unit (CRCCU)

35.1 Description

The Cyclic Redundancy Check Calculation Unit (CRCCU) has its own DMA which functions as a Master with the Bus Matrix. Three different polynomials are available: CCITT802.3, CASTAGNOLI and CCITT16.

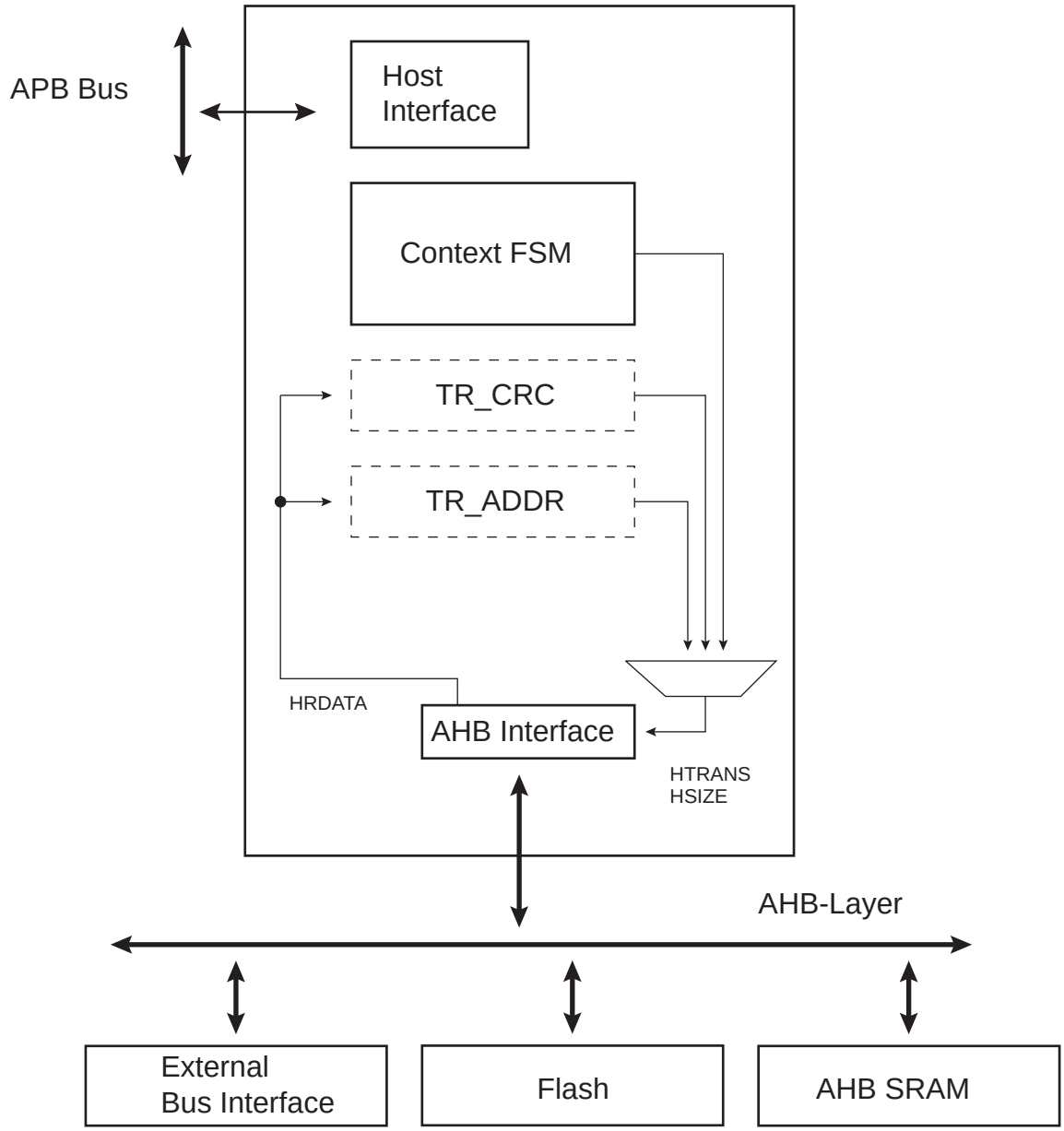
The CRCCU is designed to perform data integrity checks of off-/on-chip memories as a background task without CPU intervention.

35.2 Embedded Characteristics

- Data Integrity Check of Off-/On-Chip Memories
- Background Task Without CPU Intervention
- Performs Cyclic Redundancy Check (CRC) Operation on Programmable Memory Area
- Programmable Bus Burden

35.3 CRCCU Block Diagram

Figure 35-1. Block Diagram



35.4 Product Dependencies

35.4.1 Power Management

The CRCCU is clocked through the Power Management Controller (PMC), the programmer must first configure the CRCCU in the PMC to enable the CRCCU clock.

35.4.2 Interrupt Source

The CRCCU has an interrupt line connected to the Interrupt Controller. Handling the CRCCU interrupt requires programming the Interrupt Controller before configuring the CRCCU.

35.5 CRCCU Functional Description

35.5.1 CRC Calculation Unit

The CRCCU integrates a dedicated Cyclic Redundancy Check (CRC) engine. When configured and activated, this CRC engine performs a checksum computation on a memory area. CRC computation is performed from the LSB to MSB. Three different polynomials are available: CCITT802.3, CASTAGNOLI and CCITT16 (see field description "PTYPE: Primitive Polynomial" in Section 35.7.10 "CRCCU Mode Register" for details).

35.5.2 CRC Calculation Unit Operation

The CRCCU has a DMA controller that supports programmable CRC memory checks. When enabled, the DMA channel reads a programmable amount of data and computes CRC on the fly.

The CRCCU is controlled by two registers, TR_ADDR and TR_CTRL, which need to be mapped in the internal SRAM. The addresses of these two registers are pointed to by the CRCCU_DSCR.

Table 35-1. CRCCU Descriptor Memory Mapping

		SRAM Memory
CRCCU_DSCR+0x0	---->	TR_ADDR
CRCCU_DSCR+0x4	---->	TR_CTRL
CRCCU_DSCR+0x8	---->	Reserved
CRCCU_DSCR+0xC	---->	Reserved
CRCCU_DSCR+0x10	---->	TR_CRC

TR_ADDR defines the start address of memory area targeted for CRC calculation.

TR_CTRL defines the buffer transfer size, the transfer width (byte, halfword, word) and the transfer-completed interrupt enable.

To start the CRCCU, set the CRC enable bit (ENABLE) and configure the mode of operation in the CRCCU Mode Register (CRCCU_MR), then configure the Transfer Control Registers and finally, set the DMA enable bit (DMAEN) in the CRCCU DMA Enable Register (CRCCU_DMA_EN).

When the CRCCU is enabled, the CRCCU reads the predefined amount of data (defined in TR_CTRL) located from TR_ADDR start address and computes the checksum.

The CRCCU_SR contains the temporary CRC value.

The BTSIZE field located in the TR_CTRL register (located in memory), is automatically decremented if its value is different from zero. Once the value of the BTSIZE field is equal to zero, the CRCCU is disabled by hardware. In this case, the relevant CRCCU DMA Status Register bit DMASR is automatically cleared.

If the COMPARE field of the CRCCU_MR is set to true, the TR_CRC (Transfer Reference Register) is compared with the last CRC computed. If a mismatch occurs, an error flag is set and an interrupt is raised (if unmasked).

The CRCCU accesses the memory by single access (TRWIDTH size) in order not to limit the bandwidth usage of the system, but the DIVIDER field of the CRCCU Mode Register can be used to lower it by dividing the frequency of the single accesses.

The CRCCU scrolls the defined memory area using ascending addresses.

In order to compute the CRC for a memory size larger than 256 Kbytes or for non-contiguous memory area, it is possible to re-enable the CRCCU on the new memory area and the CRC will be updated accordingly. Use the RESET field of the CRCCU_CR to reset the CRCCU Status Register to its default value (0xFFFFFFFF).

35.6 Transfer Control Registers Memory Mapping

Table 35-2. Transfer Control Register Memory Mapping

Offset	Register	Name	Access	Reset
CRCCU_DSCR + 0x0	CRCCU Transfer Address Register	TR_ADDR	Read/Write	0x00000000
CRCCU_DSCR + 0x4	CRCCU Transfer Control Register	TR_CTRL	Read/Write	0x00000000
CRCCU_DSCR + 0xC–0x10	Reserved	–	–	–
CRCCU_DSCR+0x10	CRCCU Transfer Reference Register	TR_CRC	Read/Write	0x00000000

Note: These registers are memory mapped.

35.6.1 Transfer Address Register

Name: TR_ADDR

Access: Read/Write

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

- **ADDR: Transfer Address**

35.6.2 Transfer Control Register

Name: TR_CTRL

Access: Read/Write

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

- **BTSIZE: Buffer Transfer Size**

- **TRWIDTH: Transfer Width Register**

Value	Name	Description
0	BYTE	The data size is 8-bit
1	HALFWORD	The data size is 16-bit
2	WORD	The data size is 32-bit

- **IEN: Context Done Interrupt Enable (Active Low)**

0: Bit DMAISR of CRCCU_DMA_ISR is set at the end of the current descriptor transfer.

1: Bit DMAISR of CRCCU_DMA_ISR remains cleared.

35.6.3 Transfer Reference Register

Name: TR_CRC

Access: Read/Write

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

- **REFCRC: Reference CRC**

When Compare mode is enabled, the checksum is compared with this field.

35.7 Cyclic Redundancy Check Calculation Unit (CRCCU) User Interface

Table 35-3. Register Mapping

Offset	Register	Name	Access	Reset
0x000	CRCCU Descriptor Base Register	CRCCU_DSCR	Read/Write	0x00000000
0x004	Reserved	–	–	–
0x008	CRCCU DMA Enable Register	CRCCU_DMA_EN	Write-only	–
0x00C	CRCCU DMA Disable Register	CRCCU_DMA_DIS	Write-only	–
0x010	CRCCU DMA Status Register	CRCCU_DMA_SR	Read-only	0x00000000
0x014	CRCCU DMA Interrupt Enable Register	CRCCU_DMA_IER	Write-only	–
0x018	CRCCU DMA Interrupt Disable Register	CRCCU_DMA_IDR	Write-only	–
0x001C	CRCCU DMA Interrupt Mask Register	CRCCU_DMA_IMR	Read-only	0x00000000
0x020	CRCCU DMA Interrupt Status Register	CRCCU_DMA_ISR	Read-only	0x00000000
0x024–0x030	Reserved	–	–	–
0x034	CRCCU Control Register	CRCCU_CR	Write-only	–
0x038	CRCCU Mode Register	CRCCU_MR	Read/Write	0x00000000
0x03C	CRCCU Status Register	CRCCU_SR	Read-only	0xFFFFFFFF
0x040	CRCCU Interrupt Enable Register	CRCCU_IER	Write-only	–
0x044	CRCCU Interrupt Disable Register	CRCCU_IDR	Write-only	–
0x048	CRCCU Interrupt Mask Register	CRCCU_IMR	Read-only	0x00000000
0x004C	CRCCU Interrupt Status Register	CRCCU_ISR	Read-only	0x00000000
0x050–0x0FC	Reserved	–	–	–

35.7.1 CRCCU Descriptor Base Address Register

Name: CRCCU_DSCR

Address: 0x40048000

Access: Read/Write

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

- **DSCR: Descriptor Base Address**

DSCR needs to be aligned with 512-byte boundaries.

35.7.2 CRCCU DMA Enable Register

Name: CRCCU_DMA_EN

Address: 0x40048008

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
–	–	–	–	–	–	–	DMAEN

- **DMAEN: DMA Enable**

0: No effect

1: Enable CRCCU DMA channel

35.7.3 CRCCU DMA Disable Register

Name: CRCCU_DMA_DIS

Address: 0x4004800C

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
–	–	–	–	–	–	–	DMADIS

- **DMADIS: DMA Disable**

0: No effect

1: Disable CRCCU DMA channel

35.7.4 CRCCU DMA Status Register

Name: CRCCU_DMA_SR

Address: 0x40048010

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
–	–	–	–	–	–	–	DMASR

- **DMASR: DMA Status**

0: DMA channel disabled

1: DMA channel enabled

35.7.5 CRCCU DMA Interrupt Enable Register

Name: CRCCU_DMA_IER

Address: 0x40048014

Access: Write-only

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

- **DMAIER: Interrupt Enable**

0: No effect

1: Enable interrupt

35.7.6 CRCCU DMA Interrupt Disable Register

Name: CRCCU_DMA_IDR

Address: 0x40048018

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
–	–	–	–	–	–	–	DMAIDR

- **DMAIDR: Interrupt Disable**

0: No effect

1: Disable interrupt

35.7.7 CRCCU DMA Interrupt Mask Register

Name: CRCCU_DMA_IMR

Address: 0x4004801C

Access: Read-only

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

- **DMAIMR: Interrupt Mask**

0: Buffer Transfer Completed interrupt disabled

1: Buffer Transfer Completed interrupt enabled

35.7.8 CRCCU DMA Interrupt Status Register

Name: CRCCU_DMA_ISR

Address: 0x40048020

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
–	–	–	–	–	–	–	DMAISR

- **DMAISR: Interrupt Status**

0: DMA buffer transfer has not yet started or transfer still in progress

1: DMA buffer transfer has terminated. This flag is reset after read.

35.7.9 CRCCU Control Register

Name: CRCCU_CR

Address: 0x40048034

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
–	–	–	–	–	–	–	RESET

- **RESET: CRC Computation Reset**

0: No effect

1: Sets the CRCCU_SR to 0xFFFFFFFF

35.7.10 CRCCU Mode Register

Name: CRCCU_MR

Address: 0x40048038

Access: Read/Write

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

- **ENABLE: CRC Enable**

Always write a 1 to this bit.

- **COMPARE: CRC Compare**

If set to one, this bit indicates that the CRCCU DMA will compare the CRC computed on the data stream with the value stored in the TR_CRC reference register. If a mismatch occurs, the ERRISR bit in the CRCCU_ISR is set.

- **PTYPE: Primitive Polynomial**

Value	Name	Description
0	CCITT8023	Polynom 0x04C11DB7
1	CASTAGNOLI	Polynom 0x1EDC6F41
2	CCITT16	Polynom 0x1021

- **DIVIDER: Request Divider**

CRCCU DMA performs successive transfers. It is possible to reduce the bandwidth drained by the CRCCU DMA by programming the DIVIDER field. The transfer request frequency is divided by 2^(DIVIDER+1).

- **BITORDER: Precomputation Bit Swap Operation of the CRC**

Value	Name	Description
0	MSBFIRST	CRC computation is performed from the most significant bit to the least significant bit
1	LSBFIRST	CRC computation is performed from the least significant bit to the most significant bit

35.7.11 CRCCU Status Register

Name: CRCCU_SR

Address: 0x4004803C

Access: Read-only

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

- **CRC: Cyclic Redundancy Check Value**

This register can not be read if the COMPARE bit in the CRCCU_MR is set to true.

35.7.12 CRCCU Interrupt Enable Register

Name: CRCCU_IER

Address: 0x40048040

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
–	–	–	–	–	–	–	ERRIER

- **ERRIER: CRC Error Interrupt Enable**

0: No effect

1: Enable interrupt

35.7.13 CRCCU Interrupt Disable Register

Name: CRCCU_IDR

Address: 0x40048044

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
–	–	–	–	–	–	–	ERRIDR

- **ERRIDR: CRC Error Interrupt Disable**

0: No effect

1: Disable interrupt

35.7.14 CRCCU Interrupt Mask Register

Name: CRCCU_IMR

Address: 0x40048048

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
–	–	–	–	–	–	–	ERRIMR

- **ERRIMR: CRC Error Interrupt Mask**

0: Interrupt disabled

1: Interrupt enabled

35.7.15 CRCCU Interrupt Status Register

Name: CRCCU_ISR

Address: 0x4004804C

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
–	–	–	–	–	–	–	ERRISR

- **ERRISR: CRC Error Interrupt Status**

0: Interrupt disabled

1: Interrupt enabled

36. USB Host Port (UHP)

36.1 Description

The USB Host Port (UHP) interfaces the USB with the host application. It handles the Open HCI protocol (Open Host Controller Interface) as well as USB v2.0 Full-speed and Low-speed protocols.

The USB Host Port integrates a root hub and transceivers on downstream ports. It provides several high-speed half-duplex serial communication ports at a baud rate of 12 Mbit/s. Up to 127 USB devices (printer, camera, mouse, keyboard, disk, etc.) and the USB hub can be connected to the USB host in the USB “tiered star” topology.

The USB Host Port controller is fully compliant with the OpenHCI specification. The USB Host Port User Interface (registers description) can be found in the Open HCI Rev 1.0 Specification.

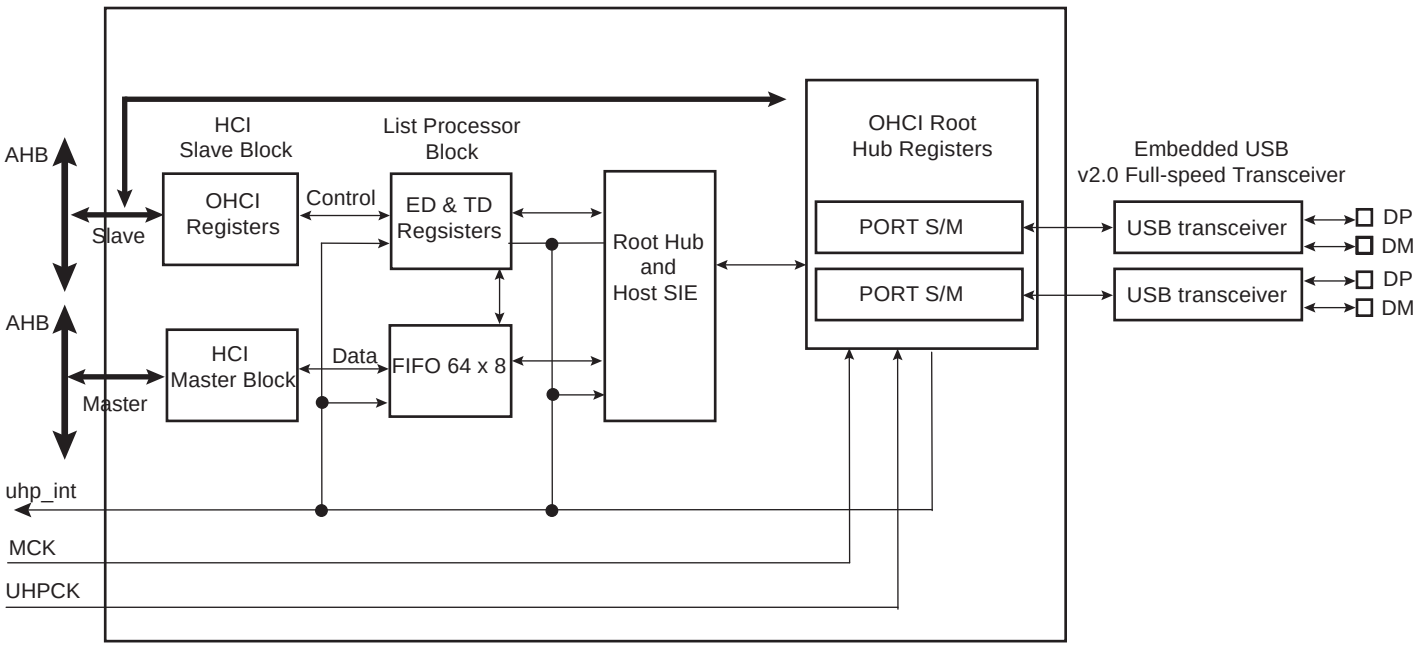
This means that all standard class devices are automatically detected and available to the user application. As an example, integrating an HID (Human Interface Device) class driver provides a plug & play feature for all USB keyboards and mouses.

36.2 Embedded Characteristics

- Compliant with OpenHCI Rev 1.0 Specification
- Compliant with USB V2.0 Full-speed and Low-speed Specification
- Supports Both Low-speed 1.5 Mbps and Full-speed 12 Mbps USB Devices
- Root Hub Integrated with 1 Downstream USB Ports
- Embedded USB Transceivers
- Supports Power Management

36.3 Block Diagram

Figure 36-1. Block Diagram



Access to the USB host operational registers is achieved through the AHB bus slave interface. The OpenHCI host controller initializes master DMA transfers through the ASB bus master interface as follows:

- Fetches endpoint descriptors and transfer descriptors
- Access to endpoint data from system memory
- Access to the HC communication area
- Write status and retire transfer Descriptor

Memory access errors (abort, misalignment) lead to an “UnrecoverableError” indicated by the corresponding flag in the host controller operational registers.

The USB root hub is integrated in the USB host. Several USB downstream ports are available. The number of downstream ports can be determined by the software driver reading the root hub’s operational registers. Device connection is automatically detected by the USB host port logic.

USB physical transceivers are integrated in the product and driven by the root hub’s ports.

36.4 Product Dependencies

36.4.1 I/O Lines

DPs and DMs are not controlled by any PIO controllers. The embedded USB physical transceivers are controlled by the USB host controller.

36.4.2 Power Management

The USB host controller requires a 48 MHz clock. This clock must be generated by a PLL with a correct accuracy of ± 0.25%.

Thus the USB device peripheral receives two clocks from the Power Management Controller (PMC): the Master clock (MCK) used to drive the peripheral user interface (MCK domain) and the UHPCLK 48 MHz clock used to interface with the bus USB signals (Recovered 12 MHz domain).

36.4.3 Interrupt Sources

The USB host interface has an interrupt line connected to the interrupt controller.

Handling USB host interrupts requires programming the interrupt controller before configuring the UHP.

Table 36-1. Peripheral IDs

Instance	ID
UHP	47

36.5 Functional Description

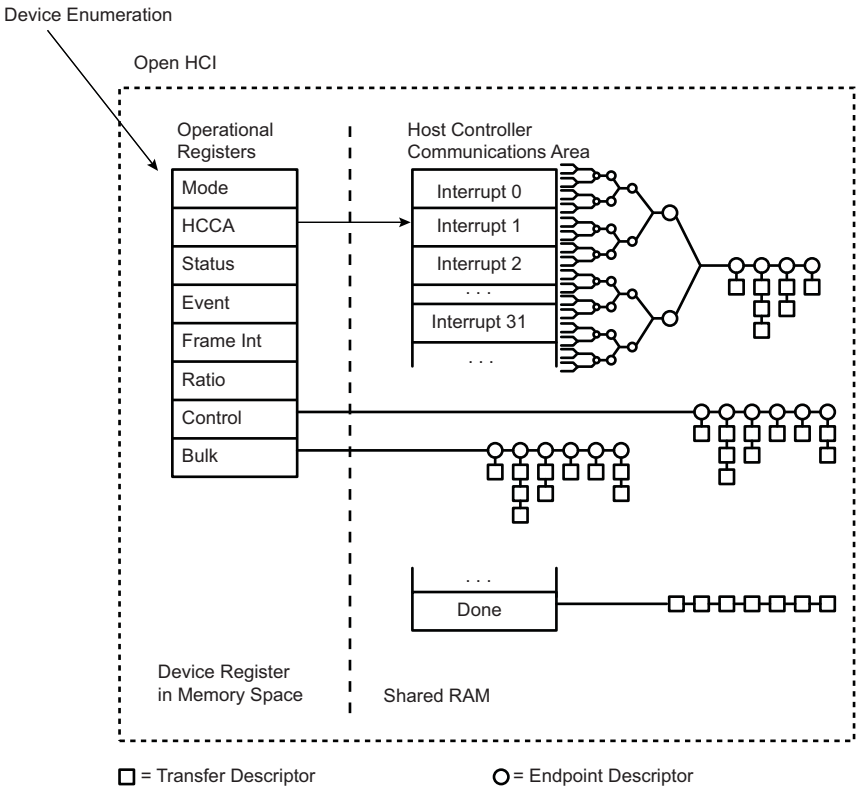
Refer to the Open Host Controller Interface Specification for USB Release 1.0.a.

36.5.1 Host Controller Interface

There are two communication channels between the Host Controller and the Host Controller Driver. The first channel uses a set of operational registers located on the USB Host Controller. The Host Controller is the target for all communications on this channel. The operational registers contain control, status and list pointer registers. They are mapped in the memory mapped area. Within the operational register set there is a pointer to a location in the processor address space named the Host Controller Communication Area (HCCA). The HCCA is the second communication channel. The host controller is the master for all communication on this channel. The HCCA contains the head pointers to the interrupt Endpoint Descriptor lists, the head pointer to the done queue and status information associated with start-of-frame processing.

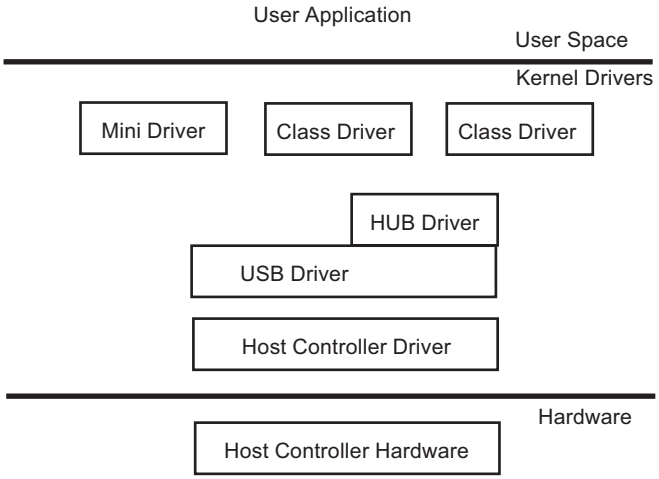
The basic building blocks for communication across the interface are Endpoint Descriptors (ED, 4 double words) and Transfer Descriptors (TD, 4 or 8 double words). The host controller assigns an Endpoint Descriptor to each endpoint in the system. A queue of Transfer Descriptors is linked to the Endpoint Descriptor for the specific endpoint.

Figure 36-2. USB Host Communication Channels



36.5.2 Host Controller Driver

Figure 36-3. USB Host Drivers

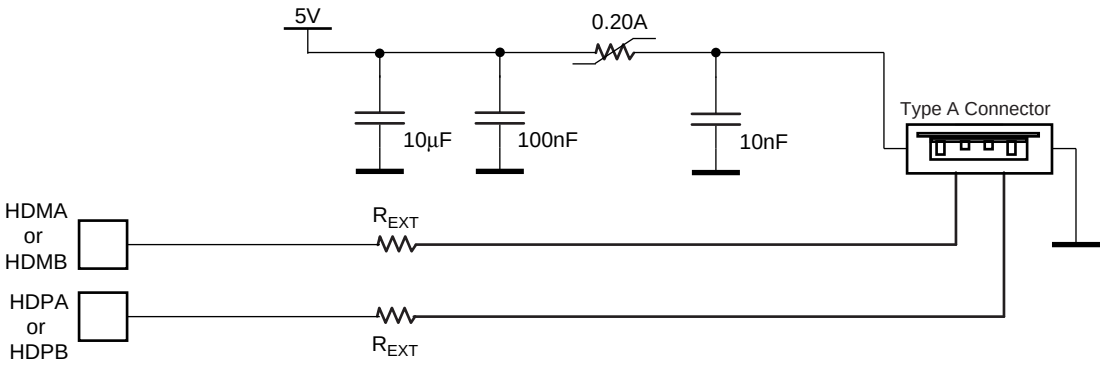


USB Handling is done through several layers as follows:

- Host controller hardware and serial engine—Transmits and receives USB data on the bus.
- Host controller driver—Drives the Host controller hardware and handles the USB protocol.
- USB Bus driver and hub driver—Handles USB commands and enumeration. Offers a hardware independent interface.
- Mini driver—Handles device specific commands.
- Class driver—Handles standard devices. This acts as a generic driver for a class of devices, e.g., the HID driver.

36.6 Typical Connection

Figure 36-4. Board Schematic to Interface UHP Device Controller



A termination serial resistor must be connected to HDP and HDM. Refer to the section “Electrical Characteristics” for definition of the resistor value (R_{EXT}).

36.7 USB Host Port (UHP) User Interface

Most of the host controller (HC) registers are OHCI operational registers, defined by the OHCI Specification for USB. Four additional registers not specified by the OHCI Specification for USB provide additional information about the USB1.1 host controller state. The USB1.1 host controller registers can be accessed in user and supervisor modes.

To enhance code reusability with possible future versions of the USB1.1 host controller, reads and writes to reserved USB1.1 host controller register addresses are to be avoided. Unless otherwise specified, when writing registers that have reserved bits, read-modify-write operations must be used so that the reserved bits are written with their previous values.

Table 36-2. Register Mapping

Offset	Register	Name	Access	Reset
0x00	OHCI Revision Number Register	UHP_HCREVISION	Read-only	0x0000 0010
0x04	HC Operating Mode Register	UHP_HCCONTROL	Read/Write	0x0000 0000
0x08	HC Command and Status Register	UHP_HCCOMMANDSTATUS	Read/Write	0x0000 0000
0x0C	HC Interrupt and Status Register	UHP_HCINTERRUPTSTATUS	Read/Write	0x0000 0000
0x10	HC Interrupt Enable Register	UHP_HCINTERRUPTENABLE	Read/Write	0x0000 0000
0x14	HC Interrupt Disable Register	UHP_HCINTERRUPTDISABLE	Read/Write	0x0000 0000
0x18	HC HCCA Address Register(1)	UHP_HCHCCA	Read/Write	0x0000 0000
0x1C	HC Current Periodic Register(1)	UHP_HCPERIODCURRENTED	Read-only	0x0000 0000
0x20	HC Head Control Register(1)	UHP_HCCONTROLHEADED	Read/Write	0x0000 0000
0x24	HC Current Control Register(1)	UHP_HCCONTROLCURRENTED	Read/Write	0x0000 0000
0x28	HC Head Bulk Register(1)	UHP_HCBULKHEADED	Read/Write	0x0000 0000
0x2C	HC Current Bulk Register(1)	UHP_HCBULKCURRENTED	Read/Write	0x0000 0000
0x30	HC Head Done Register(1)	UHP_HCDONEHEAD	Read-only	0x0000 0000
0x34	HC Frame Interval Register	UHP_HCFMINTERVAL	Read/Write	0x0000 2EDF
0x38	HC Frame Remaining Register	UHP_HCFMREMAINING	Read-only	0x0000 0000
0x3C	HC Frame Number Register	UHP_HCFMNUMBER	Read-only	0x0000 0000
0x40	HC Periodic Start Register	UHP_HCPERIODICSTART	Read/Write	0x0000 0000
0x44	HC Low-Speed Threshold Register	UHP_HCLSTHRESHOLD	Read/Write	0x0000 0628
0x48	HC Root Hub A Register	UHP_HCRHDESCRIPTORA	Read/Write	0x0A00 1203
0x4C	HC Root Hub B Register	UHP_HCRHDESCRIPTORB	Read/Write	0x0000 0000
0x50	HC Root Hub Status Register	UHP_HCRHSTATUS	Read/Write	0x0000 0000
0x54	HC Port 1 Status and Control Register(2)	UHP_HCRHPORTSTATUS1	Read/Write	0x0000 0100
0x58	HC Port 2 Status and Control Register(3)	UHP_HCRHPORTSTATUS2	Read/Write	0x0000 0100

Notes:
1. Restrictions apply to the physical addresses used in these registers.
2. Connected to the integrated USB1.1 phy pins (DM, DP).
3. Although the controller implements two ports, the second port cannot be used.

36.7.1 UHP OHCI Revision Number Register

Name: UHP_HCREVISION

Address: 0x4004C000

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
REV							

- **REV: OHCI Revision Number**

10h: This read-only field contains the BCD representation of the version of the HCI specification that is implemented by this HC. For example, a value of 11h corresponds to version 1.1. All of the HC implementations that are compliant with this specification will have a value of 10h.

36.7.2 UHP HC Operating Mode Register

Name: UHP_HCCONTROL

Address: 0x4004C004

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
–	–	–	–	–	RWE	RWC	IR
7	6	5	4	3	2	1	0
HCFS		BLE	CLE	IE	PLE	CBSR	

This register controls the operating mode of the USB1.1 host controller.

- **CBSR: Control/Bulk Service Ratio**

Specifies the ratio between control and bulk EDs processed in a frame.

0: 1 control ED per bulk ED.

1h: 2 control EDs per bulk ED.

2h: 3 control EDs per bulk ED.

3h: 4 control EDs per bulk ED.

- **PLE: Periodic List Enable**

0: Periodic ED lists are not processed. Periodic list processing is disabled beginning with the next frame.

1: Enables processing of the periodic ED lists. Periodic list processing begins in the next frame.

- **IE: Isochronous Enable**

0: Isochronous EDs are not processed. The USB1.1 host controller checks this bit every time it finds an isochronous ED in the periodic list.

1: Enables processing of isochronous EDs in the next frame, if not in the current frame.

- **CLE: ControL List Enable**

0: The control ED list is not processed in the next 1 ms frame. The host controller driver can modify the control ED list. If the driver removes the ED pointed to by the UHP_HCCONTROLCURRENTED register from the ED list, it must update the UHP_HCCONTROLCURRENTED register to point to a current ED before it reenables the control list.

1: Enables processing of the control ED list. The UHP_HCCONTROLHEADED register must be 0 or point to a valid ED before setting this bit. The UHP_HCCONTROLCURRENTED register must be 0 or point to a valid ED before setting this bit.

- **BLE: Bulk List Enable**

0: The bulk ED list is not processed in the next 1 ms frame. The host controller driver can modify the bulk ED list. If the driver removes the ED pointed to by the UHP_HCBULKCURRENTED register from the ED list, it must update the UHP_HCBULKCURRENTED register to point to a current ED before it reenables the bulk list.

1: Enables processing of the bulk ED list. The UHP_HCBULKHEADED register must be 0 or point to a valid ED before setting this bit. The UHP_HCBULKCURRENTED register must be 0 or point to a valid ED before setting this bit.

- **HCFS: Host Controller Functional State**

A transition to USB operational causes SOF generation to begin in 1 ms. The USB1.1 host controller can automatically transition from USB suspend to USB resume, if a downstream resume is received. The USB1.1 host controller enters USB suspend after a software reset. The USB1.1 host controller enters USB reset after a hardware reset. The USB reset state resets the root hub and causes downstream signaling of USB reset.

0: USB reset.

1h: USB resume.

2h: USB operational.

3h: USB suspend.

- **IR: Interrupt Routing**

0: The USB1.1 host controller does not provide an SMI interrupt. This bit must be 0 to allow the USB1.1 host controller interrupt to propagate to the MPU level 2 interrupt controller.

- **RWC: Remote Wakeup Connected**

0–1: This bit indicates whether HC supports remote wakeup signaling. If remote wakeup is supported and used by the system it is the responsibility of system firmware to set this bit during POST. HC clears the bit upon a hardware reset but does not alter it upon a software reset. Remote wakeup signaling of the host system is host-bus-specific and is not described in this specification.

- **RWE: Remote Wakeup Enable**

0–1: This bit is used by HCD to enable or disable the remote wakeup feature upon the detection of upstream resume signaling. When this bit is set and the ResumeDetected bit in HcInterruptStatus is set, a remote wakeup is signaled to the host system. Setting this bit has no impact on the generation of hardware interrupt.

36.7.3 UHP HC Command and Status Register

Name: UHP_HCCOMMANDSTATUS

Address: 0x4004C008

Access: Read/Write

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

This register shows the current state of the host controller and accepts commands from the host controller driver.

- **HCR: Host Controller Reset (read/write)**

0: No effect.

1: Initiates a software reset of the USB1.1 host controller. This transitions the USB1.1 host controller to the USB suspend state. This resets most USB1.1 host controller OHCI registers. OHCI register accesses must not be attempted until a read of this bit returns a 0. A write of 1 to this bit does not reset the root hub and does not signal USB reset to downstream USB functions.

- **CLF: Control List Filled (read/write)**

0–1: The host controller driver must set this bit if it modifies the control list to include new TDs. If the UHP_HCCONTROLHEADED register is 0, the USB1.1 host controller does not begin processing control list EDs unless this bit is set. When the USB1.1 host controller sees this bit set and begins processing the control list, it clears this bit to 0.

- **BLF: Bulk List Filled (read/write)**

0–1: The host controller driver must set this bit if it modifies the bulk list to include new TDs. If the UHP_HCBULKCURRENTED register is 0, the USB1.1 host controller does not begin processing bulk list EDs unless this bit is set. When the USB1.1 host controller sees this bit set and begins processing the bulk list, it clears this bit to 0.

- **OCR: Ownership Change Request (read/write)**

0–1: The host controller driver sets this bit to gain ownership of the host controller. The processor does not support SMI interrupts, so no ownership change interrupt occurs.

- **SOC: Scheduling Overrun Count (read-only)**

0–3: Counts the number of times a scheduling overrun occurs. This count is incremented even if the host controller driver has not acknowledged any previous pending scheduling overrun interrupt.

36.7.4 UHP HC Interrupt and Status Register

Name: UHP_HCINTERRUPTSTATUS

Address: 0x4004C00C

Access: Read/Write

31	30	29	28	27	26	25	24
–	OC	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	RHSC	FNO	UE	RD	SF	WDH	SO

This register reports the status of the USB1.1 host controller internal interrupt sources.

- **SO: Scheduling Overrun (read/write, write ‘1’ to clear)**

A write of 1 clears this bit; a write of 0 has no effect.

0: A scheduling overrun has not occurred.

1: A scheduling overrun has occurred.

- **WDH: Write Done Head (read/write, write ‘1’ to clear)**

The host controller driver must read the value from the UHP_HCDONEHEAD register before writing 1 to this bit. A write of 1 clears this bit; a write of 0 has no effect.

0: USB1.1 host controller has not updated the UHP_HCDONEHEAD register.

1: USB1.1 host controller has updated the UHP_HCDONEHEAD register.

- **SF: Start of Frame (read/write, write ‘1’ to clear)**

A write of 1 clears this bit; a write of 0 has no effect.

0: A SOF has not been issued.

1: A SOF has been issued.

- **RD: Resume Detected (read/write, write ‘1’ to clear)**

A write of 1 clears this bit; a write of 0 has no effect.

0: A downstream device has not issued a resume request.

1: A downstream device has issued a resume request.

- **UE: Unrecoverable Error (read/write, write ‘1’ to clear)**

A write of 1 clears this bit; a write of 0 has no effect.

0: An unrecoverable error has not occurred.

1: An unrecoverable error has occurred on the OCPI bus, or that an isochronous TD PSW field condition code was not set to Not Accessed when the USB1.1 host controller attempted to perform a transfer using that PSW/offset pair.

- **FNO: Frame Number Overflow (read/write, write ‘1’ to clear)**

A write of 1 clears this bit; a write of 0 has no effect.

0: A frame number overflow has not occurred.

1: A frame number overflow has occurred.

- **RHSC: Root Hub Status Change (read/write, write ‘1’ to clear)**

A write of 1 clears this bit; a write of 0 has no effect.

0: A root hub status change has not occurred.

1: A root hub status change has occurred.

- **OC: Ownership Change (read-only)**

0–1: This bit is set by HC when HCD sets the OwnershipChangeRequest field in UHP_HCCOMMANDSTATUS. This event, when unmasked, will always generate an System Management Interrupt (SMI) immediately. This bit is tied to 0 when the SMI pin is not implemented.

36.7.5 UHP HC Interrupt Enable Register

Name: UHP_HCINTERRUPTENABLE

Address: 0x4004C010

Access: Read/Write

31	30	29	28	27	26	25	24
MIE	OC	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	RHSC	FNO	UE	RD	SF	WDH	SO

This register enables various OHCI interrupt sources to generate interrupts to the level 2 interrupt controller.

- **SO: Scheduling Overrun (read/write, write ‘1’ to set)**

A write of 1 sets this bit; a write of 0 has no effect. A write of 1 to the corresponding bit in the UHP_HCINTERRUPTDISABLE register clears this bit.

0: Scheduling overrun interrupts do not propagate.

1: When MIE is 1, allows scheduling overrun interrupts to propagate to the level 2 interrupt controller.

- **WDH: Write Done Head (read/write, write ‘1’ to set)**

A write of 1 sets this bit; a write of 0 has no effect. A write of 1 to the corresponding bit in the UHP_HCINTERRUPTDISABLE register clears this bit.

0: Write done head interrupts do not propagate.

1: When MIE is 1, allows write done head interrupts to propagate to the level 2 interrupt controller.

- **SF: Start of Frame (read/write, write ‘1’ to set)**

A write of 1 sets this bit; a write of 0 has no effect. A write of 1 to the corresponding bit in the UHP_HCINTERRUPTDISABLE register clears this bit.

0: Start of frame interrupts do not propagate.

1: When MIE is 1, allows start of frame interrupts to propagate to the level 2 interrupt controller.

- **RD: Resume Detected (read/write, write ‘1’ to set)**

A write of 1 sets this bit; a write of 0 has no effect. A write of 1 to the corresponding bit in the UHP_HCINTERRUPTDISABLE register clears this bit.

0: Resume detected interrupts do not propagate.

1: When MIE is 1, allows resume detected interrupts to propagate to the level 2 interrupt controller.

- **UE: Unrecoverable Error (read/write, write ‘1’ to set)**

A write of 1 sets this bit; a write of 0 has no effect. A write of 1 to the corresponding bit in the UHP_HCINTERRUPTDISABLE register clears this bit.

0: Unrecoverable error interrupts do not propagate.

1: When MIE is 1, allows unrecoverable error interrupts to propagate to the level 2 interrupt controller.

- **FNO: Frame Number Overflow (read/write, write ‘1’ to set)**

A write of 1 sets this bit; a write of 0 has no effect. A write of 1 to the corresponding bit in the UHP_HCINTERRUPTDISABLE register clears this bit.

0: Frame number overflow interrupts do not propagate.

1: When MIE is 1, allows frame number overflow interrupts to propagate to the level 2 interrupt controller.

- **RHSC: Root Hub Status Change (read/write, write ‘1’ to set)**

A write of 1 sets this bit; a write of 0 has no effect. A write of 1 to the corresponding bit in the UHP_HCINTERRUPTDISABLE register clears this bit.

0: Root hub status change interrupts do not propagate.

1: When MIE is 1, allows root hub status change interrupts to propagate to the level 2 interrupt controller.

- **OC: Ownership Change (read-only)**

0: Ignore.

1: Enable interrupt generation due to Ownership Change.

- **MIE: Master Interrupt Enable (read/write, write ‘1’ to set)**

A write of 1 sets this bit; a write of 0 has no effect. A write of 1 to the corresponding bit in the UHP_HCINTERRUPTDISABLE register clears this bit.

0: OHCI interrupt sources are ignored and USB1.1 host controller interrupts are not propagated to the level 2 interrupt controller.

1: Allows other enabled OHCI interrupt sources to propagate to the level 2 interrupt controller.

36.7.6 UHP HC Interrupt Disable Register

Name: UHP_HCINTERRUPTDISABLE

Address: 0x4004C014

Access: Read/Write

31	30	29	28	27	26	25	24
MIE	OC	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	RHSC	FNO	UE	RD	SF	WDH	SO

This register is used to clear bits in the UHP_HCINTERRUPTENABLE register.

- **SO: Scheduling Overrun (read/write)**

Read always returns 0.

0: No effect.

1: Clears the SO bit in the UHP_HCINTERRUPTENABLE register.

- **WDH: Write Done Head (read/write)**

Read always returns 0.

0: No effect.

1: Clears the WDH bit in the UHP_HCINTERRUPTENABLE register.

- **SF: Start of Frame (read/write)**

Read always returns 0.

0: No effect.

1: Clears the SF bit in the UHP_HCINTERRUPTENABLE register.

- **RD: Resume Detected (read/write)**

Read always returns 0.

0: No effect.

1: Clears the RD bit in the UHP_HCINTERRUPTENABLE register.

- **UE: Unrecoverable Error (read/write)**

Read always returns 0.

0: No effect.

1: Clears the UE bit in the UHP_HCINTERRUPTENABLE register.

- **FNO: Frame Number Overflow (read/write)**

Read always returns 0.

0: No effect.

1: Clears the FNO bit in the UHP_HCINTERRUPTENABLE register.

- **RHSC: Root Hub Status Change (read/write)**

Read always returns 0.

0: No effect.

1: Clears the RHSC bit in the UHP_HCINTERRUPTENABLE register.

- **OC: Ownership Change (read-only)**

0: Ignore.

1: Disable interrupt generation due to Ownership Change.

- **MIE: Master Interrupt Enable (read/write)**

Read always returns 0.

0: No effect.

1: Clears the MIE bit in the UHP_HCINTERRUPTENABLE register.

36.7.7 UHP HC HCCA Address Register

Name: UHP_HCHCCA

Address: 0x4004C018

Access: Read/Write

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

This register defines the physical address of the beginning of the HCCA.

- **HCCA: Physical Address of the Beginning of the HCCA**

0–FF FFFFh: This is the base address of the Host Controller Communication Area.

36.7.8 UHP HC Current Periodic Register

Name: UHP_HCPERIODCURRENTED

Address: 0x4004C01C

Access: Read-only

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

This register defines the physical address of the next endpoint descriptor (ED) on the periodic ED list.

- **PCED: Physical Address of the Current ED on the Periodic ED list**

0–FFF FFFFh: This field represents bits 31:4 of the physical address of the next ED on the periodic ED list. EDs are assumed to begin on a 16-byte aligned address, so bits 3:0 of this pointer are assumed to be 0.

36.7.9 UHP HC Head Control Register

Name: UHP_HCCONTROLHEADED

Address: 0x4004C020

Access: Read/Write

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

This register defines the physical address of the head endpoint descriptor (ED) on the control ED list.

- **CHED: Physical Address of the Head ED on the Control ED list**

0–FFF FFFFh: This field represents bits 31:4 of the physical address of the head ED on the control ED list. EDs are assumed to begin on a 16-byte aligned address, so bits 3:0 of this pointer are assumed to be 0.

36.7.10 UHP HC Current Control Register

Name: UHP_HCCONTROLCURRENTED

Address: 0x4004C024

Access: Read/Write

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

This register defines the physical address of the next endpoint descriptor (ED) on the control ED list.

- **CCED: Physical Address of the Current ED on the Control ED List**

0–FFF FFFFh: This field represents bits 31:4 of the physical address of the next ED on the control ED list. EDs are assumed to begin on a 16-byte aligned address, so bits 3:0 of this pointer are assumed to be 0.

A value of 0 indicates that the USB1.1 host controller has reached the end of the control ED list without finding any transfers to process. This register is automatically updated by the USB1.1 host controller.

36.7.11 UHP HC Head Bulk Register

Name: UHP_HCBULKHEADED

Address: 0x4004C028

Access: Read/Write

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

This register defines the physical address of the head endpoint descriptor (ED) on the bulk ED list.

- **BHED: Physical Address of the Head ED on the Bulk ED List**

0–FFF FFFFh: This field represents bits 31:4 of the physical address of the head ED on the bulk ED list. EDs are assumed to begin on a 16-byte aligned address, so bits 3:0 of this pointer are assumed to be 0.

36.7.12 UHP HC Current Bulk Register

Name: UHP_HCBULKCURRENTED

Address: 0x4004C02C

Access: Read/Write

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

This register defines the physical address of the next endpoint descriptor (ED) on the bulk ED list.

- **BCED: Physical Address of the Current ED on the Bulk ED List**

0–FFF FFFFh: This field represents bits 31:4 of the physical address of the next ED on the bulk ED list. EDs are assumed to begin on a 16-byte aligned address, so bits 3:0 of this pointer are assumed to be 0.

A value of 0 indicates that the USB1.1 host controller has reached the end of the bulk ED list without finding any transfers to process. This register is automatically updated by the USB1.1 host controller.

36.7.13 UHP HC Head Done Register

Name: UHP_HCDONEHEAD

Address: 0x4004C030

Access: Read-only

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

This register defines the physical address of the current head of the done TD queue.

- **DH: Physical Address of the Last TD that has added to the done queue**

0–FFF FFFFh: This field represents bits 31:4 of the physical address of the top TD on the done TD queue. TDs are assumed to begin on a 16-byte aligned address, so bits 3:0 of this pointer are assumed to be 0.

A value of 0 indicates that there are no TDs on the done queue. This register is automatically updated by the USB1.1 host controller.

36.7.14 UHP HC Frame Interval Register

Name: UHP_HCFMINTERVAL

Address: 0x4004C034

Access: Read/Write

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

This register defines the number of 12-MHz clock pulses in each USB frame.

- **FRAMEINTERVAL: Frame Interval**

0–3FFFh: Number of 12-MHz clocks in the USB frame. Nominally, this is set to 11,999 (2EDFh) to give a 1-ms frame. The host controller driver can make minor changes to this field to attempt to manually synchronize with another clock source.

- **FSMPS: Largest Data Packet**

0–7FFFh: Largest data packet size allowed for full-speed packets, in bit times.

- **FIT: Frame Interval Toggle**

0–1: The host controller driver must toggle this bit any time it changes the frame interval field.

36.7.15 UHP HC Frame Remaining Register

Name: UHP_HCFMREMAINING

Address: 0x4004C038

Access: Read-only

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

This register reports the number of full-speed bit times remaining in the current frame.

- **FR: Frame Remaining**

0–3FFFh: The number of full-speed bit times remaining in the current frame. This field is automatically reloaded with the frame interval (FI) value in the UHP_HCFMINTERVAL register at the beginning of every frame.

- **FRT: Frame Remaining Toggle**

0–1: This bit is loaded with the frame interval toggle bit every time the USB1.1 host controller loads the frame interval field into the frame remaining field.

36.7.16 UHP HC Frame Number Register

Name: UHP_HCFMNUMBER

Address: 0x4004C03C

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
FN							
7	6	5	4	3	2	1	0
FN							

This register reports the current USB frame number.

- **FN: Frame Number**

0–FFFFh: This field reports the current USB frame number. It is incremented when the frame remaining field is reloaded with the frame interval (FI) value in the UHP_HCFMINTERVAL register. Frame number automatically rolls over from FFFFh to 0. After frame number is incremented, its new value is written to the HCCA and the USB1.1 host controller sets the SOF interrupt status bit and begins processing the ED lists.

36.7.17 UHP HC Periodic Start Register

Name: UHP_HCPERIODICSTART

Address: 0x4004C040

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
–	–	PS					
7	6	5	4	3	2	1	0
PS							

This register defines the position within the USB frame where endpoint descriptors (EDs) on the periodic list have priority over EDs on the bulk and control lists.

- **PS: Periodic Start**

0–3FFFh: The host controller driver must program this value to be about 10% less than the frame interval (FI) value in the UHP_HCFMINTERVAL register, so that control and bulk EDs have priority for the first 10% of the frame; then periodic EDs have priority for the remaining 90% of the frame.

36.7.18 UHP HC Low-Speed Threshold Register

Name: UHP_HCLSTHRESHOLD

Address: 0x4004C044

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
–	–	LST					
7	6	5	4	3	2	1	0
LST							

This register defines the latest time in a frame that the USB1.1 host controller can begin a low-speed packet.

- **LST: Low-Speed Threshold**

0–3FFFh: This field defines the number of full-speed bit times in the frame after which the USB1.1 host controller cannot start an 8-byte low-speed packet. The USB1.1 host controller only begins a low-speed transaction if the frame remaining (FR) value in the UHP_HCFMREMAINING register is greater than the low-speed threshold.

The host controller driver must set this field to a value that ensures that an 8-byte low-speed TD completes before the end of the frame. When set, the host controller driver must not change the value.

36.7.19 UHP HC Root Hub A Register

Name: UHP_HCRHDESCRIPTORA

Address: 0x4004C048

Access: Read/Write

31	30	29	28	27	26	25	24
POTPG							
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	NOCP	OCPM	DT	NPS	PSM
7	6	5	4	3	2	1	0
NDP							

This register defines several aspects of the USB1.1 host controller root hub functionality.

- **NDP: Number of Downstream Ports (read-only)**

0–FFh: The USB signal multiplexing mode and top-level pin multiplexing features can place the device in a mode where 0, 1, 2, or 3 of the USB1.1 host controller downstream ports are usable. This register reports three ports, regardless of USB signal multiplexing mode and top-level pin multiplexing mode.

- **PSM: Power Switching Mode (read/write)**

0: Because the device does not provide signals from the USB1.1 host controller to control external VBUS switching, this bit defaults to 0 that indicates that all ports are powered at the same time.

- **NPS: No Power Switching (read/write)**

1: Because the device does not provide connections from the USB1.1 host controller to control external VBUS switching, this bit defaults to 1 that indicates that VBUS power switching is not supported and that power is available to all downstream ports when the USB1.1 host controller is powered. This bit has no relationship to the OTG controller register bits that relate to VBUS. System software can update this register to simplify host controller driver and/or OTG driver coding.

- **DT: Device Type (read-only)**

0: This bit is always 0, which indicates that the USB1.1 host controller implemented is not a compound device.

- **OCPM: Overcurrent Protection Mode (read/write)**

0: Because the device does not provide host controller overcurrent protection input signals, this bit has no effect. This bit has no relationship to the OTG controller register bits that relate to VBUS.

- **NOCP: No Overcurrent Protection (read/write)**

1: Because the device does not provide signals to allow connection of external overcurrent indication signals to the USB1.1 host controller, this bit defaults to 1 that indicates that the USB1.1 host controller does not implement overcurrent protection inputs. This bit has no relationship to the OTG controller register bits that relate to VBUS.

- **POTPG: Power-On to Power-good Time (read/write)**

0–FFh: Defines the minimum amount of time (2 ms x POTPG) between the USB1.1 host controller turning on power to a downstream port and when the USB1.1 host can access the downstream device. This field has no effect on USB1.1 host controller operation. After turning on power to a port, the USB1.1 host controller driver must delay the amount of time implied by POTPG before attempting to reset an attached downstream device. The required amount of time is implementation-specific and must be calculated based on the amount of time the VBUS supply takes to provide valid VBUS to a worst-case downstream USB function controller. The implementation-specific value must be computed and then written to this register before the USB1.1 host controller driver is initialized. Because the device does not provide a direct control from the USB1.1 host controller to switch VBUS on and off, this value must take into account any delays caused by other methods of controlling VBUS externally. This field has no relationship to the OTG controller register bits that relate to VBUS. System software can update this register to simplify host controller driver and/or OTG driver coding.

36.7.20 UHP HC Root Hub B Register

Name: UHP_HCRHDESCRIPTORB

Address: 0x4004C04C

Access: Read/Write

31	30	29	28	27	26	25	24
PPCM15	PPCM14	PPCM13	PPCM12	PPCM11	PPCM10	PPCM9	PPCM8
23	22	21	20	19	18	17	16
PPCM7	PPCM6	PPCM5	PPCM4	PPCM3	PPCM2	PPCM1	PPCM0
15	14	13	12	11	10	9	8
DR15	DR14	DR13	DR12	DR11	DR10	DR9	DR8
7	6	5	4	3	2	1	0
DR7	DR6	DR5	DR4	DR3	DR2	DR1	DR0

This register defines several aspects of the USB1.1 host controller root hub functionality.

Note: The device does not provide connections from the USB1.1 host controller to pins to provide external port power switching. Systems that implement port power switching must use other mechanisms to control port power.

- **DR0: Device Removable**

Reserved.

- **DR1: Device Removable Bit for Downstream Port 1**

Defines whether downstream port 1 has a removable or nonremovable device.

0: Downstream port 1 may have a removable device attached.

1: Downstream port 1 has a nonremovable device attached.

- **DR2: Device Removable Bit for Downstream Port 2**

Defines whether downstream port 2 has a removable or nonremovable device.

0: Downstream port 2 may have a removable device attached.

1: Downstream port 2 has a nonremovable device attached.

- **DR3: Device Removable Bit for Downstream Port 3**

Defines whether downstream port 3 has a removable or nonremovable device.

0: Downstream port 3 may have a removable device attached.

1: Downstream port 3 has a nonremovable device attached.

- **DRx[x=4..15]: Device Removable**

Reserved.

- **PPCM0: Port Power Control Mask**

Reserved.

- **PPCM1: Port Power Control Mask for Downstream Port 1**

Defines whether downstream port 1 has port power controlled by the global power control. System software can update these bits to simplify host controller driver and/or OTG driver coding.

0: Global power control is implemented for downstream port 1.

1: Per-port power control is implemented for downstream port 1.

- **PPCM2: Port Power Control Mask for Downstream Port 2**

Defines whether downstream port 2 has port power controlled by the global power control. System software can update these bits to simplify host controller driver and/or OTG driver coding.

0: Global power control is implemented for downstream port 2.

1: Per-port power control is implemented for downstream port 2.

- **PPCM3: Port Power Control Mask for Downstream Port 3**

Defines whether downstream port 3 has port power controlled by the global power control. System software can update these bits to simplify host controller driver and/or OTG driver coding.

0: Global power control is implemented for downstream port 3.

1: Per-port power control is implemented for downstream port 3.

- **PPCMx[x=4..15]: Port Power Control Mask**

Reserved.

36.7.21 UHP HC Root Hub Status Register

Name: UHP_HCRHSTATUS

Address: 0x4004C050

Access: Read/Write

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

This register reports the USB1.1 host controller root hub status.

- **LPS: Local Power Status (read/write)**

0: Because the root hub does not support the local power status feature, this bit defaults to 0 and has no effect. This bit has no relationship to the OTG controller register bits that relate to VBUS. System software can update this register to simplify host controller driver and/or OTG driver coding.

- **OCI: Overcurrent Indicator (read-only)**

0: Because the device does not provide signals for external hardware to report overcurrent status to the USB1.1 host controller, this bit is always 0. This bit has no relationship to the OTG controller register bits that relate to VBUS.

- **DRWE: Device Remote Wakeup Enable (read/write)**

0: A write of 0 has no effect. Connect status change events do not cause a transition from USB suspend to USB resume state and the resume detected interrupt is not changed.

1: A write of 1 sets the device remote wakeup enable bit. This bit enables a connect status change event to be treated as a resume event, which causes a transition from USB suspend to USB resume state and sets the resume detected interrupt status bit.

- **LPSC: Local Power Status Change (read/write)**

0: Because the root hub does not support the local power status feature, this bit defaults to 0 and has no effect. This bit has no relationship to the OTG controller register bits that relate to VBUS. System software can update this register to simplify host controller driver and/or OTG driver coding.

- **OCIC: Overcurrent Indication Change (read/write)**

This bit is automatically set when the overcurrent indicator bit changes. This bit has no relationship to the OTG controller register bits that relate to VBUS. System software can update this register to simplify host controller driver and/or OTG driver coding.

0: No effect.

1: Clears this bit.

- **CRWE: Clear Remote Wakeup Enable (read/write)**

0: No effect.

1: Clears the device remote wakeup enable bit.

36.7.22 UHP HC Port 1 Status and Control Register

Name: UHP_HCRHPORTSTATUS1

Address: 0x4004C054

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	PRSC	OCIC	PSSC	PESC	CSC
15	14	13	12	11	10	9	8
–	–	–	–	–	–	LSDA/CPP	PPS/SPP
7	6	5	4	3	2	1	0
–	–	–	PRS/SPR	POCI/CSS	PSS/SPS	PES/SPE	CCS/CPE

This register reports and controls the state of USB1.1 host port 1.

- **CCS/CPE: Port 1 Current Connection Status/clear Port Enable (read/write)**

If the DR[1] bit in the UHP_HCRHDESCRIPTORB register is set to 1 to indicate a nonremovable USB device on port 1, this bit is set after a root hub reset to inform the system that the device is attached. A write of 1 clears this bit; a write of 0 has no effect.

0: No USB device is attached to port 1.

1: USB device is attached to port 1.

- **PES/SPE: Port 1 Port Enable Status/set Port Enable (read/write)**

A write of 1 to this bit when port 1 current connect status is 1 sets the port 1 port enable status bit; a write of 1 when port 1 current connect status is 0 has no effect; a write of 0 has no effect. This bit is automatically set at completion of port 1 USB reset, if it was not already set before the USB reset completed; and this bit is automatically set at the end of a USB suspend, if the port was not enabled when the USB resume completed.

0: Port 1 is disabled.

1: Port 1 is enabled.

- **PSS/SPS: Port 1 Port Suspend Status/set Port Suspend (read/write)**

A write of 1 to this bit when port 1 current connect status is 1 sets the port 1 port suspend status bit and places port 1 in USB suspend state; a write of 1 when port 1 current connect status is 0 sets the connect status change to inform the USB1.1 host controller driver software of an attempt to suspend a disconnected device; a write of 0 has no effect. This bit is cleared automatically at the end of the USB resume sequence and also at the end of the USB reset sequence.

0: Port 1 is not in the USB suspend state.

1: Port 1 is in the USB suspend state or is in the resume sequence.

- **POCI/CSS: Port 1 port Overcurrent Indicator/clear Suspend Status (read/write)**

A write of 1 to this bit when port 1 port suspend status is 1 causes resume signaling on port 1; a write of 1 when port 1 port suspend status is 0 has no effect; a write of 0 has no effect. The device does not provide inputs for signaling external overcurrent indication to the USB1.1 host controller. Overcurrent monitoring, if required, must be handled through some other mechanism.

0: Port 1 port overcurrent condition has not occurred.

1: Port 1 port overcurrent condition has occurred.

- **PRS/SPR: Port 1 port Reset Status/set Port Reset (read/write)**

A write of 1 to this bit sets the port 1 port reset status bit and causes the USB1.1 host controller to begin signaling USB reset to port 1; a write of 0 has no effect.

0: USB reset is not being sent to port 1.

1: Port 1 is signaling the USB reset.

- **PPS/SPP: Port 1 Port Power Status/set Port Power (read/write)**

The host controller driver can write a 1 to this bit to set the port 1 port power status bit; a write of 0 has no effect. The device does not provide signals from the USB1.1 host controller to control external port power, so if required, USB1.1 host port power control signals must be controlled through other means. Software can track the current power state using the port power status bit and other power control bits, but those bits have no direct effect on external port power control. This bit has no relationship to the OTG controller register bits that relate to VBUS. System software can update this register to simplify host controller driver and/or OTG driver coding.

0: Port 1 power is disabled.

1: Port 1 power is enabled.

- **LSDA/CPP: Port 1 Low-speed Device Attached/clear Port Power (read/write)**

This bit is valid only when port 1 current connect status is 1. The host controller driver can write a 1 to this bit to clear the port 1 port power status bit; a write of 0 has no effect. The USB1.1 host controller does not control external port power using OHCI mechanisms, so, if required, USB1.1 host port power must be controlled through other means. This bit has no relationship to the OTG controller register bits that relate to VBUS. System software can update this register to simplify host controller driver and/or OTG driver coding.

0: Full-speed device is attached to port 1.

1: Low-speed device is attached to port 1.

- **CSC: Port 1 Connect Status Change (read/write, write ‘1’ to clear)**

If the DR[1] bit in the UHP_HCRHDESCRIPTORB register is set to 1 to indicate a nonremovable USB device on port 1, this bit is set only after a root hub reset to inform the system that the device is attached. A write of 1 clears this bit; a write of 0 has no effect.

0: Port 1 current connect status has not changed.

1: Port 1 current connect status has changed due to a connect or disconnect event. If current connect status is 0 when a set port reset, set port enable, or set port suspend write occurs, then this bit is set.

- **PESC: Port 1 Enable Status Change (read/write, write ‘1’ to clear)**

A write of 1 clears this bit; a write of 0 has no effect.

0: Port 1 port enable status has not changed.

1: Port 1 port enable status has changed.

- **PSSC: Port 1 Suspend Status Change (read/write, write ‘1’ to clear)**

A write of 1 clears this bit; a write of 0 has no effect.

0: Port 1 port suspend status has not changed.

1: Port 1 port suspend status has changed. Suspend status is considered to have changed only after the resume pulse, low-speed EOP, and 3-ms synchronization delays have been completed.

- **OCIC: Port 1 Overcurrent Indicator Change (read/write)**

0: Because the device does not provide inputs for signaling external overcurrent indication to the USB1.1 host controller, this bit is always 0. Overcurrent monitoring, if required, must be handled through some other mechanism. This bit has no relationship to the OTG controller register bits that relate to VBUS.

- **PRSC: Port 1 Reset Status Change (read/write, write ‘1’ to clear)**

A write of 1 clears this bit; a write of 0 has no effect.

0: Port 1 port reset status bit has not changed.

1: Port 1 port reset status bit has changed.

36.7.23 UHP HC Port 2 Status and Control Register

Name: UHP_HCRHPORTSTATUS2

Address: 0x4004C058

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	PRSC	OCIC	PSSC	PESC	CSC
15	14	13	12	11	10	9	8
–	–	–	–	–	–	LSDA/CPP	PPS/SPP
7	6	5	4	3	2	1	0
–	–	–	PRS/SPR	POCI/CSS	PSS/SPS	PES/SPE	CCS/CPE

This register reports and controls the state of USB1.1 host port 2.

- **CCS/CPE: Port 2 Current Connection Status/clear Port Enable (read/write)**

When read as 1, indicates that port 2 currently has a USB device attached. When 0, indicates that no USB device is attached to port 2. This bit is set to 1 after root hub reset if the UHP_HCRHDESCRIPTORB.DR[2] bit is set to indicate a non-removable device on port 2.

0: A write of 0 to this bit has no effect.

1: A write of 1 to this bit clears the port 2 port enable bit.

- **PES/SPE: Port 2 Current Connection Status/clear Port Enable (read/write)**

When read as 1, indicates that port 2 is enabled. When read as 0, this bit indicates that port 2 is not enabled. This bit is automatically set at completion of port 2 USB reset if it was not already set before the USB reset completed and is automatically set at the end of a USB suspend if the port was not enabled when the USB resume completed.

0: A write of 0 has no effect.

1: A write of 1 to this bit when port 2 current connect status is 1 sets the port 2 port enable status bit. A write of 1 when port 2 current connect status is 0 has no effect.

- **PSS/SPS: Port 2 Port Suspend Status/set Port Suspend (read/write)**

When read as 1, indicates that port 2 is in the USB suspend state, or is in the resume sequence. When 0, indicates that port 2 is not in the USB suspend state. This bit is cleared automatically at the end of the USB resume sequence and also at the end of the USB reset sequence.

0: A write of 0 to this bit has no effect.

1: If port 2 current connect status is 1, a write of 1 to this bit sets the port 2 port suspend status bit and places port 2 in USB suspend state. If current connect status is 0, a write of 1 instead sets connect status change to inform the USB1.1 host controller driver software of an attempt to suspend a disconnected device.

- **POCI/CSS: Port 2 Port Overcurrent Indicator/clear Suspend Status (read/write)**

When read as 1, indicates that a port 2 port overcurrent condition has occurred. When 0, no port 2 port overcurrent condition has occurred. The device does not provide inputs for signaling external overcurrent indication to the USB1.1 host controller. Overcurrent monitoring, if required, must be handled through some other mechanism. This bit has no relationship to the OTG controller register bits that relate to VBUS.

0: A write of 0 has no effect.

1: A write of 1 to this bit when port 2 port suspend status is 1 causes resume signaling on port 2. A write of 1 when port 2 port suspend status is 0 has no effect.

- **PRS/SPR: Port 2 Port Reset Status/set Port Reset (read/write)**

When read as 1, indicates that port 2 is sending a USB reset. When read as 0, USB reset is not being sent to port 2.

0: A write of 0 to this bit has no effect.

1: A write of 1 to this bit sets the port 2 port reset status bit and causes the USB1.1 host controller to begin signaling USB reset to port 2.

- **PPS/SPP: Port 2 Port Power Status/set Port Power (read/write)**

This bit indicates, when read as 1, that the port 2 power is enabled. When read as 0, port 2 power is not enabled. The device does not provide signals from the USB1.1 host controller to control external port power, so, if required, USB1.1 host port power control signals must be controlled through other means. Software can track the current power state using the port power status bit and other power control bits, but those bits have no direct effect on external port power control. This bit has no relationship to the OTG controller register bits that relate to VBUS. System software can update this register to simplify host controller driver and/or OTG driver coding.

0: A write of 0 has no effect.

1: A write of 1 to this bit sets the port 2 port power status bit.

- **LSDA/CPP: Port 2 Low-speed Device Attached/clear Port Power (read/write)**

This bit indicates, when read as 1, that a low-speed device is attached to port 2. A 0 in this bit indicates a full-speed device. This bit is valid only when port 2 current connect status is 1. The USB1.1 host controller does not control external port power using OHCI mechanisms, so, if required, USB1.1 host port power must be controlled through other means.

0: A write of 0 to this bit has no effect.

1: The host controller driver can write a 1 to this bit to clear the port 2 port power status.

- **CSC: Port 2 Connect Status Change (read/write)**

If the DR[2] bit in the UHP_HCRHDESCRIPTORB register is set to 1 to indicate a nonremovable USB device on port 2, this bit is set only after a root hub reset to inform the system that the device is attached. A write of 1 clears this bit; a write of 0 has no effect.

0: Port 2 current connect status has not changed.

1: Port 2 current connect status has changed due to a connect or disconnect event. If current connect status is 0 when a set port reset, set port enable, or set port suspend write occurs, then this bit is set.

- **PESC: Port 2 enable status change (read/write, write ‘1’ to clear)**

A write of 1 clears this bit; a write of 0 has no effect.

0: Port 2 port enable status has not changed.

1: Port 2 port enable status has changed.

- **PSSC: Port 2 suspend status changed (read/write, write ‘1’ to clear)**

A write of 1 clears this bit; a write of 0 has no effect.

0: Port 2 port suspend status has not changed.

1: Port 2 port suspend status has changed. Suspend status is considered to have changed only after the resume pulse, low-speed EOP, and 3-ms synchronization delays have been completed.

- **OCIC: Port 2 Overcurrent Indicator Change (read/write)**

0: Because the device does not provide inputs for signaling external overcurrent indication to the USB1.1 host controller, this bit is always 0. Overcurrent monitoring, if required, must be handled through some other mechanism. This bit has no relationship to the OTG controller register bits that relate to VBUS.

- **PRSC: Port 2 Reset Status Change (read/write, write ‘1’ to clear)**

A write of 1 clears this bit; a write of 0 has no effect.

0: Port 2 port reset status bit has not changed.

1: Port 2 port reset status bit has changed.

37. USB Device Port (UDP)

37.1 Description

The USB Device Port (UDP) is compliant with the Universal Serial Bus (USB) 2.0 full-speed device specification.

Each endpoint can be configured in one of several USB transfer types. It can be associated with one or two banks of a dual-port RAM used to store the current data payload. If two banks are used, one DPR bank is read or written by the processor, while the other is read or written by the USB device peripheral. This feature is mandatory for isochronous endpoints. Thus the device maintains the maximum bandwidth (1 Mbyte/s) by working with endpoints with two banks of DPR.

Table 37-1. USB Endpoint Description

Endpoint No.	Mnemonic	Dual-Bank(1)	Max. Endpoint Size	Endpoint Type
0	EP0	No	64	Control/Bulk/Interrupt
1	EP1	Yes	128	Bulk/Iso/Interrupt
2	EP2	Yes	128	Bulk/Iso/Interrupt
3	EP3	No	64	Control/Bulk/Interrupt
4	EP4	Yes	512	Bulk/Iso/Interrupt
5	EP5	Yes	512	Bulk/Iso/Interrupt

Note: 1. The Dual-Bank function provides two banks for an endpoint. This feature is used for ping-pong mode.

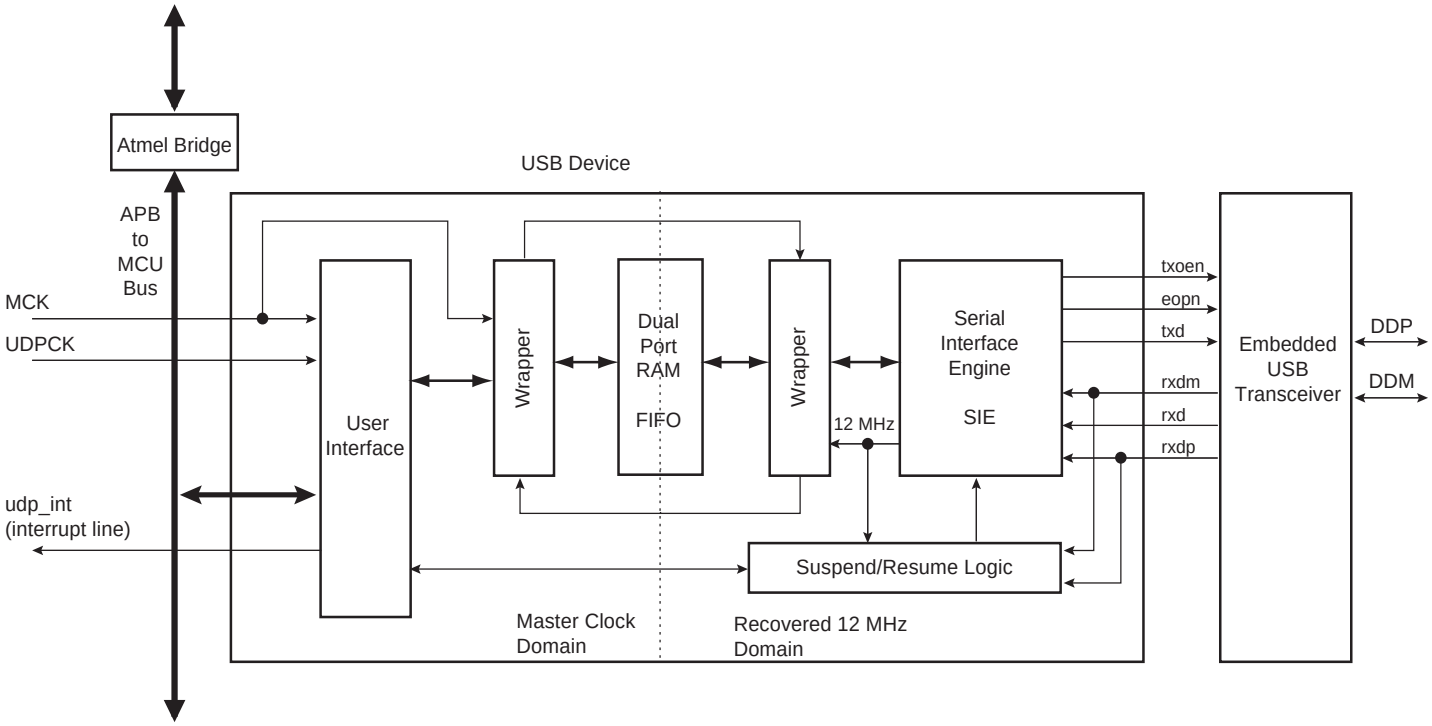
Suspend and resume are automatically detected by the USB device, which notifies the processor by raising an interrupt. Depending on the product, an external signal can be used to send a wakeup request to the USB host controller.

37.2 Embedded Characteristics

- USB 2.0 Full-speed Compliant, 12 Mbit/s
- Embedded USB 2.0 Full-speed Transceiver
- Integrated Pull-up on DDP
- Integrated Pull-down on DDM
- 6 Endpoints
- Embedded Dual-port RAM for Endpoints
- Suspend/Resume Logic
- Ping-pong Mode (2 Memory Banks) for Isochronous and Bulk Endpoints

37.3 Block Diagram

Figure 37-1. Block Diagram



Access to the UDP is via the APB bus interface. Read and write to the data FIFO are done by reading and writing 8-bit values to APB registers.

The UDP peripheral requires two clocks: one peripheral clock used by the Master Clock domain (MCK) and a 48 MHz clock (UDPCK) used by the 12 MHz domain.

A USB 2.0 full-speed pad is embedded and controlled by the Serial Interface Engine (SIE).

The signal external_resume is optional. It allows the UDP peripheral to wake up once in system mode. The host is then notified that the device asks for a resume. This optional feature must also be negotiated with the host during the enumeration.

37.3.1 Signal Description

Table 37-2. Signal Names

Signal Name	Description	Type
UDPCK	48 MHz clock	Input
MCK	Master clock	Input
udp_int	Interrupt line connected to the Interrupt Controller	Input
DDP	USB D+ line	I/O
DDM	USB D- line	I/O

37.4 Product Dependencies

For further details on the USB Device hardware implementation, see the specific Product Properties document.

The USB physical transceiver is integrated into the product. The bidirectional differential signals DDP and DDM are available from the product boundary.

One I/O line may be used by the application to check that VBUS is still available from the host. Self-powered devices may use this entry to be notified that the host has been powered off. In this case, the pull-up on DDP must be disabled in order to prevent feeding current to the host. The application should disconnect the transceiver, then remove the pull-up.

37.4.1 I/O Lines

The USB pins are shared with PIO lines. By default, the DDP/DDM pins are configured as PIOs.

37.4.2 Power Management

The USB device peripheral requires a 48 MHz clock. This clock must be generated by a PLL driven by a clock source with an accuracy of ± 0.25% (note that the fast RC oscillator cannot be used).

Thus, the USB device receives two clocks from the Power Management Controller (PMC):

- Master clock, MCK, used to drive the peripheral user interface
- UDPCK, used to interface with the bus USB signals (recovered 12 MHz domain)

WARNING: The UDP peripheral clock in the PMC must be enabled before any read/write operations to the UDP registers including the Transceiver Control Register (UDP_TXVC).

37.4.3 Interrupt Sources

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

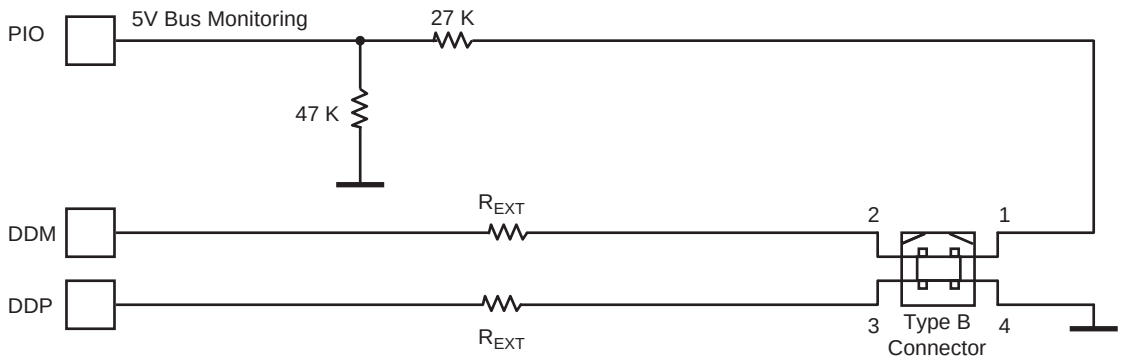
Handling the USB device interrupt requires programming the Interrupt Controller before configuring the UDP.

Table 37-3. Peripheral IDs

Instance	ID
UDP	48

37.5 Typical Connection

Figure 37-2. Board Schematic to Interface Device Peripheral



37.5.1 USB Device Transceiver

The USB device transceiver is embedded in the product. However, discrete components are required for each of the following actions:

- to monitor VBUS voltage
- for line termination
- to disconnect the host for reduced power consumption

37.5.2 VBUS Monitoring

VBUS monitoring is required to detect host connection. VBUS monitoring is done using a standard PIO with internal pull-up disabled. When the host is switched off, it should be considered as a disconnect, the pull-up must be disabled in order to prevent powering the host through the pull-up resistor.

When the host is disconnected and the transceiver is enabled, then DDP and DDM are floating. This may lead to over consumption. A solution is to enable the integrated pull-down by disabling the transceiver (UDP_TXVC.TXVDIS = 1) and then remove the pull-up (UDP_TXVC.PUON = 0).

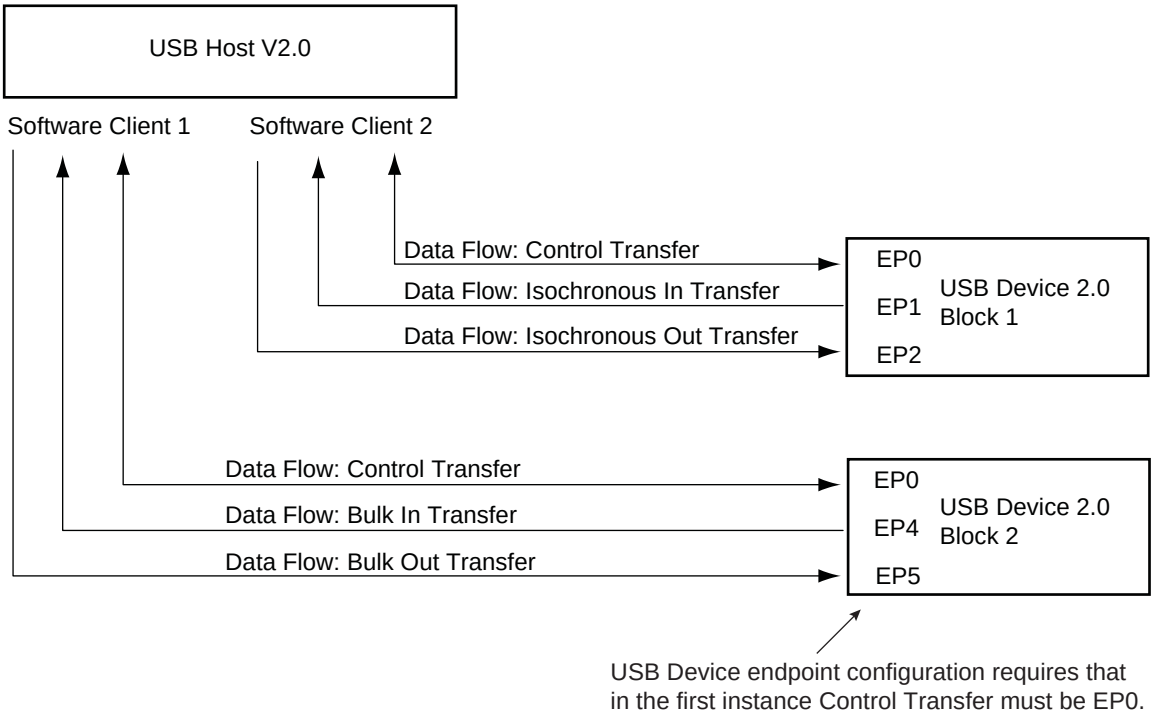
A termination serial resistor must be connected to DDP and DDM. The resistor value is defined in the electrical characteristics of the product (R_{EXT}).

37.6 Functional Description

37.6.1 USB 2.0 Full-speed Introduction

The USB 2.0 full-speed provides communication services between host and attached USB devices. Each device is offered with a collection of communication flows (pipes) associated with each endpoint. Software on the host communicates with a USB device through a set of communication flows.

Figure 37-3. Example of USB 2.0 Full-speed Communication Control



The Control Transfer endpoint EP0 is always used when a USB device is first configured (USB 2.0 specifications).

37.6.1.1 USB 2.0 Full-speed Transfer Types

A communication flow is carried over one of four transfer types defined by the USB device.

Table 37-4. USB Communication Flow

Transfer	Direction	Bandwidth	Supported Endpoint Size	Error Detection	Retrying
Control	Bidirectional	Not guaranteed	8, 16, 32, 64	Yes	Automatic
Isochronous	Unidirectional	Guaranteed	512	Yes	No
Interrupt	Unidirectional	Not guaranteed	≤ 64	Yes	Yes
Bulk	Unidirectional	Not guaranteed	8, 16, 32, 64	Yes	Yes

37.6.1.2 USB Bus Transactions

Each transfer results in one or more transactions over the USB bus. There are three kinds of transactions flowing across the bus in packets:

- Setup Transaction
- Data IN Transaction
- Data OUT Transaction

37.6.1.3 USB Transfer Event Definitions

As indicated below, transfers are sequential events carried out on the USB bus.

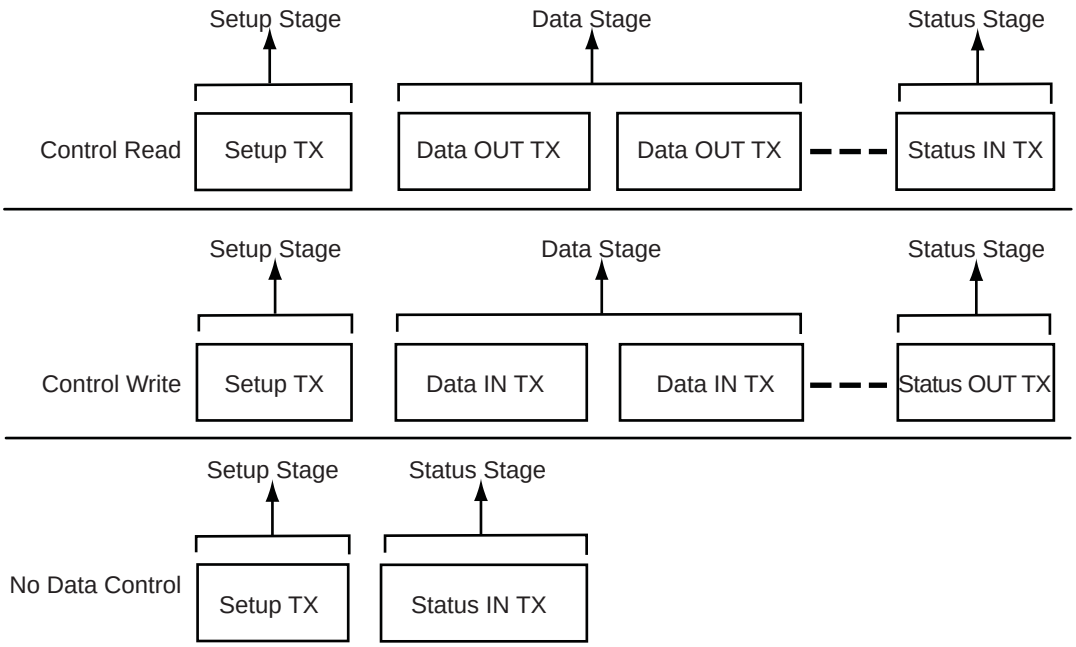
Table 37-5. USB Transfer Events

Transfer		Transaction
Direction	**Type**	
CONTROL (bidirectional)	Control[1][3]	Setup transaction → Data IN transactions → Status OUT transaction
		Setup transaction → Data OUT transactions → Status IN transaction
		Setup transaction → Status IN transaction
IN (device toward host)	Interrupt IN	Data IN transaction → Data IN transaction
	Isochronous IN[2]	
	Bulk IN	
OUT (host toward device)	Interrupt OUT	Data OUT transaction → Data OUT transaction
	Isochronous OUT[2]	
	Bulk OUT	

Notes:
1. Control transfer must use endpoints with no ping-pong attributes.
2. Isochronous transfers must use endpoints with ping-pong attributes.
3. Control transfers can be aborted using a stall handshake.

A status transaction is a special type of host-to-device transaction used only in a control transfer. The control transfer must be performed using endpoints with no ping-pong attributes. According to the control sequence (read or write), the USB device sends or receives a status transaction.

Figure 37-4. Control Read and Write Sequences



Notes:
1. During the Status IN stage, the host waits for a zero length packet (Data IN transaction with no data) from the device using DATA1 PID. Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev. 2.0,* for more information on the protocol layer.
2. During the Status OUT stage, the host emits a zero length packet to the device (Data OUT transaction with no data).

37.6.2 Handling Transactions with USB 2.0 Device Peripheral

37.6.2.1 Setup Transaction

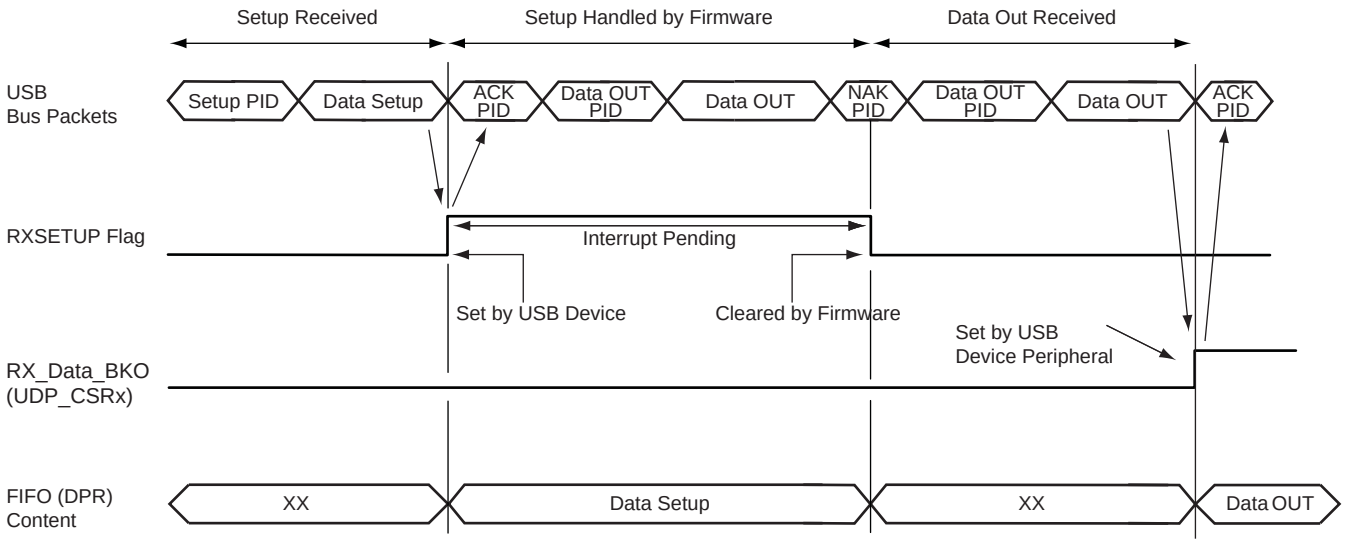
Setup is a special type of host-to-device transaction used during control transfers. Control transfers must be performed using endpoints with no ping-pong attributes. A setup transaction needs to be handled as soon as possible by the firmware. It is used to transmit requests from the host to the device. These requests are then handled by the USB device and may require more arguments. The arguments are sent to the device by a Data OUT transaction which follows the setup transaction. These requests may also return data. The data is carried out to the host by the next Data IN transaction which follows the setup transaction. A status transaction ends the control transfer.

When a setup transfer is received by the USB endpoint:

- The USB device automatically acknowledges the setup packet
- RXSETUP is set in the corresponding Endpoint Control and Status Register x (UDP_CSRx)
- An endpoint interrupt is generated while the RXSETUP is not cleared. This interrupt is carried out to the microcontroller if interrupts are enabled for this endpoint.

Thus, firmware must detect the RXSETUP polling the UDP_CSRx or catching an interrupt, read the setup packet in the FIFO, then clear the RXSETUP. RXSETUP cannot be cleared before the setup packet has been read in the FIFO. Otherwise, the USB device would accept the next Data OUT transfer and overwrite the setup packet in the FIFO.

Figure 37-5. Setup Transaction Followed by a Data OUT Transaction



37.6.2.2 Data IN Transaction

Data IN transactions are used in control, isochronous, bulk and interrupt transfers and conduct the transfer of data from the device to the host. Data IN transactions in isochronous transfer must be done using endpoints with ping-pong attributes.

Using Endpoints Without Ping-pong Attributes

To perform a Data IN transaction using a non ping-pong endpoint:

1. The application checks if it is possible to write in the FIFO by polling TXPKTRDY in the endpoint's UDP_CSRx (TXPKTRDY must be cleared).
2. The application writes the first packet of data to be sent in the endpoint's FIFO, writing zero or more byte values in the endpoint's FIFO Data Register x (UDP_FDRx).

3. The application notifies the USB peripheral it has finished by setting the TXPKTRDY in the endpoint's UDP_CSRx.
4. The application is notified that the endpoint's FIFO has been released by the USB device when TXCOMP in the endpoint's UDP_CSRx has been set. Then an interrupt for the corresponding endpoint is pending while TXCOMP is set.
5. The microcontroller writes the second packet of data to be sent in the endpoint's FIFO, writing zero or more byte values in the endpoint's UDP_FDRx.
6. The microcontroller notifies the USB peripheral it has finished by setting the TXPKTRDY in the endpoint's UDP_CSRx.
7. The application clears the TXCOMP in the endpoint's UDP_CSRx.

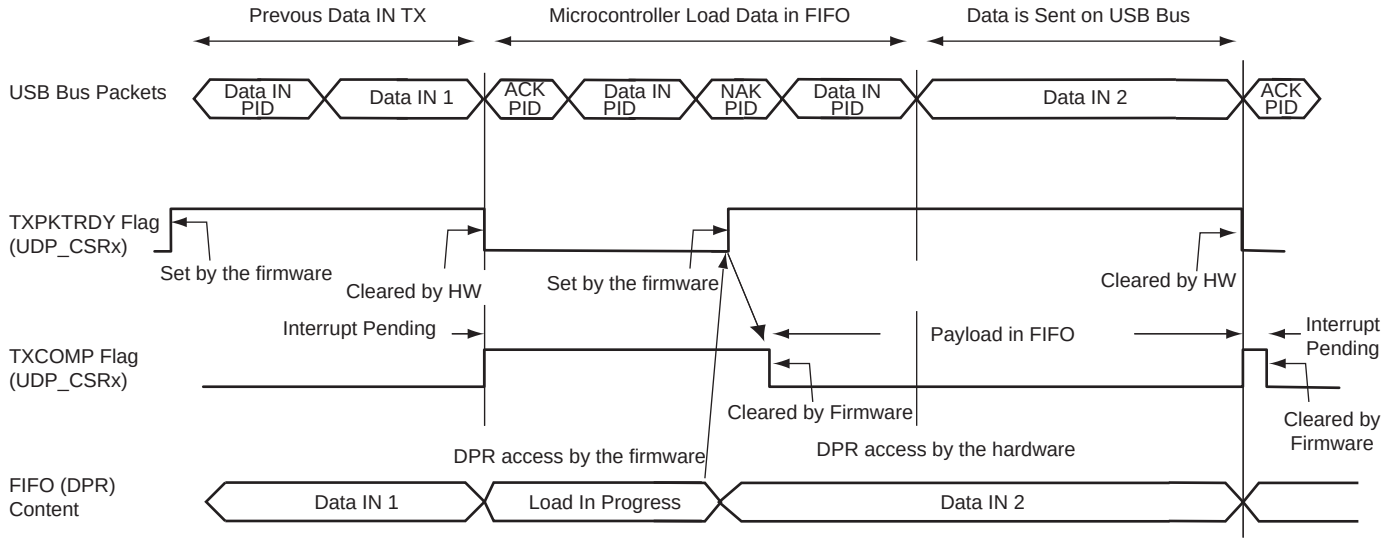
After the last packet has been sent, the application must clear TXCOMP once this has been set.

TXCOMP is set by the USB device when it has received an ACK PID signal for the Data IN packet. An interrupt is pending while TXCOMP is set.

Warning: TX_COMP must be cleared after TX_PKTRDY has been set.

Note: Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev 2.0,* for more information on the Data IN protocol layer.

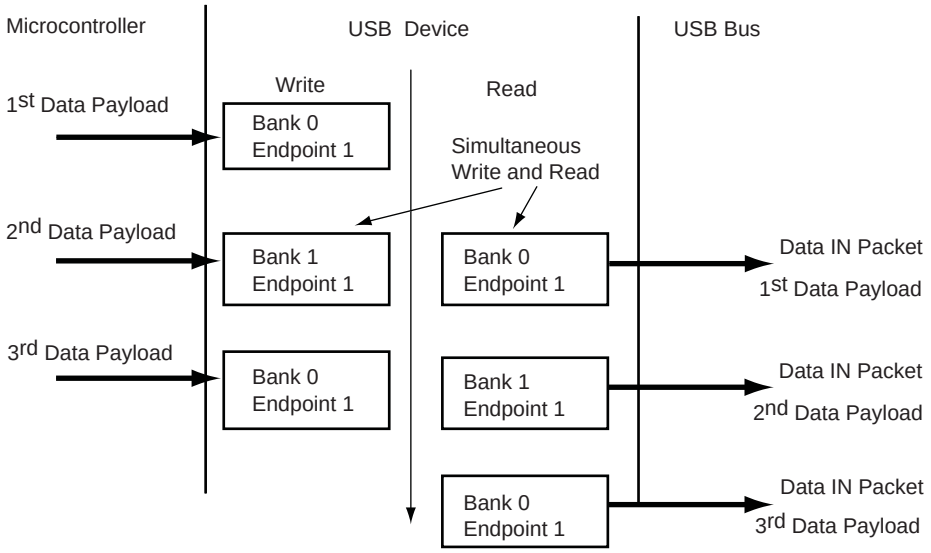
Figure 37-6. Data IN Transfer for Non Ping-pong Endpoint



Using Endpoints With Ping-pong Attribute

The use of an endpoint with ping-pong attributes is necessary during isochronous transfer. This also allows handling the maximum bandwidth defined in the USB specification during bulk transfer. To be able to guarantee a constant or the maximum bandwidth, the microcontroller must prepare the next data payload to be sent while the current one is being sent by the USB device. Thus two banks of memory are used. While one is available for the microcontroller, the other one is locked by the USB device.

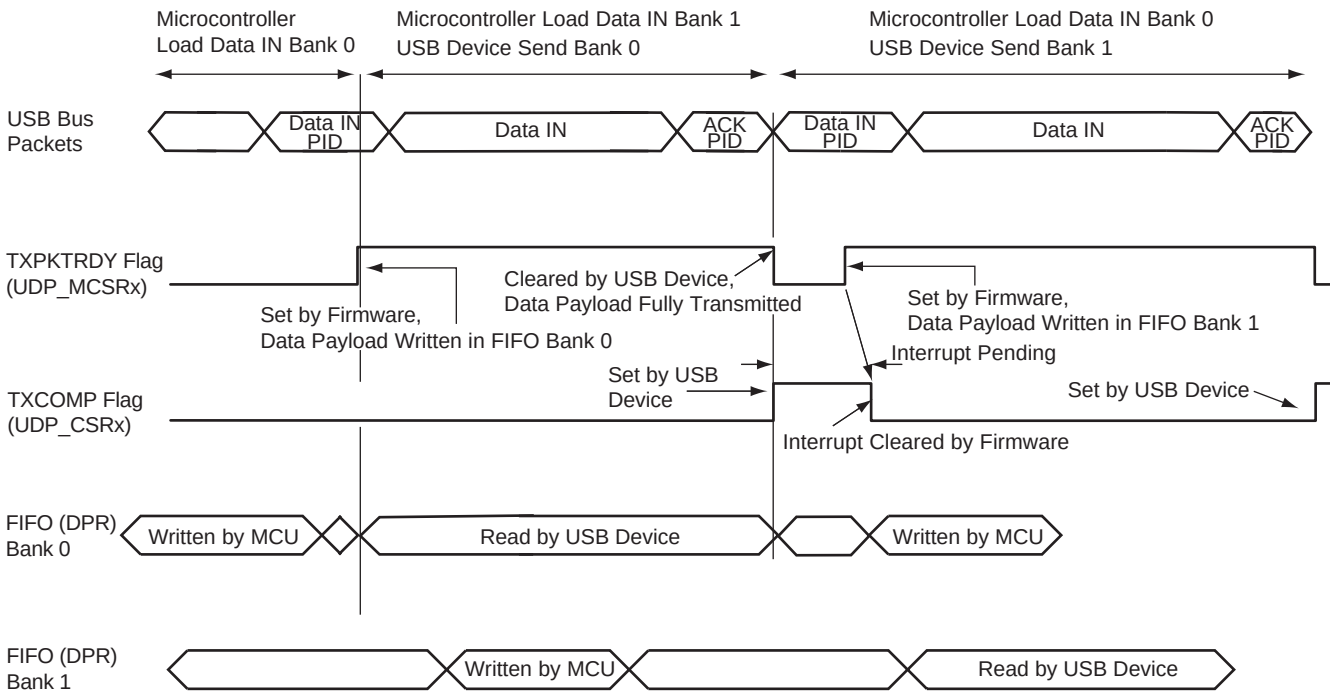
Figure 37-7. Bank Swapping Data IN Transfer for Ping-pong Endpoints



When using a ping-pong endpoint, the following procedures are required to perform Data IN transactions:

1. The microcontroller checks if it is possible to write in the FIFO by polling TXPKTRDY to be cleared in the endpoint's UDP_CSRx.
2. The microcontroller writes the first data payload to be sent in the FIFO (Bank 0), writing zero or more byte values in the endpoint's UDP_FDRx.
3. The microcontroller notifies the USB peripheral it has finished writing in Bank 0 of the FIFO by setting the TXPKTRDY in the endpoint's UDP_CSRx.
4. Without waiting for TXPKTRDY to be cleared, the microcontroller writes the second data payload to be sent in the FIFO (Bank 1), writing zero or more byte values in the endpoint's UDP_FDRx.
5. The microcontroller is notified that the first Bank has been released by the USB device when TXCOMP in the endpoint's UDP_CSRx is set. An interrupt is pending while TXCOMP is being set.
6. Once the microcontroller has received TXCOMP for the first Bank, it notifies the USB device that it has prepared the second Bank to be sent, raising TXPKTRDY in the endpoint's UDP_CSRx.
7. At this step, Bank 0 is available and the microcontroller can prepare a third data payload to be sent.

Figure 37-8. Data IN Transfer for Ping-pong Endpoint



Warning: There is software critical path due to the fact that once the second bank is filled, the driver has to wait for TX_COMP to set TX_PKTRDY. If the delay between receiving TX_COMP is set and TX_PKTRDY is set too long, some Data IN packets may be NACKed, reducing the bandwidth.

Warning: TX_COMP must be cleared after TX_PKTRDY has been set.

37.6.2.3 Data OUT Transaction

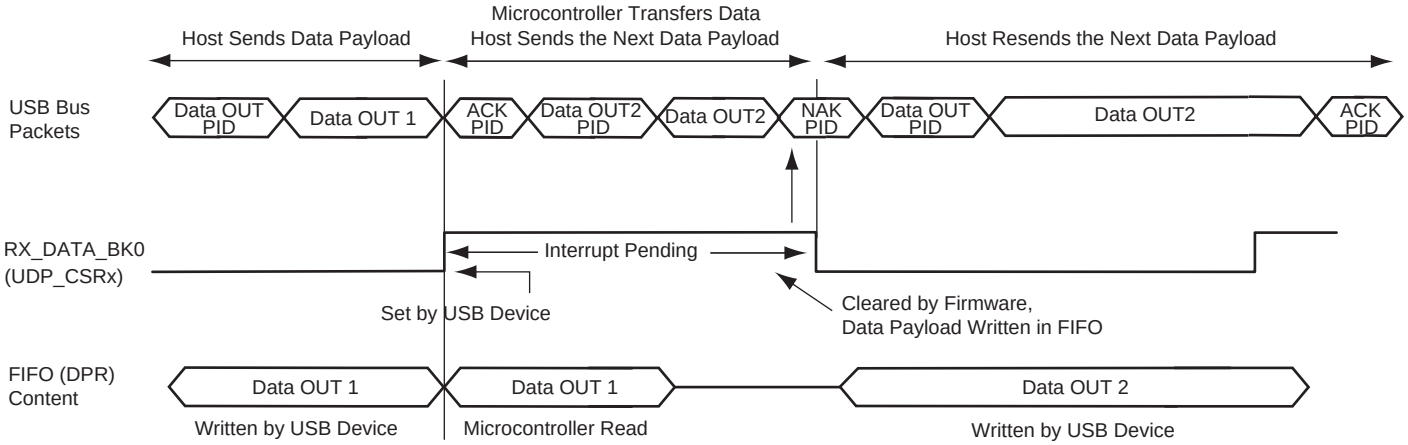
Data OUT transactions are used in control, isochronous, bulk and interrupt transfers and conduct the transfer of data from the host to the device. Data OUT transactions in isochronous transfers must be done using endpoints with ping-pong attributes.

Data OUT Transaction Without Ping-pong Attributes

To perform a Data OUT transaction, using a non ping-pong endpoint:

1. The host generates a Data OUT packet.
2. This packet is received by the USB device endpoint. While the FIFO associated to this endpoint is being used by the microcontroller, a NAK PID is returned to the host. Once the FIFO is available, data are written to the FIFO by the USB device and an ACK is automatically carried out to the host.
3. The microcontroller is notified that the USB device has received a data payload polling RX_DATA_BK0 in the endpoint's UDP_CSRx. An interrupt is pending for this endpoint while RX_DATA_BK0 is set.
4. The number of bytes available in the FIFO is made available by reading RXBYTECNT in the endpoint's UDP_CSRx.
5. The microcontroller carries out data received from the endpoint's memory to its memory. Data received is available by reading the endpoint's UDP_FDRx.
6. The microcontroller notifies the USB device that it has finished the transfer by clearing RX_DATA_BK0 in the endpoint's UDP_CSRx.
7. A new Data OUT packet can be accepted by the USB device.

Figure 37-9. Data OUT Transfer for Non Ping-pong Endpoints

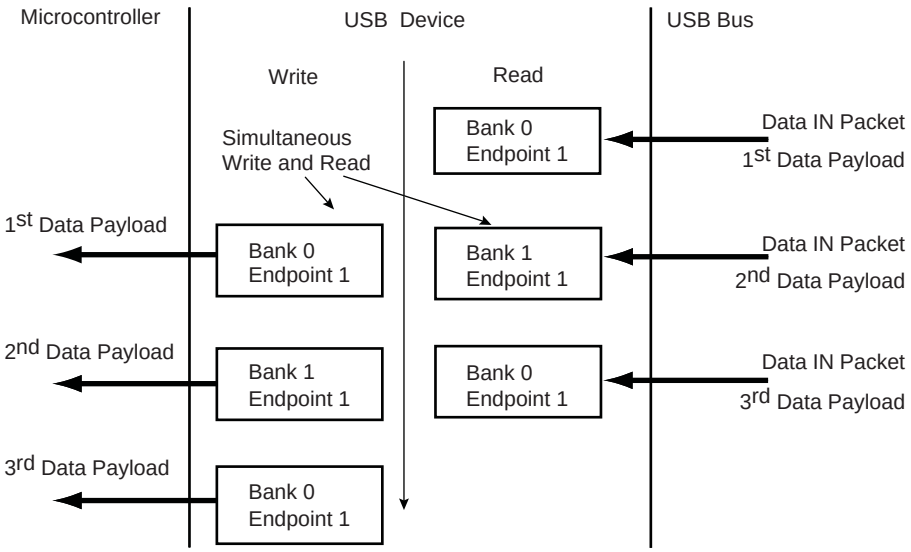


An interrupt is pending while the flag RX_DATA_BK0 is set. Memory transfer between the USB device, the FIFO and microcontroller memory is not possible after RX_DATA_BK0 has been cleared. Otherwise, the USB device would accept the next Data OUT transfer and overwrite the current Data OUT packet in the FIFO.

Using Endpoints With Ping-pong Attributes

During isochronous transfer, using an endpoint with ping-pong attributes is obligatory. To be able to guarantee a constant bandwidth, the microcontroller must read the previous data payload sent by the host, while the current data payload is received by the USB device. Thus two banks of memory are used. While one is available for the microcontroller, the other one is locked by the USB device.

Figure 37-10. Bank Swapping in Data OUT Transfers for Ping-pong Endpoints

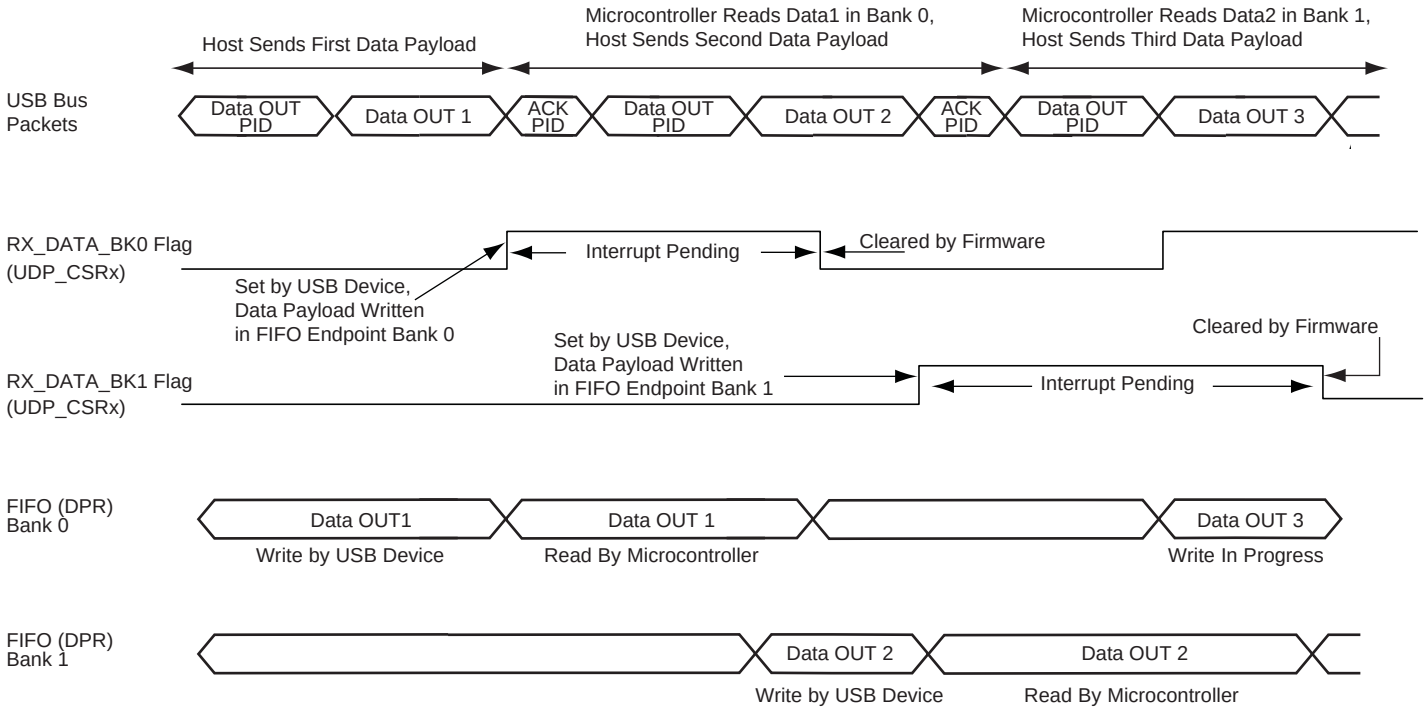


When using a ping-pong endpoint, the following procedures are required to perform Data OUT transactions:

1. The host generates a Data OUT packet.
2. This packet is received by the USB device endpoint. It is written in the endpoint's FIFO Bank 0.
3. The USB device sends an ACK PID packet to the host. The host can immediately send a second Data OUT packet. It is accepted by the device and copied to FIFO Bank 1.
4. The microcontroller is notified that the USB device has received a data payload, polling RX_DATA_BK0 in the endpoint's UDP_CSRx. An interrupt is pending for this endpoint while RX_DATA_BK0 is set.

5. The number of bytes available in the FIFO is made available by reading RXBYTECNT in the endpoint's UDP_CSRx.
6. The microcontroller transfers out data received from the endpoint's memory to the microcontroller's memory. Data received is made available by reading the endpoint's UDP_FDRx.
7. The microcontroller notifies the USB peripheral device that it has finished the transfer by clearing RX_DATA_BK0 in the endpoint's UDP_CSRx.
8. A third Data OUT packet can be accepted by the USB peripheral device and copied in the FIFO Bank 0.
9. If a second Data OUT packet has been received, the microcontroller is notified by the flag RX_DATA_BK1 set in the endpoint's UDP_CSRx. An interrupt is pending for this endpoint while RX_DATA_BK1 is set.
10. The microcontroller transfers out data received from the endpoint's memory to the microcontroller's memory. Data received is available by reading the endpoint's UDP_FDRx.
11. The microcontroller notifies the USB device it has finished the transfer by clearing RX_DATA_BK1 in the endpoint's UDP_CSRx.
12. A fourth Data OUT packet can be accepted by the USB device and copied in the FIFO Bank 1.

Figure 37-11. Data OUT Transfer for Ping-pong Endpoint



Note: An interrupt is pending while the RX_DATA_BK0 or RX_DATA_BK1 flag is set.

Warning: When RX_DATA_BK0 and RX_DATA_BK1 are both set, there is no way to determine which one to clear first. Thus the software must keep an internal counter to be sure to clear alternatively RX_DATA_BK0 then RX_DATA_BK1. This situation may occur when the software application is busy elsewhere and the two banks are filled by the USB host. Once the application comes back to the USB driver, the two flags are set.

37.6.2.4 Stall Handshake

A stall handshake can be used in one of two distinct occasions. (For more information on the stall handshake, refer to Chapter 8 of the *Universal Serial Bus Specification, Rev 2.0.*)

- A functional stall is used when the halt feature associated with the endpoint is set. (Refer to Chapter 9 of the *Universal Serial Bus Specification, Rev 2.0,* for more information on the halt feature.)
- To abort the current request, a protocol stall is used, but uniquely with control transfer.

The following procedure generates a stall packet:

1. The microcontroller sets the FORCESTALL flag in the UDP_CSRx endpoint's register.
2. The host receives the stall packet.
3. The microcontroller is notified that the device has sent the stall by polling the STALLSENT to be set. An endpoint interrupt is pending while STALLSENT is set. The microcontroller must clear STALLSENT to clear the interrupt.

When a setup transaction is received after a stall handshake, STALLSENT must be cleared in order to prevent interrupts due to STALLSENT being set.

Figure 37-12. Stall Handshake (Data IN Transfer)

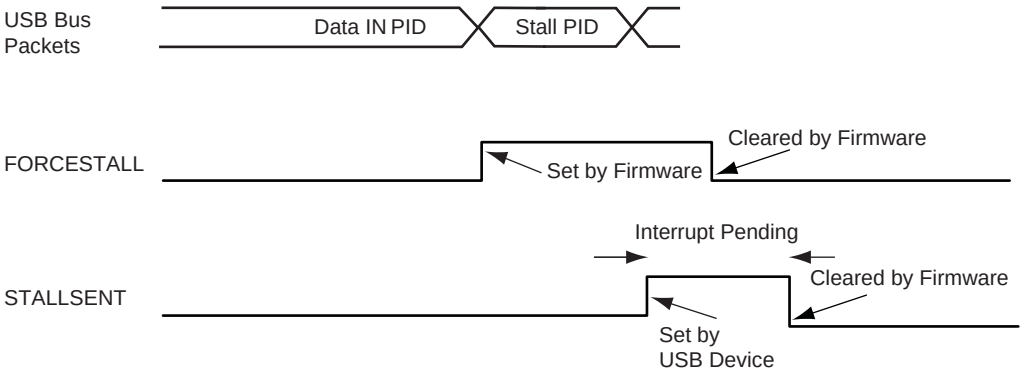
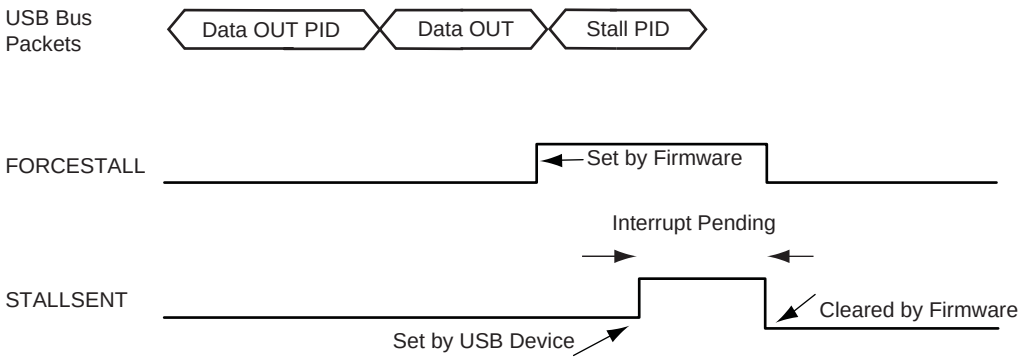


Figure 37-13. Stall Handshake (Data OUT Transfer)



37.6.2.5 Transmit Data Cancellation

Some endpoints have dual-banks whereas some endpoints have only one bank. The procedure to cancel transmission data held in these banks is described below.

To see the organization of dual-bank availability refer to Table 37-1 ”USB Endpoint Description”.

*Endpoints **Without** Dual-Banks*

The cancellation procedure depends on the TXPKTRDY flag value in the UDP_CSR:

- TXPKTRDY is not set:
  - Reset the endpoint to clear the FIFO (pointers). (See Section 37.7.9 ”UDP Reset Endpoint Register”.)
- TXPKTRDY has already been set:
  - Clear TXPKTRDY so that no packet is ready to be sent
  - Reset the endpoint to clear the FIFO (pointers). (See Section 37.7.9 ”UDP Reset Endpoint Register”.)

*Endpoints **With** Dual-Banks*

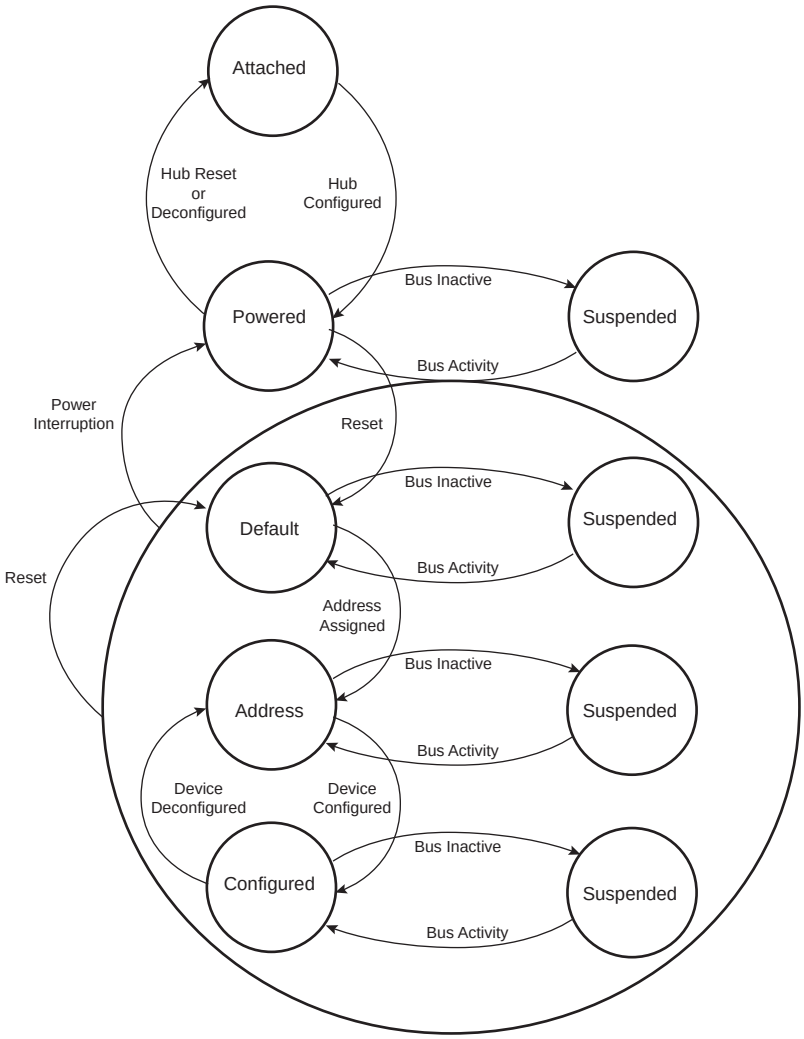
The cancellation procedure depends on the TXPKTRDY flag value in the UDP_CSR:

- TXPKTRDY is not set:
  - Reset the endpoint to clear the FIFO (pointers). (See Section 37.7.9 ”UDP Reset Endpoint Register”.)
- TXPKTRDY has already been set:
  - Clear TXPKTRDY and read it back until actually read at 0.
  - Set TXPKTRDY and read it back until actually read at 1.
  - Clear TXPKTRDY so that no packet is ready to be sent.
  - Reset the endpoint to clear the FIFO (pointers). (See Section 37.7.9 ”UDP Reset Endpoint Register”.)

37.6.3 Controlling Device States

A USB device has several possible states. Refer to Chapter 9 of the *Universal Serial Bus Specification, Rev 2.0*.

Figure 37-14. USB Device State Diagram



Movement from one state to another depends on the USB bus state or on standard requests sent through control transactions via the default endpoint (endpoint 0).

After a period of bus inactivity, the USB device enters Suspend Mode. Accepting Suspend/Resume requests from the USB host is mandatory. Constraints in Suspend Mode are very strict for bus-powered applications; devices must not consume more than 2.5 mA on the USB bus.

While in Suspend Mode, the host may wake up a device by sending a resume signal (bus activity) or a USB device may send a wakeup request to the host, e.g., waking up a PC by moving a USB mouse.

The wakeup feature is not mandatory for all devices and must be negotiated with the host.

37.6.3.1 Not Powered State

Self powered devices can detect 5V VBUS using a PIO as described in the typical connection section. When the device is not connected to a host, device power consumption can be reduced by disabling MCK for the UDP, disabling UDPCK and disabling the transceiver. DDP and DDM lines are pulled down by 330 KΩ resistors.

37.6.3.2 Entering Attached State

To enable integrated pull-up, the PUON bit in the UDP_TXVC register must be set.

Warning: To write to the UDP_TXVC register, MCK clock must be enabled on the UDP. This is done in the PMC.

After pull-up connection, the device enters the powered state. In this state, the UDPCK and MCK must be enabled in the PMC. The transceiver can remain disabled.

37.6.3.3 From Powered State to Default State

After its connection to a USB host, the USB device waits for an end-of-bus reset. The unmaskable flag ENDBUSRES is set in the Interrupt Status Register (UDP_ISR) and an interrupt is triggered.

Once the ENDBUSRES interrupt has been triggered, the device enters Default State. In this state, the UDP software must:

- Enable the default endpoint, setting the EPEDS flag in the UDP_CSR0 and, optionally, enabling the interrupt for endpoint 0 by writing 1 to the Interrupt Enable Register (UDP_IER). The enumeration then begins by a control transfer.
- Configure the Interrupt Mask Register (UDP_IMR) which has been reset by the USB reset detection
- Enable the transceiver clearing the TXVDIS flag in the UDP_TXVC register.

In this state UDPCK and MCK must be enabled.

Warning: Each time an ENDBUSRES interrupt is triggered, the UDP_IMR and UDP_CSRs have been reset.

37.6.3.4 From Default State to Address State

After a set address standard device request, the USB host peripheral enters the address state.

Warning: Before the device enters in address state, it must achieve the Status IN transaction of the control transfer, i.e., the UDP device sets its new address once the TXCOMP flag in the UDP_CSR0 has been received and cleared.

To move to address state, the driver software sets the FADDEN flag in the Global State Register (UDP_GLB_STAT), sets its new address, and sets the FEN bit in the Function Address Register (UDP_FADDR).

37.6.3.5 From Address State to Configured State

Once a valid Set Configuration standard request has been received and acknowledged, the device enables endpoints corresponding to the current configuration. This is done by setting the EPEDS and EPTYPE fields in the UDP_CSRx and, optionally, enabling corresponding interrupts in the UDP_IER.

37.6.3.6 Entering in Suspend State

When a Suspend (no bus activity on the USB bus) is detected, the RXSUSP signal in the UDP_ISR is set. This triggers an interrupt if the corresponding bit is set in the UDP_IMR. This flag is cleared by writing to the Interrupt Clear Register (UDP_ICR) and the device then enters Suspend mode.

In this state bus powered devices must drain no more than 2.5 mA from the 5V VBUS. As an example, the microcontroller switches to slow clock, disables the PLL and main oscillator, and goes into Idle mode. It may also switch off other devices on the board.

The USB device peripheral clocks can be switched off. Resume event is asynchronously detected. MCK and UDPCK can be switched off in the PMC and the USB transceiver can be disabled by setting the TXVDIS bit in the UDP_TXVC register.

Warning: Read, write operations to the UDP registers are allowed only if MCK is enabled for the UDP peripheral. Switching off MCK for the UDP peripheral must be one of the last operations after writing to the UDP_TXVC register and acknowledging the RXSUSP.

37.6.3.7 Receiving a Host Resume

In suspend mode, a resume event on the USB bus line is detected asynchronously, transceiver and clocks are disabled (however the pull-up shall not be removed).

Once the resume is detected on the bus, the WAKEUP signal in the UDP_ISR is set. It may generate an interrupt if the corresponding bit in the UDP_IMR is set. This interrupt may be used to wake up the core, enable PLL and main oscillators and configure clocks.

Warning: Read, write operations to the UDP registers are allowed only if MCK is enabled for the UDP peripheral. MCK for the UDP must be enabled before clearing the WAKEUP bit in the UDP_ICR and clearing TXVDIS in the UDP_TXVC register.

37.6.3.8 Sending a Device Remote Wakeup Request

In Suspend state it is possible to wake up the host sending an external resume.

- The device must wait at least 5 ms after being entered in suspend before sending an external resume.
- The device has 10 ms from the moment it starts to drain current and it forces a K state to resume the host.
- The device must force a K state from 1 to 15 ms to resume the host

Before sending a K state to the host, MCK, UDPCK and the transceiver must be enabled. Then to enable the remote wakeup feature, the RMWUPE bit in the UDP_GLB_STAT register must be enabled. To force the K state on the line, a transition of the ESR bit from 0 to 1 has to be done in the UDP_GLB_STAT register by first writing a 0 in the ESR bit and then writing a 1.

The K state is automatically generated and released according to the USB 2.0 specification.

37.7 USB Device Port (UDP) User Interface

WARNING: The UDP peripheral clock in the PMC must be enabled before any read/write operations to the UDP registers, including the UDP_TXVC register.

Table 37-6. Register Mapping

Offset	Register	Name	Access	Reset
0x000	Frame Number Register	UDP_FRM_NUM	Read-only	0x0000_0000
0x004	Global State Register	UDP_GLB_STAT	Read/Write	0x0000_0010
0x008	Function Address Register	UDP_FADDR	Read/Write	0x0000_0100
0x00C	Reserved	–	–	–
0x010	Interrupt Enable Register	UDP_IER	Write-only	
0x014	Interrupt Disable Register	UDP_IDR	Write-only	
0x018	Interrupt Mask Register	UDP_IMR	Read-only	0x0000_1200
0x01C	Interrupt Status Register	UDP_ISR	Read-only	–(1)
0x020	Interrupt Clear Register	UDP_ICR	Write-only	
0x024	Reserved	–	–	–
0x028	Reset Endpoint Register	UDP_RST_EP	Read/Write	0x0000_0000
0x02C	Reserved	–	–	–
0x030	Endpoint Control and Status Register 0	UDP_CSR0	Read/Write	0x0000_0000
...	...	...	...	...
0x030 + 0x4 * 5	Endpoint Control and Status Register 5	UDP_CSR5	Read/Write	0x0000_0000
0x050	Endpoint FIFO Data Register 0	UDP_FDR0	Read/Write	–(1)
...	...	...	...	...
0x050 + 0x4 * 5	Endpoint FIFO Data Register 5	UDP_FDR5	Read/Write	–(1)
0x070	Reserved	–	–	–
0x074	Transceiver Control Register	UDP_TXVC(2)	Read/Write	0x0000_0100
0x078–0x0FC	Reserved	–	–	–

Notes: 1. Reset values are not defined for UDP_ISR or UDP_FDRx. UDP_FDRs reflect Dual Port RAM memory locations which are not affected by any reset signals.
2. See Warning above Table 37-6 ”Register Mapping”.

37.7.1 UDP Frame Number Register

Name: UDP_FRM_NUM

Address: 0x40044000

Access: Read-only

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

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

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

7	6	5	4	3	2	1	0
FRM_NUM							

- **FRM_NUM: Frame Number as Defined in the Packet Field Formats**

This 11-bit value is incremented by the host on a per frame basis. This value is updated at each start of frame.

Value updated at the SOF_EOP (Start of Frame End of Packet).

- **FRM_ERR: Frame Error**

This bit is set at SOF_EOP when the SOF packet is received containing an error.

This bit is reset upon receipt of SOF_PID.

- **FRM_OK: Frame OK**

This bit is set at SOF_EOP when the SOF packet is received without any error.

This bit is reset upon receipt of SOF_PID (Packet Identification).

In the Interrupt Status Register, the SOF interrupt is updated upon receiving SOF_PID. This bit is set without waiting for EOP.

Note: In the 8-bit Register Interface, FRM_OK is bit 4 of FRM_NUM_H and FRM_ERR is bit 3 of FRM_NUM_L.

37.7.2 UDP Global State Register

Name: UDP_GLB_STAT

Address: 0x40044004

Access: Read/Write

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

This register is used to get and set the device state as specified in Chapter 9 of the *USB Serial Bus Specification, Rev.2.0*.

- **FADDEN: Function Address Enable**

Read:

0: Device is not in address state

1: Device is in address state

Write:

0: No effect, only a reset can bring back a device to the default state.

1: Sets device in address state. This occurs after a successful Set Address request. Beforehand, the UDP_FADDR must have been initialized with Set Address parameters. Set Address must complete the Status Stage before setting FADDEN. Refer to chapter 9 of the *Universal Serial Bus Specification, Rev. 2.0* for more details.

- **CONFG: Configured**

Read:

0: Device is not in configured state

1: Device is in configured state

Write:

0: Sets device in a non configured state

1: Sets device in configured state

The device is set in configured state when it is in address state and receives a successful Set Configuration request. Refer to Chapter 9 of the *Universal Serial Bus Specification, Rev. 2.0* for more details.

- **ESR: Enable Send Resume**

0: Mandatory value prior to starting any Remote Wakeup procedure

1: Starts the Remote Wakeup procedure if this bit value was 0 and if RMWUPE is enabled

- **RMWUPE: Remote Wakeup Enable**

0: The Remote Wakeup feature of the device is disabled.

1: The Remote Wakeup feature of the device is enabled.

37.7.3 UDP Function Address Register

Name: UDP_FADDR

Address: 0x40044008

Access: Read/Write

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

- **FADD: Function Address Value**

The Function Address Value must be programmed by firmware once the device receives a set address request from the host, and has achieved the status stage of the no-data control sequence. Refer to the *Universal Serial Bus Specification, Rev. 2.0* for more information. After power up or reset, the function address value is set to 0.

- **FEN: Function Enable**

Read:

0: Function endpoint disabled

1: Function endpoint enabled

Write:

0: Disables function endpoint

1: Default value

The Function Enable bit (FEN) allows the microcontroller to enable or disable the function endpoints. The microcontroller sets this bit after receipt of a reset from the host. Once this bit is set, the USB device is able to accept and transfer data packets from and to the host.

37.7.4 UDP Interrupt Enable Register

Name: UDP_IER

Address: 0x40044010

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	WAKEUP	–	SOFINT	EXTRSM	RXRSM	RXSUSP
7	6	5	4	3	2	1	0
		EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Enable Endpoint 0 Interrupt**
- **EP1INT: Enable Endpoint 1 Interrupt**
- **EP2INT: Enable Endpoint 2Interrupt**
- **EP3INT: Enable Endpoint 3 Interrupt**
- **EP4INT: Enable Endpoint 4 Interrupt**
- **EP5INT: Enable Endpoint 5 Interrupt**

0: No effect

1: Enables corresponding Endpoint Interrupt

- **RXSUSP: Enable UDP Suspend Interrupt**

0: No effect

1: Enables UDP Suspend Interrupt

- **RXRSM: Enable UDP Resume Interrupt**

0: No effect

1: Enables UDP Resume Interrupt

- **SOFINT: Enable Start Of Frame Interrupt**

0: No effect

1: Enables Start Of Frame Interrupt

- **WAKEUP: Enable UDP Bus Wakeup Interrupt**

0: No effect

1: Enables USB bus Interrupt

37.7.5 UDP Interrupt Disable Register

Name: UDP_IDR

Address: 0x40044014

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	WAKEUP	–	SOFINT	EXTRSM	RXRSM	RXSUSP
7	6	5	4	3	2	1	0
		EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Disable Endpoint 0 Interrupt**
- **EP1INT: Disable Endpoint 1 Interrupt**
- **EP2INT: Disable Endpoint 2 Interrupt**
- **EP3INT: Disable Endpoint 3 Interrupt**
- **EP4INT: Disable Endpoint 4 Interrupt**
- **EP5INT: Disable Endpoint 5 Interrupt**

0: No effect

1: Disables corresponding Endpoint Interrupt

- **RXSUSP: Disable UDP Suspend Interrupt**

0: No effect

1: Disables UDP Suspend Interrupt

- **RXRSM: Disable UDP Resume Interrupt**

0: No effect

1: Disables UDP Resume Interrupt

- **SOFINT: Disable Start Of Frame Interrupt**

0: No effect

1: Disables Start Of Frame Interrupt

- **WAKEUP: Disable USB Bus Interrupt**

0: No effect

1: Disables USB Bus Wakeup Interrupt

37.7.6 UDP Interrupt Mask Register

Name: UDP_IMR

Address: 0x40044018

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	WAKEUP	BIT12	SOFINT	EXTRSM	RXRSM	RXSUSP
7	6	5	4	3	2	1	0
		EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Mask Endpoint 0 Interrupt**
- **EP1INT: Mask Endpoint 1 Interrupt**
- **EP2INT: Mask Endpoint 2 Interrupt**
- **EP3INT: Mask Endpoint 3 Interrupt**
- **EP4INT: Mask Endpoint 4 Interrupt**
- **EP5INT: Mask Endpoint 5 Interrupt**

0: Corresponding Endpoint Interrupt is disabled

1: Corresponding Endpoint Interrupt is enabled

- **RXSUSP: Mask UDP Suspend Interrupt**

0: UDP Suspend Interrupt is disabled

1: UDP Suspend Interrupt is enabled

- **RXRSM: Mask UDP Resume Interrupt.**

0: UDP Resume Interrupt is disabled

1: UDP Resume Interrupt is enabled

- **SOFINT: Mask Start Of Frame Interrupt**

0: Start of Frame Interrupt is disabled

1: Start of Frame Interrupt is enabled

- **BIT12: UDP_IMR Bit 12**

Bit 12 of UDP_IMR cannot be masked and is always read at 1.

- **WAKEUP: USB Bus Wakeup Interrupt**

0: USB Bus Wakeup Interrupt is disabled

1: USB Bus Wakeup Interrupt is enabled

Note: When the USB block is in suspend mode, the application may power down the USB logic. In this case, any USB HOST resume request that is made must be taken into account and, thus, the reset value of the RXRSM bit of the register UDP_IMR is enabled.

37.7.7 UDP Interrupt Status Register

Name: UDP_ISR

Address: 0x4004401C

Access: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	WAKEUP	ENDBUSRES	SOFINT	EXTRSM	RXRSM	RXSUSP
7	6	5	4	3	2	1	0
		EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Endpoint 0 Interrupt Status**
- **EP1INT: Endpoint 1 Interrupt Status**
- **EP2INT: Endpoint 2 Interrupt Status**
- **EP3INT: Endpoint 3 Interrupt Status**
- **EP4INT: Endpoint 4 Interrupt Status**
- **EP5INT: Endpoint 5 Interrupt Status**

0: No Endpointx Interrupt pending

1: Endpointx Interrupt has been raised

Several signals can generate this interrupt. The reason can be found by reading UDP_CSR0:

RXSETUP set to 1

RX_DATA_BK0 set to 1

RX_DATA_BK1 set to 1

TXCOMP set to 1

STALLSENT set to 1

EP0INT is a sticky bit. Interrupt remains valid until EP0INT is cleared by writing in the corresponding UDP_CSR0 bit.

- **RXSUSP: UDP Suspend Interrupt Status**

0: No UDP Suspend Interrupt pending

1: UDP Suspend Interrupt has been raised

The USB device sets this bit when it detects no activity for 3 ms. The USB device enters Suspend mode.

- **RXRSM: UDP Resume Interrupt Status**

0: No UDP Resume Interrupt pending

1: UDP Resume Interrupt has been raised

The USB device sets this bit when a UDP resume signal is detected at its port.

After reset, the state of this bit is undefined, the application must clear this bit by setting the RXRSM flag in the UDP_ICR.

- **SOFINT: Start of Frame Interrupt Status**

0: No Start of Frame Interrupt pending

1: Start of Frame Interrupt has been raised

This interrupt is raised each time a SOF token has been detected. It can be used as a synchronization signal by using isochronous endpoints.

- **ENDBUSRES: End of BUS Reset Interrupt Status**

0: No End of Bus Reset Interrupt pending

1: End of Bus Reset Interrupt has been raised

This interrupt is raised at the end of a UDP reset sequence. The USB device must prepare to receive requests on the endpoint 0. The host starts the enumeration, then performs the configuration.

- **WAKEUP: UDP Resume Interrupt Status**

0: No Wakeup Interrupt pending

1: A Wakeup Interrupt (USB Host Sent a RESUME or RESET) occurred since the last clear.

After reset the state of this bit is undefined; the application must clear this bit by setting the WAKEUP flag in the UDP_ICR.

37.7.8 UDP Interrupt Clear Register

Name: UDP_ICR

Address: 0x40044020

Access: Write-only

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

- **RXSUSP: Clear UDP Suspend Interrupt**

0: No effect

1: Clears UDP Suspend Interrupt

- **RXRSM: Clear UDP Resume Interrupt**

0: No effect

1: Clears UDP Resume Interrupt

- **SOFINT: Clear Start Of Frame Interrupt**

0: No effect

1: Clears Start Of Frame Interrupt

- **ENDBUSRES: Clear End of Bus Reset Interrupt**

0: No effect

1: Clears End of Bus Reset Interrupt

- **WAKEUP: Clear Wakeup Interrupt**

0: No effect

1: Clears Wakeup Interrupt

37.7.9 UDP Reset Endpoint Register

Name: UDP_RST_EP

Address: 0x40044028

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
		EP5	EP4	EP3	EP2	EP1	EP0

- **EP0: Reset Endpoint 0**
- **EP1: Reset Endpoint 1**
- **EP2: Reset Endpoint 2**
- **EP3: Reset Endpoint 3**
- **EP4: Reset Endpoint 4**
- **EP5: Reset Endpoint 5**

This flag is used to reset the FIFO associated with the endpoint and the bit RXBYTECOUNT in the UDP_CSRx. It also resets the data toggle to DATA0. It is useful after removing a HALT condition on a BULK endpoint. Refer to Chapter 5.8.5 in the *USB Serial Bus Specification, Rev.2.0*.

Warning: This flag must be cleared at the end of the reset. It does not clear UDP_CSRx flags.

0: No reset

1: Forces the corresponding endpoint FIF0 pointers to 0, therefore RXBYTECNT field is read at 0 in UDP_CSRx

Resetting the endpoint is a two-step operation:

1. Set the corresponding EPx field.
2. Clear the corresponding EPx field.

37.7.10 UDP Endpoint Control and Status Register (CONTROL_BULK)

Name: UDP_CSRx [x = 0..5] (CONTROL_BULK)

Address: 0x40044030

Access: Read/Write

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

23	22	21	20	19	18	17	16
RXBYTECNT							

15	14	13	12	11	10	9	8
EPEDS	–	–	–	DTGLE	EPTYPE		

7	6	5	4	3	2	1	0
DIR	RX_DATA_BK1	FORCESTALL	TXPKTRDY	STALLSENT	RXSETUP	RX_DATA_ BK0	TXCOMP

WARNING: Due to synchronization between MCK and UDPCK, the software application must wait for the end of the write operation before executing another write by polling the bits which must be set/cleared.

As an example, to perform a control operation on the endpoint without modifying the status flags while accessing the control bits and fields of this register, the status flag bits must first be defined with the “No effect” value ‘1’. Once the overall UDP_CSR value is defined, the register can be written and then the synchronization wait procedure must be executed.

```
//! Bitmap for all status bits in CSR that are not affected by a value 1.
#define UDP_REG_NO_EFFECT_1_ALL (UDP_CSR_RX_DATA_BK0 |\
                                 UDP_CSR_RX_DATA_BK1 |\
                                 UDP_CSR_STALLSENT   |\
                                 UDP_CSR_RXSETUP     |\
                                 UDP_CSR_TXCOMP)



/*! Sets specified bit(s) in the UDP_CSR.
*  \param ep   Endpoint number.
*  \param bits Bitmap to set to 1.
*/

#define udp_set_csr(ep, bits)                                    \
       do {                                                       \
              volatile uint32_t reg;                             \
              volatile uint32_t nop_count;                       \
              reg  = UDP->UDP_CSR[ep];                           \
              reg |= UDP_REG_NO_EFFECT_1_ALL;                    \
              reg |= (bits);                                     \
              UDP->UDP_CSR[ep] = reg;                            \
              for (nop_count = 0; nop_count < 20; nop_count ++) {\
                     __NOP();                                   \
              }                                                  \
       } while (0)
```

```
/*! Clears specified bit(s) in the UDP_CSR.
*  \param ep   Endpoint number.
*  \param bits Bitmap to set to 0.
*/
#define udp_clear_csr(ep, bits)                                    \
       do {                                                         \
              volatile uint32_t reg;                               \
              volatile uint32_t nop_count;                         \
              reg  = UDP->UDP_CSR[ep];                             \
              reg |= UDP_REG_NO_EFFECT_1_ALL;                      \
              reg &= ~(bits);                                      \
              UDP->UDP_CSR[ep] = reg;                              \
              for (nop_count = 0; nop_count < 20; nop_count ++) {\
                     __NOP();                                     \
              }                                                    \
       } while (0)
```

In a preemptive environment, set or clear the flag and wait for a time of 1 UDPCK clock cycle and 1peripheral clock cycle. However, RX_DATA_BK0, TXPKTRDY, RX_DATA_BK1 require wait times of 3 UDPCK clock cycles and 5 peripheral clock cycles before accessing DPR.

- **TXCOMP: Generates an IN Packet with Data Previously Written in the DPR**

This flag generates an interrupt while it is set to one.

Write (cleared by the firmware):

0: Clear the flag, clear the interrupt

1: No effect

Read (Set by the USB peripheral):

0: Data IN transaction has not been acknowledged by the Host

1: Data IN transaction is achieved, acknowledged by the Host

After having issued a Data IN transaction setting TXPKTRDY, the device firmware waits for TXCOMP to be sure that the host has acknowledged the transaction.

- **RX_DATA_BK0: Receive Data Bank 0**

This flag generates an interrupt while it is set to one.

Write (cleared by the firmware):

0: Notify USB peripheral device that data have been read in the FIFO's Bank 0.

1: To leave the read value unchanged.

Read (Set by the USB peripheral):

0: No data packet has been received in the FIFO's Bank 0.

1: A data packet has been received, it has been stored in the FIFO's Bank 0.

When the device firmware has polled this bit or has been interrupted by this signal, it must transfer data from the FIFO to the microcontroller memory. The number of bytes received is available in RXBYTCENT field. Bank 0 FIFO values are read through the UDP_FDRx. Once a transfer is done, the device firmware must release Bank 0 to the USB peripheral device by clearing RX_DATA_BK0.

After setting or clearing this bit, a wait time of 3 UDPCK clock cycles and 3 peripheral clock cycles is required before accessing DPR.

- **RXSETUP: Received Setup**

This flag generates an interrupt while it is set to one.

Read:

0: No setup packet available.

1: A setup data packet has been sent by the host and is available in the FIFO.

Write:

0: Device firmware notifies the USB peripheral device that it has read the setup data in the FIFO.

1: No effect.

This flag is used to notify the USB device firmware that a valid Setup data packet has been sent by the host and successfully received by the USB device. The USB device firmware may transfer Setup data from the FIFO by reading the UDP_FDRx to the microcontroller memory. Once a transfer has been done, RXSETUP must be cleared by the device firmware.

Ensuing Data OUT transaction is not accepted while RXSETUP is set.

- **STALLSENT: Stall Sent**

This flag generates an interrupt while it is set to one.

This ends a STALL handshake.

Read:

0: Host has not acknowledged a stall

1: Host has acknowledged the stall

Write:

0: Resets the STALLSENT flag, clears the interrupt

1: No effect

This is mandatory for the device firmware to clear this flag. Otherwise the interrupt remains.

Refer to chapters 8.4.5 and 9.4.5 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the STALL handshake.

- **TXPKTRDY: Transmit Packet Ready**

This flag is cleared by the USB device.

This flag is set by the USB device firmware.

Read:

0: There is no data to send.

1: The data is waiting to be sent upon reception of token IN.

Write:

0: Can be used in the procedure to cancel transmission data. (See Section 37.6.2.5 “Transmit Data Cancellation” on page 973)

1: A new data payload has been written in the FIFO by the firmware and is ready to be sent.

This flag is used to generate a Data IN transaction (device to host). Device firmware checks that it can write a data payload in the FIFO, checking that TXPKTRDY is cleared. Transfer to the FIFO is done by writing in the UDP_FDRx. Once the data

payload has been transferred to the FIFO, the firmware notifies the USB device setting TXPKTRDY to one. USB bus transactions can start. TXCOMP is set once the data payload has been received by the host.

After setting or clearing this bit, a wait time of 3 UDPCK clock cycles and 3 peripheral clock cycles is required before accessing DPR.

- **FORCESTALL: Force Stall (used by Control, Bulk and Isochronous Endpoints)**

Read:

0: Normal state

1: Stall state

Write:

0: Return to normal state

1: Send STALL to the host

Refer to chapters 8.4.5 and 9.4.5 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the STALL handshake.

Control endpoints: During the data stage and status stage, this bit indicates that the microcontroller cannot complete the request.

Bulk and interrupt endpoints: This bit notifies the host that the endpoint is halted.

The host acknowledges the STALL, device firmware is notified by the STALLSENT flag.

- **RX_DATA_BK1: Receive Data Bank 1 (only used by endpoints with ping-pong attributes)**

This flag generates an interrupt while it is set to one.

Write (cleared by the firmware):

0: Notifies USB device that data have been read in the FIFO's Bank 1.

1: To leave the read value unchanged.

Read (Set by the USB peripheral):

0: No data packet has been received in the FIFO's Bank 1.

1: A data packet has been received, it has been stored in FIFO's Bank 1.

When the device firmware has polled this bit or has been interrupted by this signal, it must transfer data from the FIFO to microcontroller memory. The number of bytes received is available in RXBYTECNT field. Bank 1 FIFO values are read through UDP_FDRx. Once a transfer is done, the device firmware must release Bank 1 to the USB device by clearing RX_DATA_BK1.

After setting or clearing this bit, a wait time of 3 UDPCK clock cycles and 3 peripheral clock cycles is required before accessing DPR.

- **DIR: Transfer Direction (only available for control endpoints) (Read/Write)**

0: Allows Data OUT transactions in the control data stage.

1: Enables Data IN transactions in the control data stage.

Refer to Chapter 8.5.3 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the control data stage.

This bit must be set before UDP_CSRx/RXSETUP is cleared at the end of the setup stage. According to the request sent in the setup data packet, the data stage is either a device to host (DIR = 1) or host to device (DIR = 0) data transfer. It is not necessary to check this bit to reverse direction for the status stage.

- **EPTYPE: Endpoint Type (Read/Write)**

Value	Name	Description
0	CTRL	Control
1	ISO_OUT	Isochronous OUT
2	BULK_OUT	Bulk OUT
3	INT_OUT	Interrupt OUT
4	–	Reserved
5	ISO_IN	Isochronous IN
6	BULK_IN	Bulk IN
7	INT_IN	Interrupt IN

- **DTGLE: Data Toggle (Read-only)**

0: Identifies DATA0 packet

1: Identifies DATA1 packet

Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on DATA0, DATA1 packet definitions.

- **EPEDS: Endpoint Enable Disable**

Read:

0: Endpoint disabled

1: Endpoint enabled

Write:

0: Disables endpoint

1: Enables endpoint

Control endpoints are always enabled. Reading or writing this field has no effect on control endpoints.

Note: After reset, all endpoints are configured as control endpoints (zero).

- **RXBYTECNT: Number of Bytes Available in the FIFO (Read-only)**

When the host sends a data packet to the device, the USB device stores the data in the FIFO and notifies the microcontroller. The microcontroller can load the data from the FIFO by reading RXBYTECENT bytes in the UDP_FDRx.

37.7.11 UDP Endpoint Control and Status Register (ISOCHRONOUS)

Name: UDP_CSRx [x = 0..5] (ISOCHRONOUS)

Address: 0x40044030

Access: Read/Write

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

23	22	21	20	19	18	17	16
RXBYTECNT							

15	14	13	12	11	10	9	8
EPEDS	–	–	–	DTGLE	EPTYPE		

7	6	5	4	3	2	1	0
DIR	RX_DATA_BK1	FORCESTALL	TXPKTRDY	ISOERROR	RXSETUP	RX_DATA_ BK0	TXCOMP

- **TXCOMP: Generates an IN Packet with Data Previously Written in the DPR**

This flag generates an interrupt while it is set to one.

Write (cleared by the firmware):

0: Clear the flag, clear the interrupt.

1: No effect.

Read (Set by the USB peripheral):

0: Data IN transaction has not been acknowledged by the Host.

1: Data IN transaction is achieved, acknowledged by the Host.

After having issued a Data IN transaction setting TXPKTRDY, the device firmware waits for TXCOMP to be sure that the host has acknowledged the transaction.

- **RX_DATA_BK0: Receive Data Bank 0**

This flag generates an interrupt while it is set to one.

Write (cleared by the firmware):

0: Notify USB peripheral device that data have been read in the FIFO's Bank 0.

1: To leave the read value unchanged.

Read (Set by the USB peripheral):

0: No data packet has been received in the FIFO's Bank 0.

1: A data packet has been received, it has been stored in the FIFO's Bank 0.

When the device firmware has polled this bit or has been interrupted by this signal, it must transfer data from the FIFO to the microcontroller memory. The number of bytes received is available in RXBYTCENT field. Bank 0 FIFO values are read through the UDP_FDRx. Once a transfer is done, the device firmware must release Bank 0 to the USB peripheral device by clearing RX_DATA_BK0.

After setting or clearing this bit, a wait time of 3 UDPCK clock cycles and 3 peripheral clock cycles is required before accessing DPR.

- **RXSETUP: Received Setup**

This flag generates an interrupt while it is set to one.

Read:

0: No setup packet available.

1: A setup data packet has been sent by the host and is available in the FIFO.

Write:

0: Device firmware notifies the USB peripheral device that it has read the setup data in the FIFO.

1: No effect.

This flag is used to notify the USB device firmware that a valid Setup data packet has been sent by the host and successfully received by the USB device. The USB device firmware may transfer Setup data from the FIFO by reading the UDP_FDRx to the microcontroller memory. Once a transfer has been done, RXSETUP must be cleared by the device firmware.

Ensuing Data OUT transaction is not accepted while RXSETUP is set.

- **ISOERROR: A CRC error has been detected in an isochronous transfer**

This flag generates an interrupt while it is set to one.

Read:

0: No error in the previous isochronous transfer.

1: CRC error has been detected, data available in the FIFO are corrupted.

Write:

0: Resets the ISOERROR flag, clears the interrupt.

1: No effect.

- **TXPKTRDY: Transmit Packet Ready**

This flag is cleared by the USB device.

This flag is set by the USB device firmware.

Read:

0: There is no data to send.

1: The data is waiting to be sent upon reception of token IN.

Write:

0: Can be used in the procedure to cancel transmission data. (See Section 37.6.2.5 “Transmit Data Cancellation” on page 973)

1: A new data payload has been written in the FIFO by the firmware and is ready to be sent.

This flag is used to generate a Data IN transaction (device to host). Device firmware checks that it can write a data payload in the FIFO, checking that TXPKTRDY is cleared. Transfer to the FIFO is done by writing in the UDP_FDRx. Once the data payload has been transferred to the FIFO, the firmware notifies the USB device setting TXPKTRDY to one. USB bus transactions can start. TXCOMP is set once the data payload has been received by the host.

After setting or clearing this bit, a wait time of 3 UDPCK clock cycles and 3 peripheral clock cycles is required before accessing DPR.

- **FORCESTALL: Force Stall (used by Control, Bulk and Isochronous Endpoints)**

Read:

0: Normal state.

1: Stall state.

Write:

0: Return to normal state.

1: Send STALL to the host.

Refer to chapters 8.4.5 and 9.4.5 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the STALL handshake.

Control endpoints: During the data stage and status stage, this bit indicates that the microcontroller cannot complete the request.

Bulk and interrupt endpoints: This bit notifies the host that the endpoint is halted.

The host acknowledges the STALL, device firmware is notified by the STALLSENT flag.

- **RX_DATA_BK1: Receive Data Bank 1 (only used by endpoints with ping-pong attributes)**

This flag generates an interrupt while it is set to one.

Write (cleared by the firmware):

0: Notifies USB device that data have been read in the FIFO's Bank 1.

1: To leave the read value unchanged.

Read (set by the USB peripheral):

0: No data packet has been received in the FIFO's Bank 1.

1: A data packet has been received, it has been stored in FIFO's Bank 1.

When the device firmware has polled this bit or has been interrupted by this signal, it must transfer data from the FIFO to microcontroller memory. The number of bytes received is available in RXBYTECNT field. Bank 1 FIFO values are read through UDP_FDRx. Once a transfer is done, the device firmware must release Bank 1 to the USB device by clearing RX_DATA_BK1.

After setting or clearing this bit, a wait time of 3 UDPCK clock cycles and 3 peripheral clock cycles is required before accessing DPR.

- **DIR: Transfer Direction (only available for control endpoints) (Read/Write)**

0: Allows Data OUT transactions in the control data stage.

1: Enables Data IN transactions in the control data stage.

Refer to Chapter 8.5.3 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the control data stage.

This bit must be set before UDP_CSRx/RXSETUP is cleared at the end of the setup stage. According to the request sent in the setup data packet, the data stage is either a device to host (DIR = 1) or host to device (DIR = 0) data transfer. It is not necessary to check this bit to reverse direction for the status stage.

- **EPTYPE: Endpoint Type (Read/Write)**

Value	Name	Description
0	CTRL	Control
1	ISO_OUT	Isochronous OUT
2	BULK_OUT	Bulk OUT
3	INT_OUT	Interrupt OUT
4	–	Reserved
5	ISO_IN	Isochronous IN
6	BULK_IN	Bulk IN
7	INT_IN	Interrupt IN

- **DTGLE: Data Toggle (Read-only)**

0: Identifies DATA0 packet

1: Identifies DATA1 packet

Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on DATA0, DATA1 packet definitions.

- **EPEDS: Endpoint Enable Disable**

Read:

0: Endpoint disabled

1: Endpoint enabled

Write:

0: Disables endpoint

1: Enables endpoint

Control endpoints are always enabled. Reading or writing this field has no effect on control endpoints.

Note: After reset, all endpoints are configured as control endpoints (zero).

- **RXBYTECNT: Number of Bytes Available in the FIFO (Read-only)**

When the host sends a data packet to the device, the USB device stores the data in the FIFO and notifies the microcontroller. The microcontroller can load the data from the FIFO by reading RXBYTECENT bytes in the UDP_FDRx.

37.7.12 UDP FIFO Data Register

Name: UDP_FDRx [x = 0..5]

Address: 0x40044050

Access: Read/Write

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

- **FIFO_DATA: FIFO Data Value**

The microcontroller can push or pop values in the FIFO through this register.

RXBYTECNT in the corresponding UDP_CSRx is the number of bytes to be read from the FIFO (sent by the host).

The maximum number of bytes to write is fixed by the Max Packet Size in the Standard Endpoint Descriptor. It can not be more than the physical memory size associated to the endpoint. Refer to the *Universal Serial Bus Specification, Rev. 2.0* for more information.

37.7.13 UDP Transceiver Control Register

Name: UDP_TXVC

Address: 0x40044074

Access: Read/Write

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

WARNING: The UDP peripheral clock in the PMC must be enabled before any read/write operations to the UDP registers including the UDP_TXVC register.

- **TXVDIS: Transceiver Disable**

When UDP is disabled, power consumption can be reduced significantly by disabling the embedded transceiver. This can be done by setting TXVDIS bit.

To enable the transceiver, TXVDIS must be cleared.

- **PUON: Pull-up On**

0: The 1.5KΩ integrated pull-up on DDP is disconnected.

1: The 1.5 KΩ integrated pull-up on DDP is connected.

Note: If the USB pull-up is not connected on DDP, the user should not write in any UDP register other than the UDP_TXVC register. This is because if DDP and DDM are floating at 0, or pulled down, then SE0 is received by the device with the consequence of a USB Reset.

38. Analog-to-Digital Converter (ADC)

38.1 Description

The ADC is based on a 12-bit Analog-to-Digital Converter (ADC) managed by an ADC Controller providing enhanced resolution up to 16 bits. Refer to Figure 38-1 “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 VDDIO.

Conversion results are reported in a common register for all channels, as well as in a channel-dedicated register.

The 13-bit,14-bit,15-bit and 16-bit resolution modes are obtained by averaging multiple samples to decrease quantization noise. For the 13-bit mode, 4 samples are used, which gives a real sample rate of 1/4 of the actual sample frequency. For the 14-bit mode, 16 samples are used, giving a real sample rate of 1/16 of the actual sample frequency. For the 15-bit and 16-bit modes, respectively 64 and 256 samples are used, giving a real sample rate of respectively 1/64 and 1/256 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.

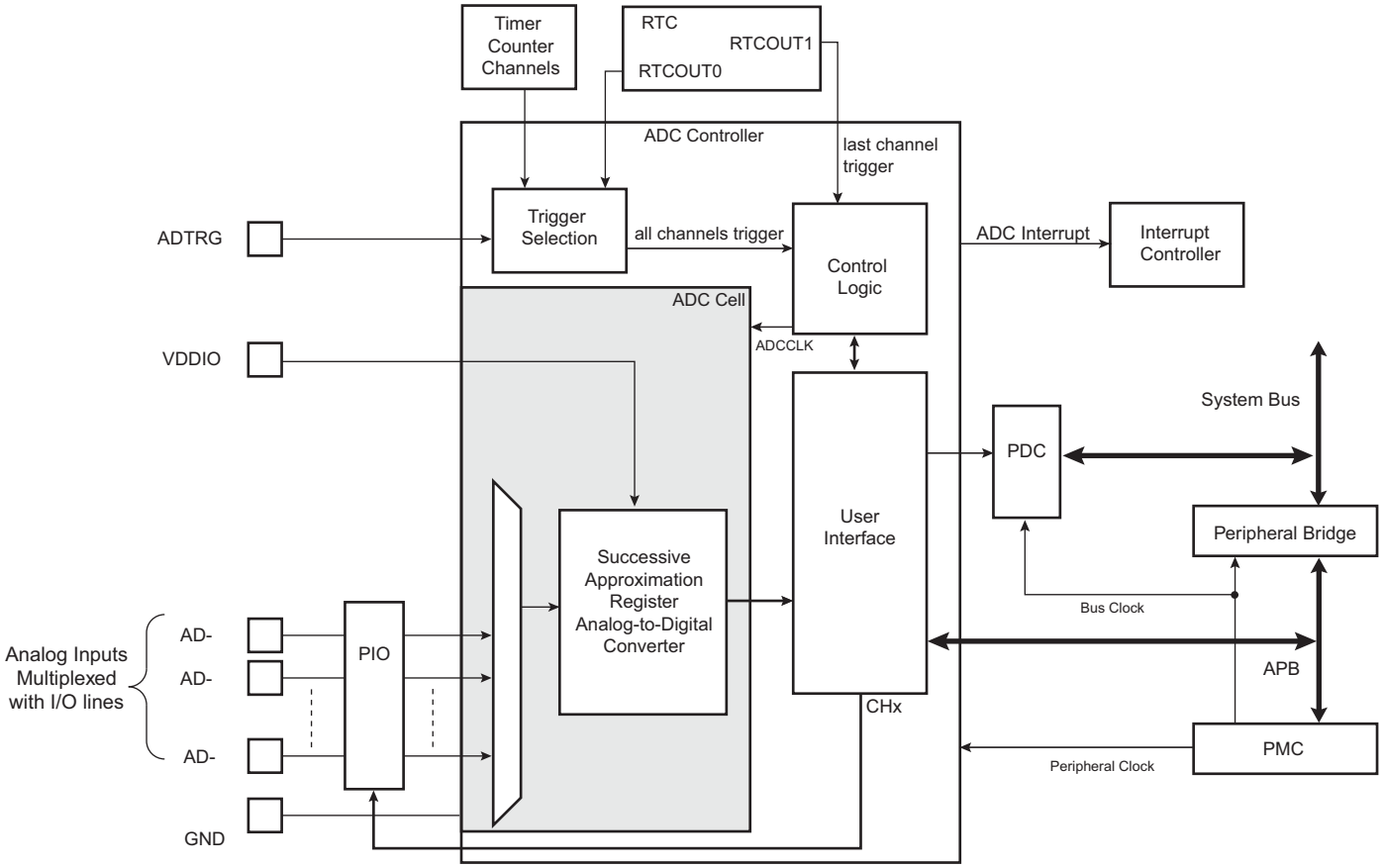
This ADC has a selectable single-ended or full differential input.

38.2 Embedded Characteristics

- 12-bit Resolution with Enhanced Mode up to 16 bits
- 500 ksps Conversion Rate
- Digital Averaging Function providing Enhanced Resolution Mode up to 16 bits
- Wide Range of Power Supply Operation
- Selectable Single-Ended or Differential Input Voltage
- Integrated Multiplexer Offering Up to 8 Independent Analog Inputs
- Individual Enable and Disable of Each Channel
- Hardware or Software Trigger from:
  - External Trigger Pin
  - Timer Counter Outputs (Corresponding TIOA Trigger)
- Up to 2 Trigger Events With Independent Rates
- PDC Support
- Two Sleep Modes (Automatic Wakeup on Trigger)
  - Lowest Power Consumption (Voltage Reference OFF Between Conversions)
  - Fast Wakeup Time Response on Trigger Event (Voltage Reference ON Between Conversions)
- Channel Sequence Customization
- Automatic Window Comparison of Converted Values
- Asynchronous Partial Wakeup (SleepWalking) on external trigger
- Register Write Protection

38.3 Block Diagram

Figure 38-1. Analog-to-Digital Converter Block Diagram



38.4 Signal Description

Table 38-1. ADC Pin Description

Pin Name	Description
AD0–AD7	Analog input Channels
ADTRG	External Trigger

38.5 Product Dependencies

38.5.1 Power Management

The ADC Controller is not continuously clocked. The programmer must first enable the ADC Controller peripheral clock in the Power Management Controller (PMC) before using the ADC Controller. However, if the application does not require ADC operations, the ADC Controller clock can be stopped when not needed and restarted when necessary. Configuring the ADC Controller does not require the ADC Controller clock to be enabled.

38.5.2 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 38-2. Peripheral IDs

Instance	ID
ADC	29

38.5.3 I/O Lines

The digital input ADTRG is multiplexed with digital functions on the I/O line and the selection of ADTRG is made using the PIO controller.

The analog inputs ADC_ADx are multiplexed with digital functions on the I/O lines. ADC_ADx inputs are selected as inputs of the ADCC when writing a one in the corresponding CHx bit of ADC_CHER and the digital functions are not selected.

Table 38-3. 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

38.5.4 Hardware Triggers

The ADC can use internal signals to start conversions. Refer to the ADC_MR.TRGSEL field description for exact wiring of internal triggers.

38.6 Functional Description

38.6.1 Analog-to-Digital Conversion

Once the programmed startup time (ADC_MR.STARTUP) has elapsed, ADC conversions are sequenced by three operating times:

- Tracking time—the time for the ADC to charge its input sampling capacitor to the input voltage. When several channels are converted consecutively, the inherent tracking time is six ADC clock cycles. However, the tracking time can be increased using the TRACKTIM field in the Mode Register (ADC_MR).
- ADC inherent conversion time—the time for the ADC to convert the sampled analog voltage. This time is constant and is defined from start of conversion to end of conversion.
- Channel conversion period—the effective time between the end of the current channel conversion and the end of the next channel conversion.

Figure 38-2. Sequence of Consecutive ADC Conversions with TRACKTIM = 0

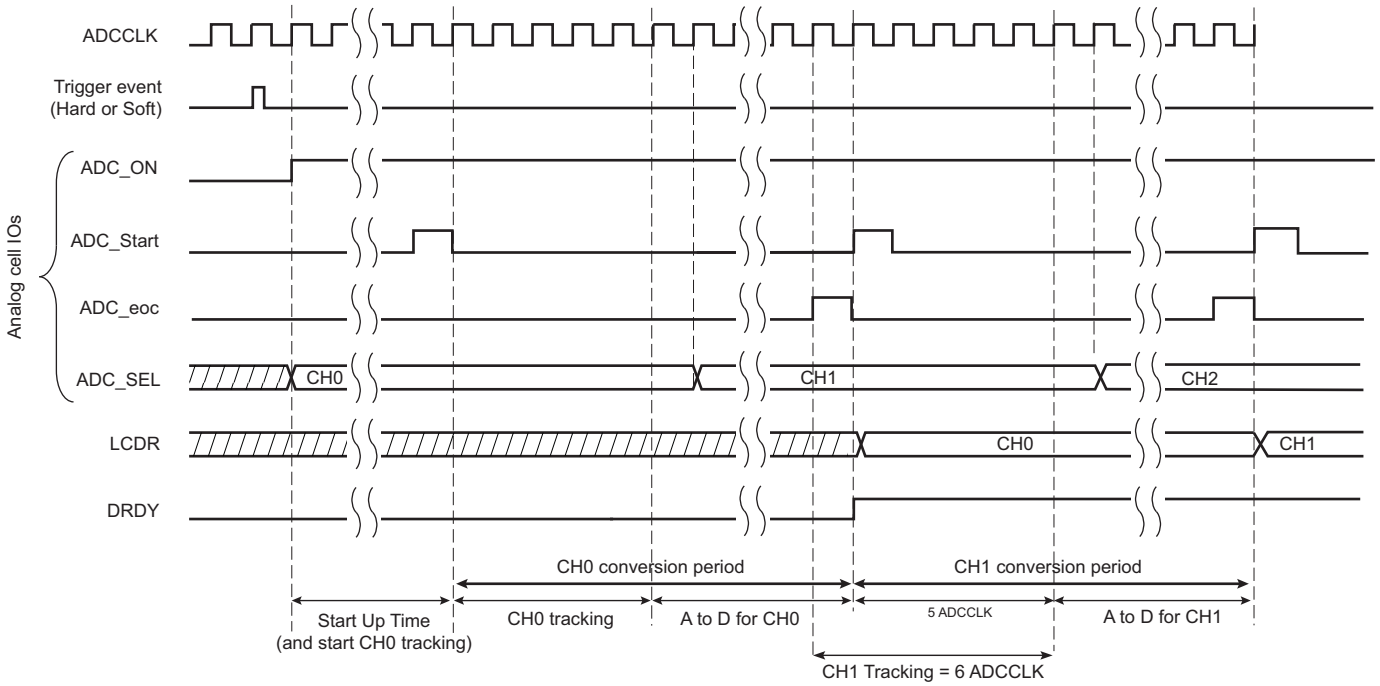
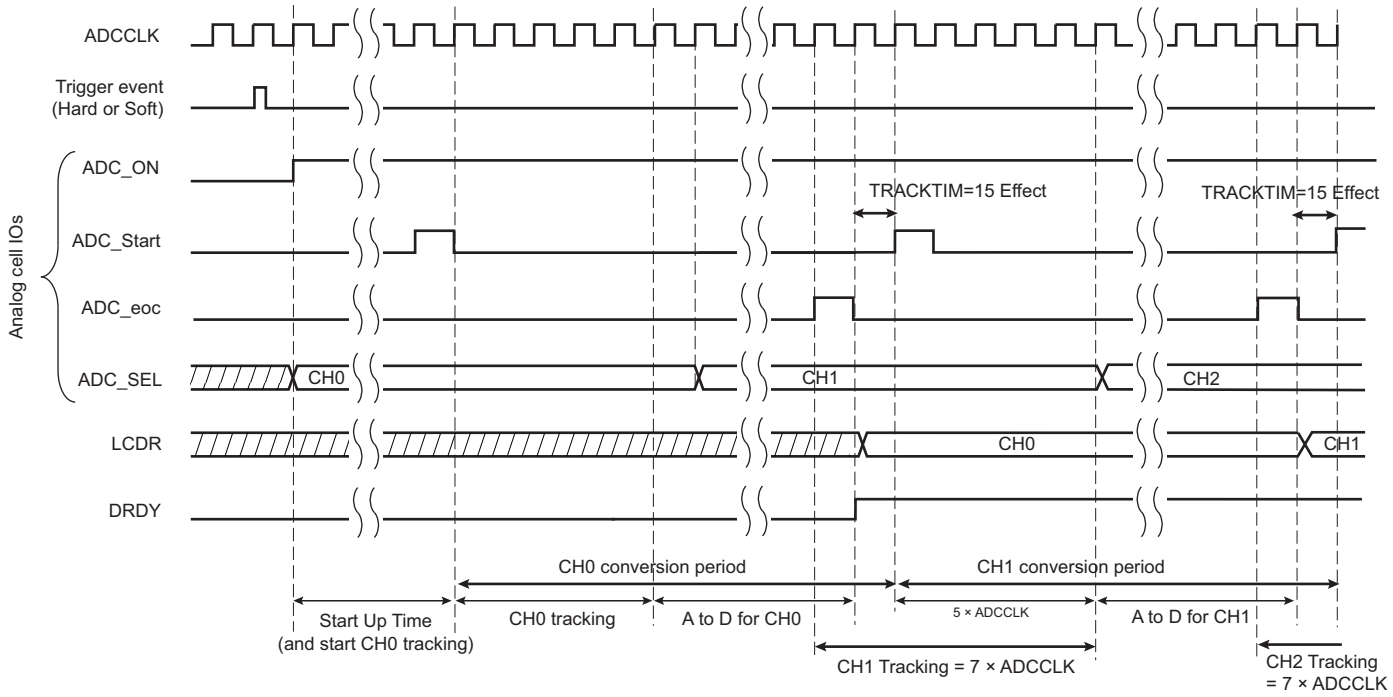


Figure 38-3. Sequence of Consecutive ADC Conversions with TRACKTIM = 15



38.6.2 ADC Clock

The ADC uses the ADC clock (ADCCLK) to perform conversions. The ADC clock frequency is selected in the PRESCAL field of ADC_MR.

To generate the ADC clock, the prescaler has two clock sources: the peripheral clock and the PMC_PCKx clock. This clock source is selected using the SRCCLK bit in the Extended Mode Register (ADC_EMR).

If PMC_PCKx is selected as a source clock, the ADC clock frequency is independent of the processor/bus clock. At reset, the peripheral clock is selected.

If the SRCCLK bit in ADC_EMR is cleared, then the prescaler clock (presc_clk) is driven by peripheral_clock. If the SRCCLK bit in ADC_EMR is set to 1, the prescaler clock is driven by PMC_PCKx. The ADC clock frequency is between fpresc_clk and fpresc_clk/512, if PRESCAL is set to 255 (0xFF).

PRESCAL must be programmed to provide the ADC clock frequency parameter given in the section “Electrical Characteristics”.

38.6.3 ADC Reference Voltage

The conversion is performed on a full range between 0V and the reference voltage connected to VDDIO.

Analog inputs between these voltages convert to values based on a linear conversion.

38.6.4 Conversion Resolution

The ADC analog cell features a 12-bit resolution.

The ADC digital controller provides enhanced resolution up to 16 bits.

If ADTRG is asynchronous to the ADC peripheral clock, the internal resynchronization introduces a jitter of 1 peripheral clock. This jitter may reduce the resolution of the converted signal.

The same applies when using the independent clock (ADC_MR.SRCCLK = 1), if the provided clock is asynchronous to ADC peripheral clock.

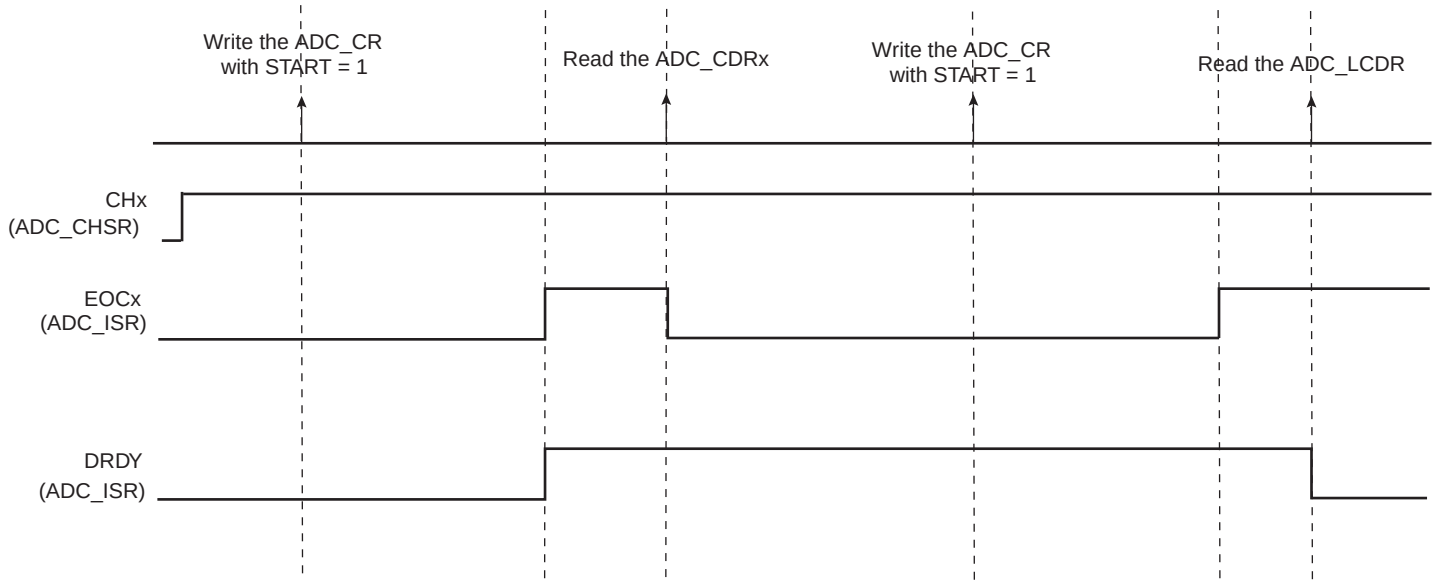
38.6.5 Conversion Results

When a conversion is completed, the resulting 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 Extended Mode Register (ADC_EMR), ADC_LCDR presents the channel number associated with the last converted data in the CHNB field.

When a conversion is completed, the channel EOC bit and the DRDY bit in the Interrupt Status Register (ADC_ISR) are set. In the case of a connected PDC channel, DRDY rising triggers a data request. In any case, either EOC and DRDY can trigger an interrupt.

Reading one of the ADC_CDRx clears the corresponding EOC bit. Reading ADC_LCDR clears the DRDY bit.

Figure 38-4. EOCx and DRDY Flag Behavior

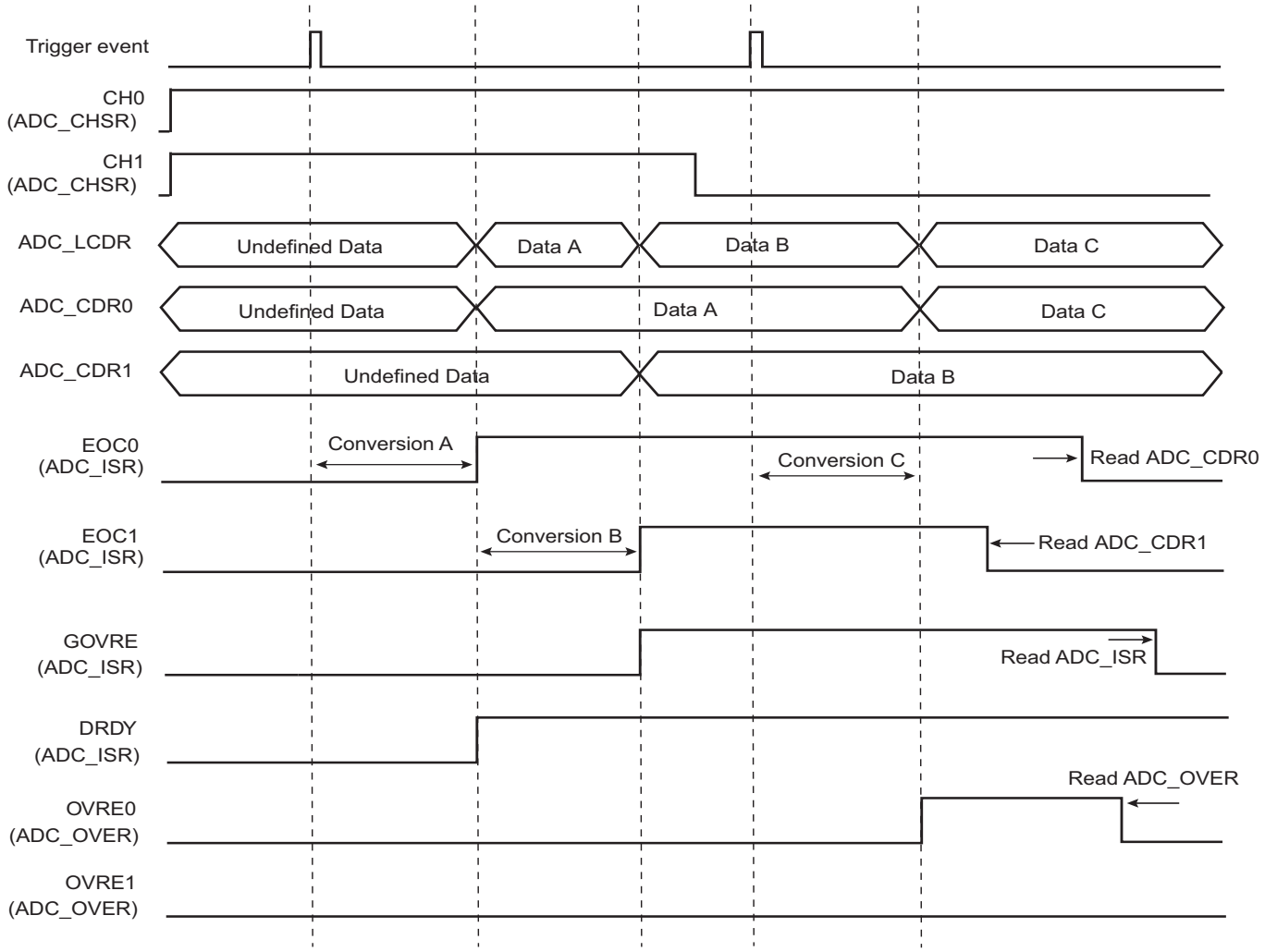


If ADC_CDR is not read before further incoming data is converted, the corresponding OVREx flag is set in the Overrun Status Register (ADC_OVER).

New data converted when DRDY is high sets the GOVRE bit in ADC_ISR.

The OVREx flag is automatically cleared when ADC_OVER is read, and the GOVRE flag is automatically cleared when ADC_ISR is read.

Figure 38-5. EOCx, OVREx and GOVREx 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 corresponding EOCx and GOVRE flags in ADC_ISR and OVREx flags in ADC_OVER are unpredictable.

38.6.6 Conversion Triggers

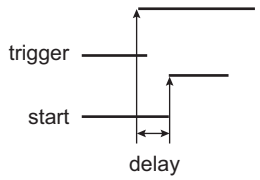
Conversions of the active analog channels are started with a software or hardware trigger. The software trigger is provided by writing the Control Register (ADC_CR) with the START bit at 1.

The list of external/internal events is provided in Section 38.7.2 “ADC Mode Register”. The hardware trigger is selected using the TRGSEL field in ADC_MR. When the TRGEN bit is set in ADC_MR, the selected hardware trigger is enabled and the software trigger is disabled.

The ADC also provides a dual trigger mode (ADC_LCTMR.DUALTRIG=1) in which the higher index channel can be sampled at a rhythm different from the other channels. The trigger of the last channel is generated by the RTC. Refer to Section 38.6.11 “Last Channel Specific Measurement Trigger”.

If a hardware trigger is selected, the start of a conversion is triggered after a delay starting at each rising edge of the selected signal. Due to asynchronous handling, the delay may vary in a range of two peripheral clock periods to one ADC clock period. This delay introduces sampling jitter in the A/D conversion process and may therefore degrade the conversion performance (e.g., SNR, THD).

Figure 38-6. 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 enable 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.

38.6.7 Sleep Mode and Conversion Sequencer

The ADC Sleep mode maximizes power saving by automatically deactivating the ADC when it is not being used for conversions. Sleep mode is selected by setting the SLEEP bit in ADC_MR.

Sleep mode is managed by a conversion sequencer, which automatically processes the conversions of all channels at lowest power consumption.

This mode can be used when the minimum period of time between two successive trigger events is greater than the startup period of the ADC. See section “Electrical Characteristics”.

When a start conversion request occurs, the ADC is automatically activated. As the analog cell requires a startup time, the logic waits during this time and starts the conversion on the enabled channels. When all conversions are complete, the ADC is deactivated until the next trigger. Triggers occurring during the sequence are ignored.

A Fast wakeup mode is available in ADC_MR as a compromise between power-saving strategy and responsiveness. Setting the FWUP bit enables the Fast wakeup mode. In Fast wakeup 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 via 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-1, in ADC_CHER.

Note: The reference voltage pins always remain connected in Normal mode as in Sleep mode.

38.6.8 Comparison Window

The ADC Controller features automatic comparison functions. It compares converted values to a low threshold, a high threshold or both, depending on the value of the CMPMODE bit in ADC_EMR. The comparison can be done on all channels or only on the channel specified in the CMPSEL field of ADC_EMR. To compare all channels, the CMPALL bit of ADC_EMR must be set.

If set to 1, the CMPTYPE bit of ADC_EMR can be used to discard all conversion results that do not match the comparison conditions. Once a conversion result matches the comparison conditions, all the subsequent conversion results are stored in ADC_LCDR (even if these results do not meet the comparison conditions). Writing a 1 to the CMPRST bit in ADC_CR immediately stops the conversion result storage until the next comparison match.

If the CMPTYPE bit in ADC_EMR is cleared, all conversions are stored in ADC_LCDR. Only the conversions that match the comparison conditions trigger the COMPE flag in ADC_ISR.

Moreover, a filtering option can be set by writing the number of consecutive comparison matches needed to raise the flag. This number can be written and read in the CMPFILTER field of ADC_EMR. The filtering option is dedicated to reinforce the detection of an analog signal overpassing a predefined threshold. The filter is cleared as soon as ADC_ISR is read, so this filtering function must be used with peripheral DMA controller and works only when using Interrupt mode (no polling).

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 Compare Window Register (ADC_CWR).

38.6.9 Differential and Single-ended Input Modes

The ADC can be configured to operate in the following input voltage modes:

- Single-ended—ADC_COR.DIFFx = 0. This is the default mode after a reset.
- Differential—ADC_COR.DIFFx = 1 (see Figure 38-7). In Differential mode, the ADC requires differential input signals having a VDD/2 common mode voltage (refer to the “Electrical Characteristics” section).

The following equations give the ADC input-output transfer function in each mode(1).

Single-ended mode:

$$\text{ADC\_LCDR.LDATA} = \frac{ADx - GND}{ADVREF - GND} \times 2^{12}$$

Differential mode:

$$\text{ADC\_LCDR.LDATA} = \left(1 + \frac{ADx - \text{ADx+1}}{ADVREF - GND}\right) \times 2^{11}$$

Note: 1. Equations assume ADC_EMR.OSR = 1

If the ANACH bit is set in ADC_MR, the ADC can manage both differential channels and single-ended channels. If the ANACH bit is cleared, the parameters defined in ADC_CGR, ADC_COR are applied to all channels.

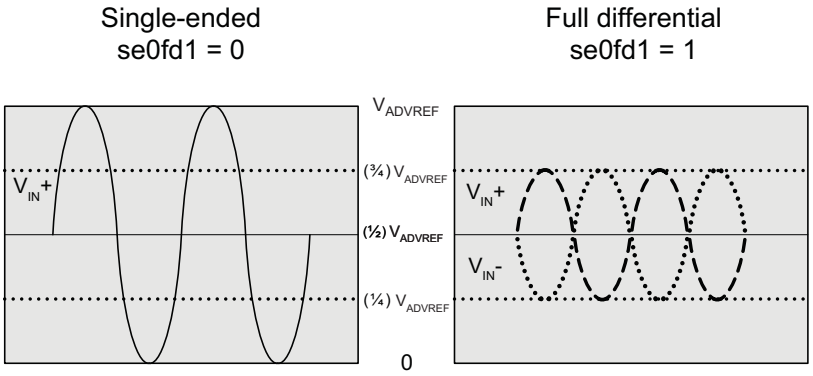
Table 38-4 gives the internal positive and negative ADC inputs assignment with respect to the programmed mode (ADC_COR.DIFFx). If Differential mode is enabled, the odd channel index is routed to the negative input.

For example, if Differential mode is required on channel 0, input pins AD0 and AD1 are used. In this case, only channel 0 must be enabled by writing a 1 to ADC_CHER.CH0.

Table 38-4. Input Pins and Channel Numbers

Input Pin	Channel Number	
	Single-ended Mode	Differential Mode
AD0	CH0	CH0
AD1	CH1	
AD2	CH2	CH2
AD3	CH3	
AD4	CH4	CH4
AD5	CH5	
AD6	CH6	CH6
AD7	CH7	

Figure 38-7. Analog Full Scale Ranges in Single-Ended and Differential Applications



38.6.10 ADC Timings

The ADC startup time is programmed through the STARTUP field in ADC_MR. See section “Electrical Characteristics”.

The ADC controller provides a tracking time of six ADC clock cycles.

It is possible to add one more ADC clock to the ADC tracking time by programming the TRACKTIM field in ADC_MR.When the offset or differential input parameters of the analog cell change between two channels, the analog cell may need a specific settling time before starting the tracking phase. In that case, the controller automatically waits during the settling time defined in ADC_MR. Obviously, if the ANACH option is not set, this time is unused.

Warning: No input buffer amplifier to isolate the source is included in the ADC. This must be taken into consideration. See section “Electrical Characteristics”.

38.6.11 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 with a variation of a different frequency from other converted channels (for example, but not limited to, temperature sensor).

The last channel can be sampled in different ways through the ADC controller. The different methods of sampling depend on the configuration bit TRGEN in ADC_MR and bit CH7 in ADC_CHSR.

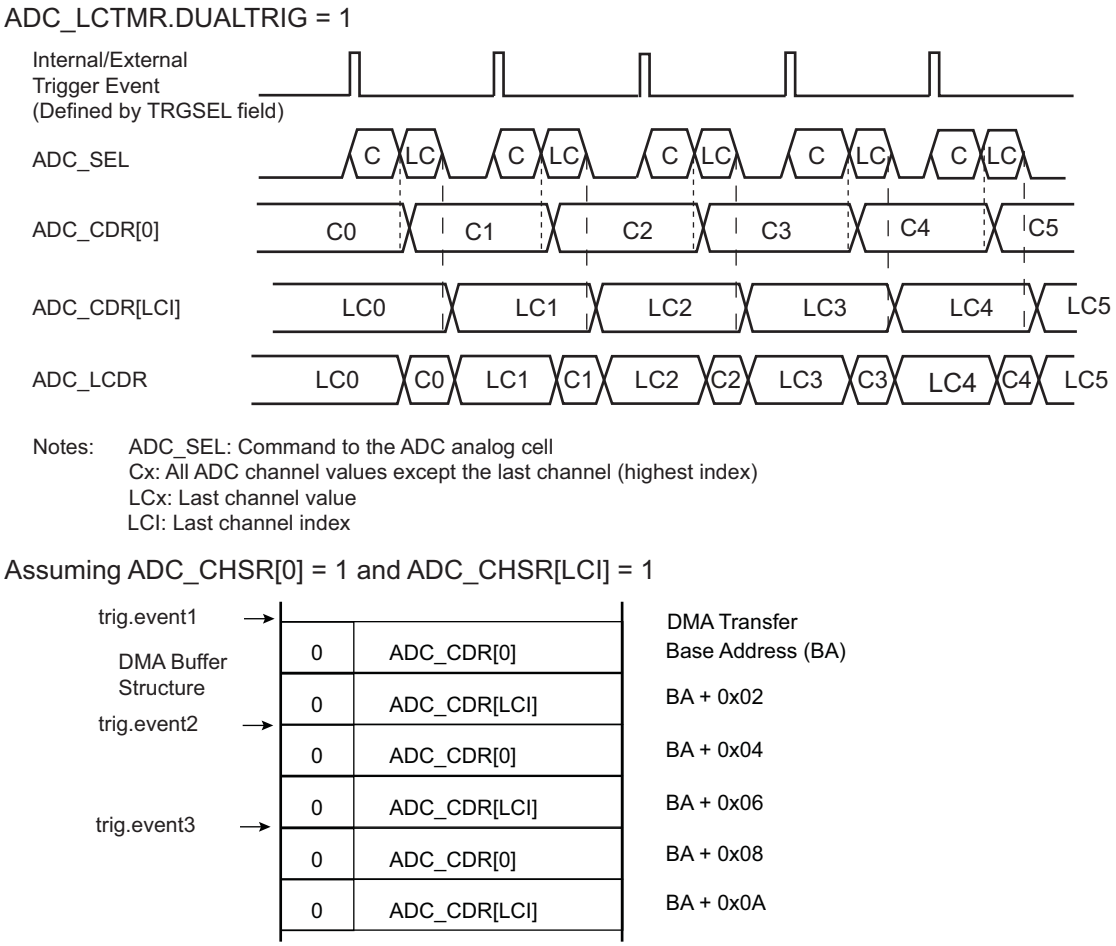
The last channel measure can be triggered like the other channels by enabling its associated conversion channel index 7, writing 1 in CH7 of ADC_CHER.

The manual start can only be performed if bit TRGEN is cleared. When the START bit in ADC_CR is set, the last channel conversion is scheduled together with the other enabled channels (if any). The result of the conversion is placed in ADC_CDR7 register and the associated flag EOC7 is set in ADC_ISR.

If the last channel is enabled in ADC_CHSR, DUALTRIG is cleared and bit TRGEN = 1, the last channel is periodically converted together with the other enabled channels and the result is placed in the ADC_LCDR and ADC_CDR7 registers. Thus the last channel conversion result is part of the Peripheral DMA Controller buffer (see Figure 38-8).

When the conversion result matches the conditions defined in the ADC_LCTMR and ADC_LCCWR, the LCCHG flag is set in ADC_ISR.

Figure 38-8. Same Trigger for All Channels (ADC_CHSR[LCI] = 1 and ADC_MR.TRGEN = 1)



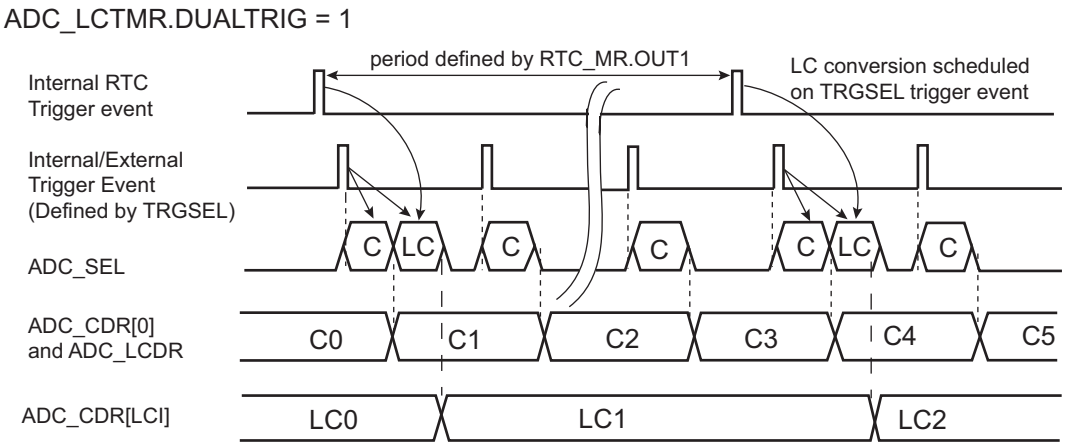
If the last channel is driven by a device with a slower variation compared to other channels (temperature sensor for example), the channel can be enabled/disabled at any time. However, this may not be optimal for downstream processing.

The ADC controller allows a different way of triggering the measurement when DUALTRIG is set in the Last Channel Trigger Mode Register (ADC_LCTMR) but CH7 is not set in ADC_CHSR.

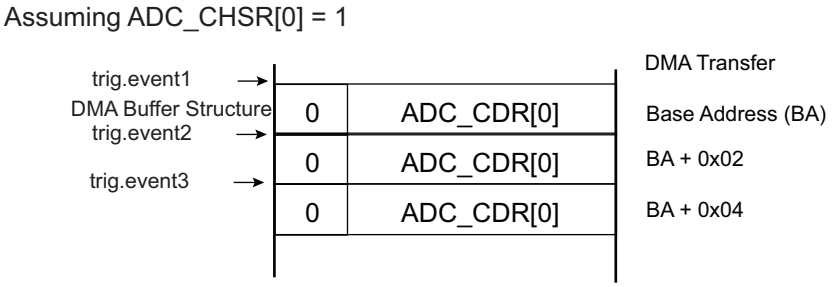
Under these conditions, the last channel conversion is triggered with a period defined by the OUT1 field in the RTC_MR (Real-time Clock Mode Register) while other channels are still active. OUT1 configures an internal trigger generated by the RTC, totally independent of the internal/external triggers. The RTC event will be processed on the next internal/external trigger event as described in Figure 38-9. The internal/external trigger for other channels is selected through the TRGSEL field of ADC_MR.

When DUALTRIG = 1, the result of each conversion of channel 7 is only uploaded in the ADC_CDR7 register and not in ADC_LCDR (see Figure 38-9). 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 the EOC7 flag in ADC_ISR.

Figure 38-9. Independent Trigger Measurement for Last Channel (ADC_CHSR[LCI] = 0 and ADC_MR.TRGEN = 1)



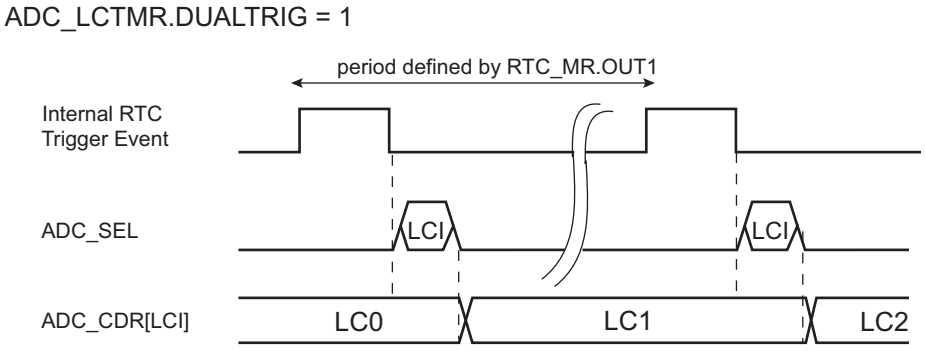
Notes: ADC_SEL: Command to the ADC analog cell
Cx: All ADC channel values except the last channel (highest index)
LCx: Last channel value
LCI: Last channel index



If DUALTRIG = 1 and bit ADC_MR.TRGEN is cleared and none of the channels are enabled in ADC_CHSR (ADC_CHSR = 0), then only channel 7 is converted at a rate defined by the trigger event signal that can be configured in RTC_MR.OUT1 (see Figure 38-10).

This mode of operation, when combined with the Sleep mode operation of the ADC Controller, provides a low-power mode for last channel measure. This assumes there is no other ADC conversion to schedule at a high sampling rate or no other channel to convert.

Figure 38-10. Only Last Channel Measurement Triggered at Low Speed (ADC_CHSR[LCI] = 0 and ADC_MR.TRGEN = 0)



Notes: ADC_SEL: Command to the ADC analog cell
LCx: Last channel value
LCI: Last channel index

38.6.12 Enhanced Resolution Mode and Digital Averaging Function

38.6.12.1 Enhanced Resolution Mode

The Enhanced Resolution mode is enabled if the OSR field is configured to 1, 2, 3 or 4 in ADC_EMR. The enhancement is based on a digital averaging function.

FREERUN in ADC_MR must be cleared when digital averaging is used (OSR ≠ 0 in ADC_EMR).

There is no averaging on the last index channel if the measure is triggered by an RTC event.

In this mode, the ADC Controller will trade off conversion speed against accuracy by averaging multiple samples, thus providing a digital low-pass filter function.

The selected oversampling ratio applies to all enabled channels except for the last channel when triggered by an RTC event.

$$\text{ADC\_LCDR.LDATA} = \frac{1}{M} \times \sum_{k=0}^{k=N-1} ADC(k)$$

where N and M are given in the table below.

Table 38-5. Digital Averaging Function Configuration versus OSR Values

ADC_EMR.OSR Value	ADC_LCDR.LDATA Length	N Value	M Value	Full Scale Value	Maximum Value
0	12 bits	1	1	4095	4095
1	13 bits	4	2	8191	8190
2	14 bits	16	4	16383	16381
3	15 bits	64	8	32767	32761
4	16 bits	256	16	65535	65521

The average result is valid in ADC_CDRx (x corresponds to the index of the channel) only if the EOCn flag is set in ADC_ISR and if the OVREn flag is cleared in ADC_OVER. The average result for all channels is valid in ADC_LCDR only if DRDY is set and GOVRE is cleared in ADC_ISR.

Note that 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 ADC_OVER rise as soon as the first sample of an averaging sequence is received. Thus the previous averaged value is not read, even if the new averaged value is not ready.

Consequently, when an overrun flag rises in 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.

When an oversampling is performed, the maximum value that can be read on ADC_CDRx or ADC_LCDR is not the full-scale value, even if the maximum voltage is supplied on the analog input. Refer to Table 38-5 "Digital Averaging Function Configuration versus OSR Values".

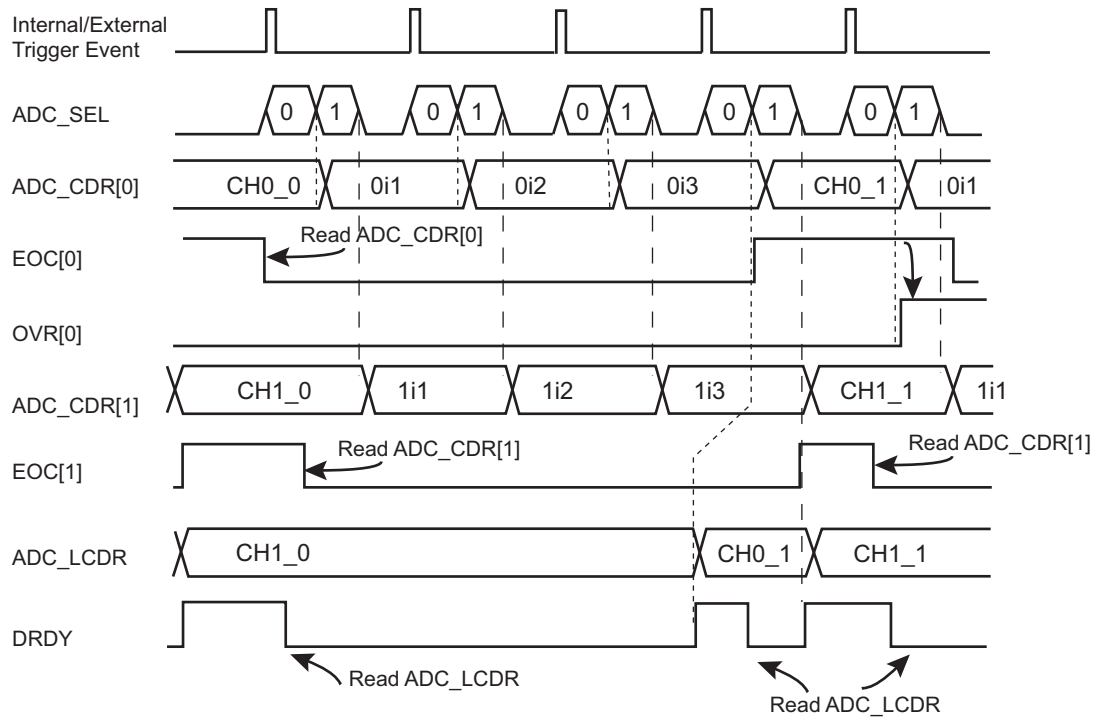
38.6.12.2 Averaging Function versus Trigger Events

The samples can be defined in different ways for the averaging function depending on the configuration of the ASTE bit in ADC_EMR and the USEQ bit in ADC_MR.

When USEQ = 0, there are two possible ways to generate the averaging through the trigger event. If ASTE = 0 in ADC_EMR, every trigger event generates one sample for each enabled channel as described in Figure 38-11. Therefore four trigger events are requested to get the result of averaging if OSR = 1.

Figure 38-11. Digital Averaging Function Waveforms Over Multiple Trigger Events

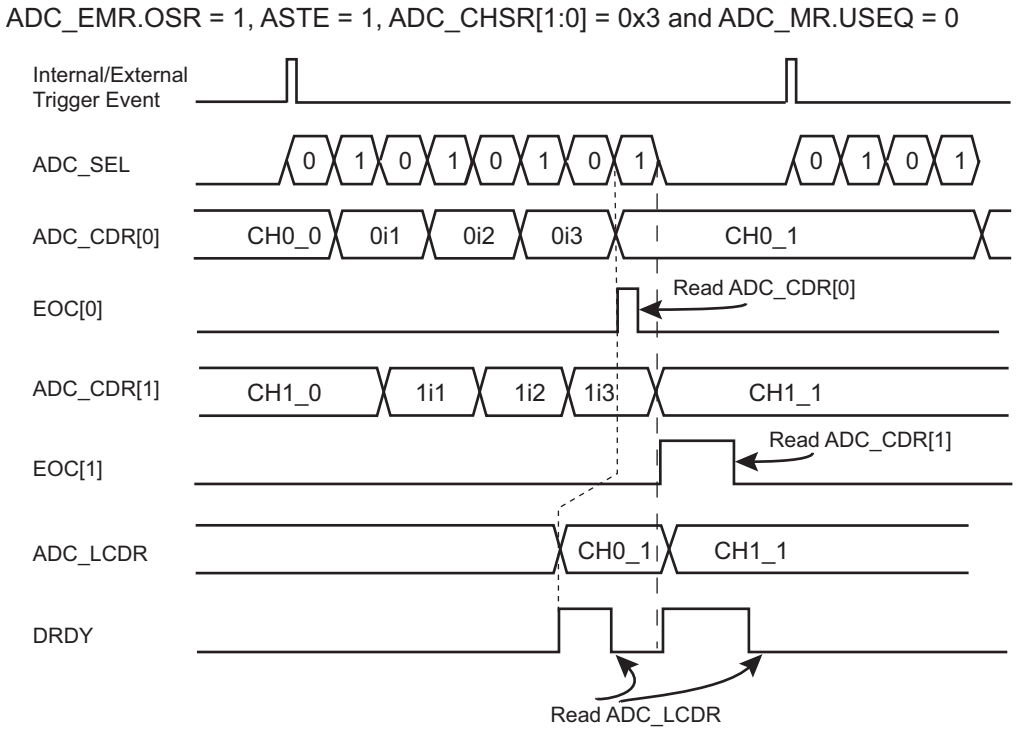
ADC_EMR.OSR = 1, ASTE = 0, ADC_CHSR[1:0] = 0x3 and ADC_MR.USEQ = 0



Notes: ADC_SEL: Command to the ADC analog cell
0i1, 0i2, 0i3, 1i1, 1i2, 1i3 are intermediate results and CH0_0, CH0_1, CH1_0 and CH1_1 are final results of average function.

If ASTE = 1 in ADC_EMR and USEQ = 0 in ADC_MR, the sequence to be converted, defined in ADC_CHSR, is automatically repeated n times (where n corresponds to the oversampling ratio defined in the OSR field in ADC_EMR). As a result, only one trigger is required to obtain the result of the averaging function as described in Figure 38-12.

Figure 38-12. Digital Averaging Function Waveforms on a Single Trigger Event



Note: ADC_SEL: Command to the ADC analog cell
0i1, 0i2, 0i3, 1i1, 1i2, 1i3 are intermediate results and CH0_0, CH0_1, CH1_0 and CH1_1 are final results of average function.

When USEQ = 1, the user can define the channel sequence to be converted by configuring ADC_SEQRx and ADC_CHER so that channels are not interleaved during the averaging period. Under these conditions, a sample is defined for each end of conversion as described in Figure 38-13.

When USEQ = 1 and ASTE = 1, OSR can be only configured to 1. Up to three channels can be converted in this mode. The averaging result will be placed in the corresponding ADC_CDRx and in ADC_LCDR for each trigger event. The ADC real sample rate remains the maximum ADC sample rate divided by 4.

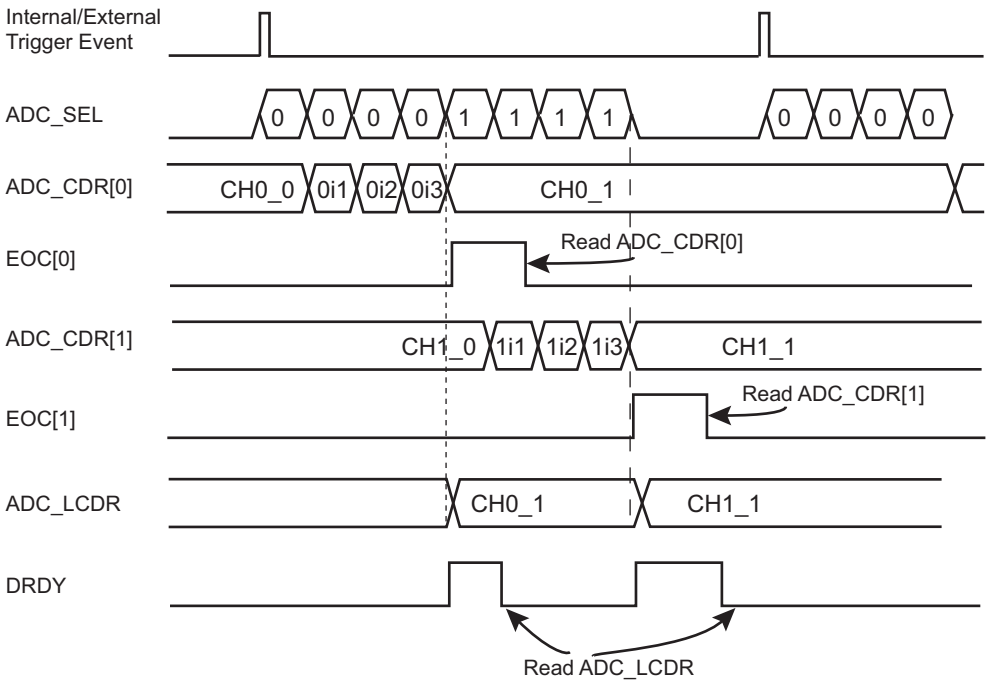
It is important that the user sequence follows a specific pattern. The user sequence must be programmed in such a way that it generates a stream of conversion, where a same channel is successively converted.

Table 38-6. Example Sequence Configurations (USEQ = 1, ASTE = 1, OSR = 1)

	Number of Channels Non-interleaved Averaging - Register Value		
Register	**1 (e.g., CH0)**	**2 (e.g., CH0, CH1)**	**3 (e.g., CH0, CH1, CH2)**
ADC_CHSR	0x0000_000F	0x0000_00FF	0x0000_0FFF
ADC_SEQR1	0x0000_0000	0x1111_0000	0x1111_0000
ADC_SEQR2	0x0000_0000	0x0000_0000	0x0000_2222

Figure 38-13. Digital Averaging Function Waveforms on a Single Trigger Event, Non-interleaved

ADC_EMR.OSR = 1, ASTE = 1, ADC_CHSR[7:0] = 0xFF and ADC_MR.USEQ = 1
ADC_SEQR1 = 0x1111_0000



Note: ADC_SEL: Command to the ADC analog cell
0i1, 0i2, 0i3, 1i1, 1i2, 1i3 are intermediate results and CH0_0, CH0_1, CH1_0 and CH1_1 are final results of average function.

38.6.13 Asynchronous and Partial Wakeup (SleepWalking)

This operating mode is a means of data pre-processing that qualifies an incoming event, thus allowing the ADC to decide whether or not to wake up the system. Asynchronous and partial wakeup is mainly used when the system is in Wait mode (see the PMC section for further details). It can also be enabled when the system is fully running.

Once the Asynchronous and partial wakeup mode is enabled, no access must be performed in the ADC before a Wakeup is performed by the ADC.

When the Asynchronous and partial wakeup mode is enabled for the ADC (see the PMC section), the PMC decodes a clock request from the ADC. The clock request is generated as soon as a trigger event occurs. Only a trigger from RTC or ADTRG pin can be used in partial wakeup mode. The selection between RTC or ADTRG pin is performed through the ADC_MR.TRGSEL field.

If the system is in Wait mode (processor and peripheral clocks switched off), the PMC restarts the fast RC oscillator and provides the clock only to the ADC.

To perform a conversion at regular intervals with RTC trigger, the RTC must be configured with the following settings: RTC_MR.OUT0=7 and RTC_MR.THIGH=7. The period of the trigger can be defined in RTC_MR.TPERIOD.

To trigger a conversion using the ADTRG pin, the minimum high level duration of the ADTRG signal must be greater than 2 clock periods of the fast RC oscillator. The maximum duration of the high level must be limited to the amount of startup and conversion time.

As soon as the clock is provided by the PMC, the ADC processes the conversions and compares the converted values with LOWTHRES and HIGHTHRES field values in ADC_CWR.

The ADC instructs the PMC to disable the clock if the converted value does not meet the conditions defined by LOWTHRES and HIGHTHRES field values in ADC_CWR.

If the converted value meets the conditions, the ADC instructs the PMC to exit the full system from Wait mode.

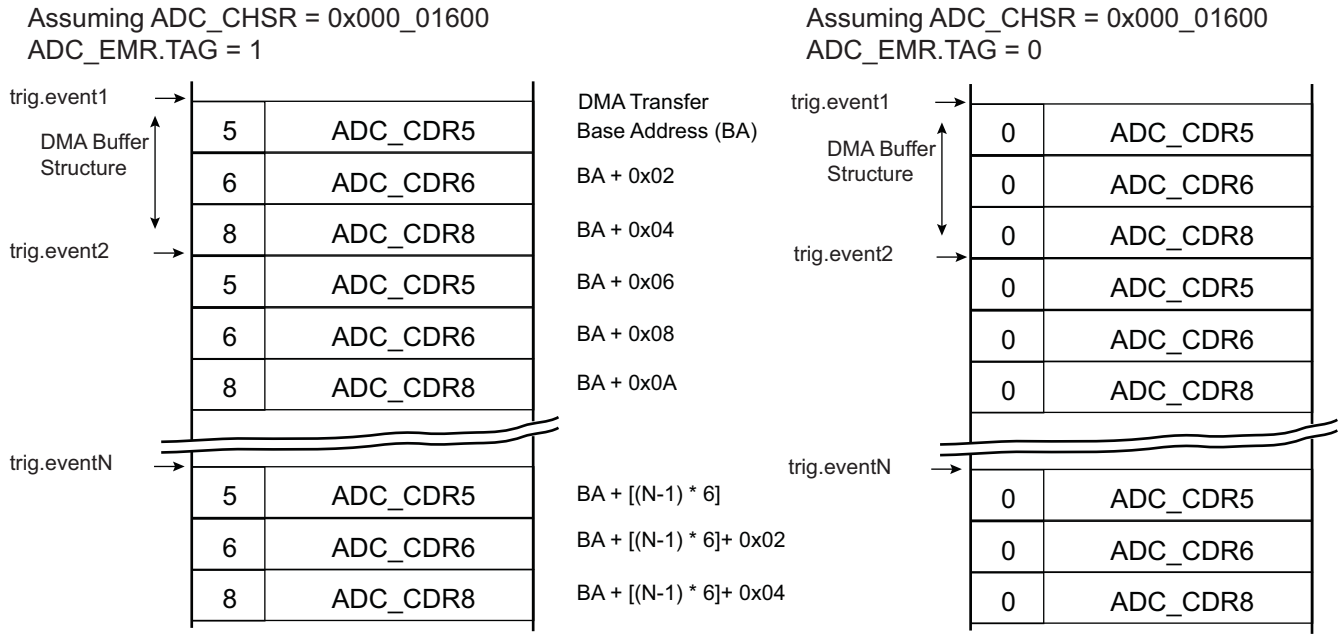
If the processor and peripherals are running, the ADC can be configured in Asynchronous and partial wakeup mode by enabling the PMC_SLPWK_ER (see the PMC section). When a trigger event occurs, the ADC requests the clock from the PMC and the comparison is performed. If there is a comparison match, the ADC continues to request the clock. If there is no match, the clock is switched off for the ADC only, until a new trigger event is detected.

It is recommended to write a '1' to the SLEEP bit to reduce the power consumption of the analog part of the ADC when the system is waiting for a trigger event.

38.6.14 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.

Figure 38-14. Buffer Structure



38.6.15 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 bit WPEN 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 ADC_WPSR.

The following registers are write-protected when WPEN is set in ADC_WPMR:

- 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
- ADC Channel Differential Input Register
- ADC Analog Control Register

38.6.16 Autotest Function

The ADC_ACR.AUTOTEST field configures the autotest modes of the analog cell. When configured for an autotest mode, a conversion must be started on a given channel to observe the gain and offset errors. At the end of the conversion, the value read on ADC_CDR is used to compute the gain and offset errors by comparison with the expected values. There is no automatic correction, and gain and offset errors must be corrected by software. Refer to the section “Electrical Characteristics” for expected values under autotest.

38.7 Analog-to-Digital (ADC) User Interface

Table 38-7. 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
0x48	Reserved	–	–	–
0x4C	Channel Differential Input Register	ADC_COR	Read/Write	0x00000000
0x50	Channel Data Register 0	ADC_CDR0	Read-only	0x00000000
0x54	Channel Data Register 1	ADC_CDR1	Read-only	0x00000000
...	...	...	...	...
0x6C	Channel Data Register 7	ADC_CDR7	Read-only	0x00000000
0x70–0x90	Reserved	–	–	–
0x94	Analog Control Register	ADC_ACR	Read/Write	0x00080000
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–0xFC	Reserved	–	–	–
0x100–0x124	Reserved for PDC registers	–	–	–

Note: Any offset not listed in the table must be considered as “reserved”.

38.7.1 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
–	–	–	CMPRST	–	–	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.

- **CMPRST: Comparison Restart**

0: No effect.

1: Stops the conversion result storage until the next comparison match.

38.7.2 ADC Mode Register

Name: ADC_MR

Address: 0x40038004

Access: Read/Write

31	30	29	28	27	26	25	24
USEQ		TRANSFER		TRACKTIM			
23	22	21	20	19	18	17	16
ANACH	–	SETTLING		STARTUP			
15	14	13	12	11	10	9	8
PRESCAL							
7	6	5	4	3	2	1	0
FREERUN	FWUP	SLEEP	–	TRGSEL			TRGEN

This register can only be written if the WPEN bit is cleared in the 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	ADTRG External trigger
1	ADC_TRIG1	TIOA0 Output of the Timer Counter Channel 0
2	ADC_TRIG2	TIOA1 Output of the Timer Counter Channel 1
3	ADC_TRIG3	TIOA2 Output of the Timer Counter Channel 2
4	ADC_TRIG4	RTCOUT0
5	ADC_TRIG5	RTT 16-Bit prescaler output
6	ADC_TRIG6	RTTEVENT
7	ADC_TRIG7	–

- **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 wakeup time can be modified by programming the FWUP bit.

- **FWUP: Fast WakeUp**

Value	Name	Description
0	OFF	If SLEEP is 1, then both ADC core and reference voltage circuitry are OFF between conversions
1	ON	If SLEEP is 1, then Fast Wakeup Sleep mode: The voltage reference is ON between conversions and ADC core is OFF

- **FREERUN: Free Run Mode**

Value	Name	Description
0	OFF	Normal Mode
1	ON	Free Run Mode: Never wait for any trigger.

Note: FREERUN must be cleared when digital averaging is used (OSR ≠ 0 in ADC_EMR).

- **PRESCAL: Prescaler Rate Selection**

PRESCAL = ($f_{peripheral\ clock}$ / (2 × f_{ADCCLK})) – 1.

- **STARTUP: Startup Time**

Value	Name	Description
0	SUT0	0 periods of ADCCLK
1	SUT8	8 periods of ADCCLK
2	SUT16	16 periods of ADCCLK
3	SUT24	24 periods of ADCCLK
4	SUT64	64 periods of ADCCLK
5	SUT80	80 periods of ADCCLK
6	SUT96	96 periods of ADCCLK
7	SUT112	112 periods of ADCCLK
8	SUT512	512 periods of ADCCLK
9	SUT576	576 periods of ADCCLK
10	SUT640	640 periods of ADCCLK
11	SUT704	704 periods of ADCCLK
12	SUT768	768 periods of ADCCLK
13	SUT832	832 periods of ADCCLK
14	SUT896	896 periods of ADCCLK
15	SUT960	960 periods of ADCCLK

- **SETTLING: Analog Settling Time**

Value	Name	Description
0	AST3	3 periods of ADCCLK
1	AST5	5 periods of ADCCLK
2	AST9	9 periods of ADCCLK
3	AST17	17 periods of ADCCLK

- **ANACH: Analog Change**

Value	Name	Description
0	NONE	No analog change on channel switching: DIFF0, and OFF0 are used for all channels.
1	ALLOWED	Allows different analog settings for each channel. See ADC_COR register.

- **TRACKTIM: Tracking Time**

Value	Name	Description
0	ADCCLK6	The tracking time is 6 ADC clock cycles.
1–14	–	The tracking time is 6 ADC clock cycles.
15	ADCCLK7	The tracking time is 7 ADC clock cycles.

- **TRANSFER: Transfer Time**

  The TRANSFER field must be set to 2 to guarantee the optimal transfer time.

- **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.

38.7.3 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 allowed range is 0 up to 7, thus only the sequencer from CH0 to CH7 can be used.

This register activates only if the USEQ field in ADC_MR field is set to ‘1’.

Any USCHx field is processed only if the CHx-1 it in ADC_CHSR 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.

38.7.4 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 ADC_MR, CHx corresponds to the enable of sequence number x+1 described in ADC_SEQR1 (e.g. CH0 enables sequence number USCH1).

38.7.5 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 and then reenabled during a conversion, its associated data and corresponding EOCx and GOVRE flags in ADC_ISR and OVREx flags in ADC_OVER are unpredictable.

38.7.6 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.

38.7.7 ADC Last Converted Data Register

Name: ADC_LCDR

Address: 0x40038020

Access: Read-only

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

- **LDATA: Last Data Converted**

The analog-to-digital conversion data is placed into this register at the end of a conversion and remains until a new conversion is completed.

If OSR = 0 and TAG = 1 in ADC_EMR, the 4 MSB of LDATA carry the channel number in order to get a packed system memory buffer made of 1 converted data stored in a halfword (16-bit) instead of 1 converted data in a 32-bit word, thus dividing by 2 the size of the memory buffer.

- **CHNBOSR: Channel Number in Oversampling Mode**

Indicates the last converted channel when the TAG bit is set in ADC_EMR and the OSR field is not equal to 0 in ADC_EMR0. If the TAG bit is not set, CHNBOSR = 0.

38.7.8 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
–	–	–	–	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**
- **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**

38.7.9 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
–	–	–	–	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**
- **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**

38.7.10 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
–	–	–	–	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**
- **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**

38.7.11 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
–	–	–	–	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 (automatically set / cleared)**

0: The corresponding analog channel is disabled, or the conversion is not finished. This flag is cleared when reading the corresponding ADC_CDRx registers.

1: The corresponding analog channel is enabled and conversion is complete.

- **LCCHG: Last Channel Change (cleared on read)**

0: There is no comparison match (defined in the Last Channel Compare Window Register (ADC_LCCWR) since the last read of ADC_ISR.

1: The temperature value reported on ADC_CDR7 has changed since the last read of ADC_ISR, according to what is defined in the Last Channel Trigger Mode Register (ADC_LCTMR) and Last Channel Compare Window Register (ADC_LCCWR).

- **DRDY: Data Ready (automatically set / cleared)**

0: No data has been converted since the last read of ADC_LCDR.

1: At least one data has been converted and is available in ADC_LCDR.

- **GOVRE: General Overrun Error (cleared on read)**

0: No general overrun error occurred since the last read of ADC_ISR.

1: At least one general overrun error has occurred since the last read of ADC_ISR.

- **COMPE: Comparison Event (cleared on read)**

0: No comparison event since the last read of ADC_ISR.

1: At least one comparison event (defined in ADC_EMR and ADC_CWR) has occurred since the last read of ADC_ISR.

- **ENDRX: End of Receive Transfer (cleared by writing ADC_RCR or ADC_RNCR)**

0: The Receive Counter Register has not reached 0 since the last write in ADC_RCR or ADC_RNCR[(1)].

1: The Receive Counter Register has reached 0 since the last write in ADC_RCR or ADC_RNCR[(1)].

- **RXBUFF: Receive Buffer Full (cleared by writing ADC_RCR or ADC_RNCR)**

0: ADC_RCR or ADC_RNCR[(1)] has a value other than 0.

1: Both ADC_RCR and ADC_RNCR[(1)] have a value of 0.

Note: 1. ADC_RCR and ADC_RNCR are PDC registers

38.7.12 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 TRGSEL in ADC_MR.

1: Last channel (higher index) trigger period is defined by OUT1 in RTC_MR.

- **CMPMOD: Last Channel Comparison Mode**

Value	Name	Description
0	LOW	Generates the LCCHG flag in ADC_ISR when the converted data is lower than the low threshold of the window.
1	HIGH	Generates the LCCHG flag in ADC_ISR when the converted data is higher than the high threshold of the window.
2	IN	Generates the LCCHG flag in ADC_ISR when the converted data is in the comparison window.
3	OUT	Generates the LCCHG flag in ADC_ISR when the converted data is out of the comparison window.

38.7.13 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 ADC_LCTMR.

- **HIGHTHRES: High Threshold**

High threshold associated to compare settings of ADC_LCTMR.

38.7.14 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: An overrun error has occurred on the corresponding channel since the last read of ADC_OVER.

Note: An overrun error does not always mean that the unread data has been replaced by a new valid data. Refer to Section 38.6.12 “Enhanced Resolution Mode and Digital Averaging Function” for details.

38.7.15 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
–	–	SRCCLK	ASTE	–	OSR		
15	14	13	12	11	10	9	8
–	–	CMPFILTER		–	–	CMPALL	–
7	6	5	4	3	2	1	0
CMPSEL				–	CMPTYPE	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	When the converted data is lower than the low threshold of the window, generates the COMPE flag in ADC_ISR or, in Partial Wakeup mode, defines the conditions to exit system from Wait mode.
1	HIGH	When the converted data is higher than the high threshold of the window, generates the COMPE flag in ADC_ISR or, in Partial Wakeup mode, defines the conditions to exit system from Wait mode.
2	IN	When the converted data is in the comparison window, generates the COMPE flag in ADC_ISR or, in Partial Wakeup mode, defines the conditions to exit system from Wait mode.
3	OUT	When the converted data is out of the comparison window, generates the COMPE flag in ADC_ISR or, in Partial Wakeup mode, defines the conditions to exit system from Wait mode.

- **CMPTYPE: Comparison Type**

Value	Name	Description
0	FLAG_ONLY	Any conversion is performed and comparison function drives the COMPE flag.
1	START_CONDITION	Comparison conditions must be met to start the storage of all conversions until the CMPRST bit is set.

- **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.

Refer to Section 38.6.8 “Comparison Window” when using filtering option (CMPFILTER > 0).

- **OSR: Oversampling 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.
3	OSR64	3-bit enhanced resolution by averaging. ADC sample rate divided by 64.
4	OSR256	4-bit enhanced resolution by averaging. ADC sample rate divided by 256.

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.

- **SRCCLK: External Clock Selection**

0 (PERIPH_CLK): The peripheral clock is the source for the ADC prescaler.

1 (PMC_PCKx): PMC_PCKx is the source clock for the ADC prescaler, thus the ADC clock can be independent of the core/peripheral clock.

- **TAG: Tag of ADC_LCDR**

0: Sets CHNB field to zero in ADC_LCDR.

1: Appends the channel number to the conversion result in ADC_LCDR.

38.7.16 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 ADC_EMR.

- **HIGHTHRES: High Threshold**

High threshold associated to compare settings of ADC_EMR.

38.7.17 ADC Channel Differential Input Register

Name: ADC_COR

Address: 0x4003804C

Access: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
DIFF7	DIFF6	DIFF5	DIFF4	DIFF3	DIFF2	DIFF1	DIFF0
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 ADC Write Protection Mode Register.

- **DIFFx: Differential Inputs for Channel x**

0: Corresponding channel is set in Single-ended mode.

1: Corresponding channel is set in Differential mode.

38.7.18 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.

38.7.19 ADC Analog Control Register

Name: ADC_ACR

Address: 0x40038094

Access: Read/Write

31	30	29	28	27	26	25	24
AUTOTEST		–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
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 ADC Write Protection Mode Register.

- **AUTOTEST: ADC Autotest Modes**

Value	Name	Description
0	NO_AUTOTEST	No auto test, normal mode of operation
1	OFFSET_ ERROR	Offset Error test (see electrical characteristics)
2	GAIN_ERROR_HIGH	Gain Error (high code) test (see electrical characteristics)
3	GAIN_ERROR_LOW	Gain Error (low code) test (see electrical characteristics)

38.7.20 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 38.6.15 “Register Write Protection” for the 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

38.7.21 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 ADC_WPSR.

1: A write protection violation has occurred since the last read of 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.

39. Electrical Characteristics

39.1 Absolute Maximum Ratings

Table 39-1. Absolute Maximum Ratings*

Storage temperature . . . . . -60°C to + 150°C
Voltage on input pins with respect to ground . . . . . -0.3V to + 4V
Absolute maximum voltage (VDDIO) 4V
Total DC output current on all I/O lines:
49-lead WLCSP. . . . . 100 mA
64-lead packages . . . . . 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.**

39.2 Recommended Operating Conditions

Table 39-2. Recommended Operating Conditions

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
T_A	Ambient Temperature Range	—	-40	—	+85	°C
V_{DDCORE}	DC Supply Core	Connected to VDDOUT	—	—	1.32	V
$V_{DDCOREXT100}$ (1)	External DC Supply Core	To run @100 MHz	1.15	—	1.32	V
$V_{DDCOREXT120}$ (1)	External DC Supply Core	To run @120 MHz	1.25	—	1.32	V
$V_{DDCOREXT\_WM}$	External DC Supply Core in Wait mode	Internal regulator in Wait mode	1.2	—	1.32	V
V_{DDIO}	DC Supply I/Os	—	1.62	3.3	3.6	V
V_{DDIO_RIPPLE}	Supply Ripple Voltage	RMS value 10 kHz to 10 MHz	—	—	30	mV
		RMS value > 10 MHz	—	—	10	
	Supply Ripple Voltage when ADC used	RMS value, 10 kHz to 20 MHz	—	—	20	mV
V_{DDIO_SLOPE}	Slope on VDDIO	In Active mode	—	—	1.5	V/ms
f_{MCK}	Master Clock Frequency	—	—	100	120 (2)	MHz

Notes:
1. The V_{DDCORE} range is defined for all conditions (process and temperature). The value may be different than the one used with the internal regulator which is trimmed in production.
2. To run at f_{MCK} = 120 MHz, the Internal regulator needs to be trimmed, using the code read in the Unique Identifier on the Flash. Refer to Section 8.3.1.5 “Unique Identifier”.

39.3 DC Characteristics

The following characteristics are applicable to the operating temperature range: T_A = -40°C to 85°C, unless otherwise specified.

Table 39-3. DC Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IL}	Low-level Input Voltage	PA21–PA22 (1)	—	—	0.8	V
		All others, NRST	-0.3	—	0.3 × V_{DDIO}	
V_{IH}	High-level Input Voltage	PA21–PA22(2)	2.0	—	—	V
		All others, NRST	0.7 × V_{DDIO}	—	V_{DDIO} + 0.3V	
V_{OH}	High-level Output Voltage	PA21–PA22 (1)	V_{DDIO} - 0.4V	—	—	V
		PA3–PA4, PA14 (I_{OH} = 6.0 mA)	V_{DDIO} - 0.4V	—	—	
		All others, Drive Low (I_{OH} = 2.0 mA), NRST	V_{DDIO} - 0.4V	—	—	
		All others, Drive High (I_{OH} = 4.0 mA), NRST	V_{DDIO} - 0.4V	—	—	
V_{OL}	Low-level Output Voltage	PA21–PA22 (1)	—	—	0.4	V
		PA3–PA4, PA14 (I_{OL} = 6.0 mA)	—	—	0.4	
		All others Drive Low (I_{OH} = 2.0 mA), NRST	—	—	0.4	
		All others Drive High (I_{OH} = 4.0 mA), NRST	—	—	—	
V_{HYS}	Hysteresis Voltage	PA21–PA22 (1)	0.2	—	—	mV
		All others, NRST	0.15	—	—	mV
I_{OH}	Source Current	V_{OH} = V_{DDIO} - 0.4				
		PA3–PA4, PA14	—	—	-6	mA
		PA21–PA22 (1)	—	—	-20	
		All others, Drive High	—	—	-4	
		All others, Drive Low	—	—	-2	
		NRST	—	—	-4	mA
I_{OL}	Sink Current	V_{OH} = V_{DDIO} - 0.4				
		PA3–PA4, PA14	—	—	6	mA
		PA21–PA22 (1)	—	—	20	
		All other Drive High	—	—	4	
		All others Drive Low	—	—	2	
		V_{OL} = 0.4V				
		NRST	—	—	4	mA
I_{IL}	Low-level Input Current	Pull-up ON on PA21–PA22	11	33	72	µA
		Pull-up OFF	-1	–	1	µA
		Pull-up ON on all other	10	–	50	

Table 39-3. DC Characteristics (Continued)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
I_{IH}	Input High	Pull-down OFF	-1	–	1	µA
		Pull-down ON on all except PA21–PA22	10	–	50	
R_{PULLUP}	Pull-up Resistor	PA21–PA22 in the following modes: - As PIO - PA22 In Full Speed mode - PA21 In Low Speed Mode	50	100 2 2	150	kΩ
	Pull-up Resistor	All others, NRST	70	100	140	kΩ
$R_{PULLDOWN}$	Pull-down Resistor	PA21–PA22 in the following modes: - As PIO - FS/HS	14.25	20 20	24.8	kΩ
		All others, NRST (2)	70	100	140	kΩ

Notes: 1. Using PA21–PA22 (USB pads) requires an external resistor R_{EXT} = 33 ohms. Voltage required for USB pads: V_{DDIO} min is 3.0V and max is 3.6V.

2. Except PA3–PA4–PA14 (no pull-down on PA3–PA4–PA14).

Table 39-4. VDDCORE Voltage Regulator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{DDIO}	DC Input Voltage Range	—	1.62	3.3	3.60	V
V_{DDOUT}	DC Output Voltage	Running mode @100 MHz(3)	1.08	—	1.32	V
		Running mode @120 MHz(3)	1.2	—	1.32	
		Wait mode	0.85	—	1.32	
$V_{ACCURACY}$	Output Voltage Accuracy	Initial output voltage accuracy	-5.5	—	5.5	%
I_{LOAD}	Maximum DC Output Current	At 1.2V At 0.95V	10 µ	—	60 m	A
V_{LINE}	Line Regulation	V_{DDIO} from 1.62V to 3.6V; I_{LOAD} max	—	—	30	mV
$V_{LINE-TR}$	Transient Line Regulation	V_{DDIO} from 1.62V to 3.6V; $t_R = t_F$ = 1 ms; I_{LOAD} max	—	—	80	mV
V_{LOAD}	Load Regulation	I_{LOAD} = 20% to 80% at I_{LOAD} max	—	—	20	mV
$V_{LOAD-TR}$	Transient Load Regulation	I_{LOAD} = 20% to 80% at max; $t_R = t_F$ = 20 µs	—	—	80	mV
I_Q	Quiescent Current	I_{LOAD} = 0 mA at 2V	—	1.03	1.8	µA
		I_{LOAD} = 0 mA at 3.6V	—	2.1	2.6	
CD_{IN}	Input Decoupling Capacitor	(1)	—	—	—	µF
CD_{OUT}	Output Decoupling Capacitor	(2)	—	1	—	µF
t_{ON}	Turn-on Time for Standby to Normal Mode	—	—	—	600	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 1 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 200 mΩ. Multilayer 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.

3. The voltage range value is given for a regulator that is not trimmed. To have the trimmed value of the regulator, please refer to Unique ID value in Section 8.3.1.5 “Unique Identifier”.

Table 39-5. Core Power Supply Brownout Detector Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{TH-}	Supply Falling Threshold	—	0.71	0.9	1.02	V
V_{HYST}	Hysteresis	—	10	60	110	mV
V_{TH+}	Supply Rising Threshold	—	0.79	1.0	1.08	V
I_{DDON}	Current Consumption on VDDCORE	Brownout detector enabled	—	3.6	6	µA
I_{DDOFF}		Brownout detector disabled	—	—	1	
I_{DDIOON}	Current Consumption on VDDIO	Brownout detector enabled	—	2.9	5	µA
$I_{DDIOOFF}$		Brownout detector disabled	—	—	1	
t_{D-}	V_{TH-} Detection Propagation Time	—	—	1	15	µs
t_{START}	Startup Time	From disabled state to enabled state	—	—	600	µs

Figure 39-1. Core Brownout Output Waveform

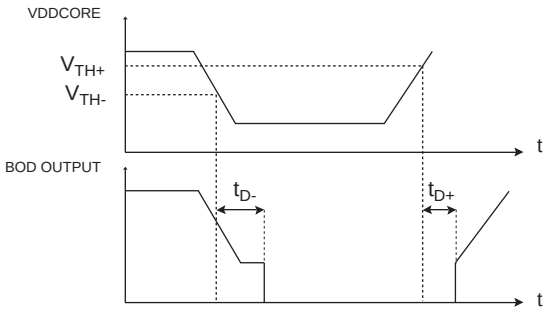


Table 39-6. VDDIO Supply Monitor

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{TH}	Supply Monitor Threshold	16 selectable steps	1.6	—	3.4	V
$T_{ACCURACY}$	Threshold Level Accuracy	-40°C to +85°C	-2.5	—	2.5	%
V_{HYST}	Hysteresis	—	—	20	30	mV
I_{DD}	Current Consumption	Normal mode, -40°C to +85°C	—	—	10	µA
		Standby mode, -40°C to +85°C	—	—	1	
t_{START}	Startup Time	From disabled state to enabled state	—	—	600	µs

The threshold is selected through the SUPC_SMMR in the Supply Controller.

Table 39-7. Threshold Selection

Digital Code	Threshold min (V)	Threshold typ (V)	Threshold max (V)
0000	—	1.6	—
0001	—	1.72	—
0010	—	1.84	—
0011	—	1.96	—
0100	—	2.08	—
0101	—	2.20	—
0110	—	2.33	—
0111	—	2.45	—
1000	—	2.57	—
1001	—	2.69	—
1010	—	2.81	—
1011	—	2.93	—
1100	—	3.05	—
1101	—	3.17	—
1110	—	3.29	—
1111	—	3.41	—

Figure 39-2. VDDIO Supply Monitor

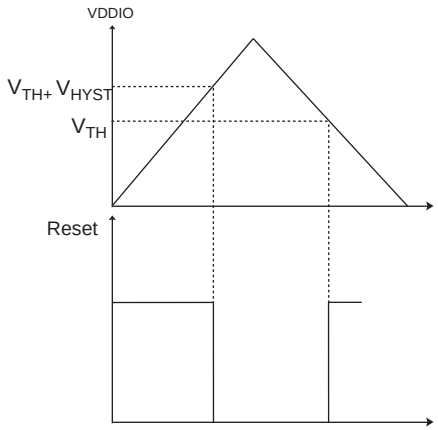
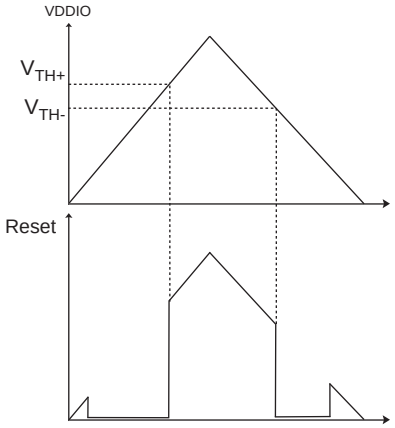


Table 39-8. Power-on Reset Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{TH+}	Threshold Voltage Rising	At startup	1.49	1.54	1.595	V
V_{TH-}	Threshold Voltage Falling	—	1.39	1.46	1.53	V
V_{HYST}	Hysteresis	—	50	80	130	mV
t_{RES}	Reset Timeout Period	—	120	320	800	µs
I_{POR}	Current Consumption	—	—	300	499	nA

Figure 39-3. Power-on Reset Characteristics



39.4 Power Consumption

This section provides information on:

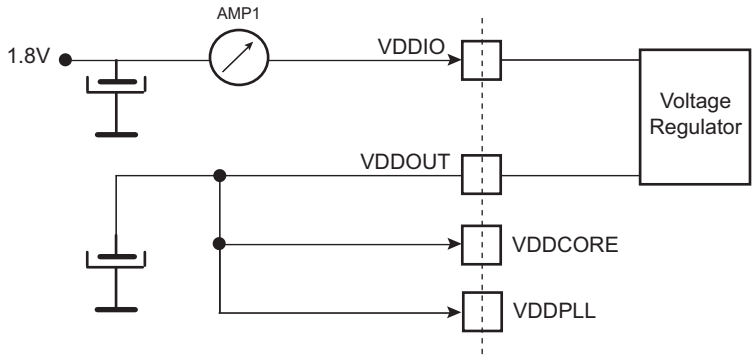
- Power consumption of the device in Low-power mode (Wait mode), Sleep mode and Active mode.
- Power consumption by peripheral: calculated as the difference in current measurement after enabling then disabling the corresponding clock.

39.4.1 Backup Mode Current Consumption

The Backup mode configuration and measurements are defined as follows.

The Backup mode values are the same in Single supply or Dual supply.

Figure 39-4. Backup Mode Measurement Setup



39.4.1.1 Configuration A: Embedded Slow Clock RC Oscillator Enabled

- Single supply
- Supply Monitor on VDDIO is disabled (SUPC_SMMR.SMSMPL = 0)
- RTC is running
- RTT is enabled on 1Hz mode
- BOD disabled
- One WKUPx enabled
- Current measurement on AMP1 (see Figure 39-4)

Table 39-9. Typical Power Consumption for Backup Mode

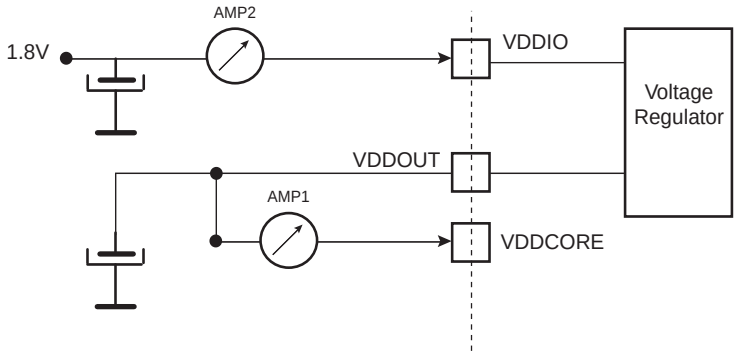
Backup Total Consumption	@25°C	
Conditions	(AMP1) Configuration A	Unit
VDDIO = 3.6V	2.6	µA
VDDIO = 3.3V	2.1	µA
VDDIO = 3.0V	1.8	µA
VDDIO = 2.5V	1.5	µA
VDDIO = 2.0V	1.3	µA
VDDIO = 1.8V	1.12	µA
VDDIO = 1.6V	1.08	µA

39.4.2 Wait Mode Power Consumption

Wait mode configuration and measurements are defined below.

39.4.2.1 Wait Mode Single Supply

Figure 39-5. Wait Mode Measurement Setup (Single Supply)



- 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
- RAM can be powered by block of 8/16/32 Kbytes in Wait mode
- BOD disabled
- V_{DDIO} = 1.8V (Single supply)
- V_{DDCORE} = internal voltage regulator used with default settings
- Temperature = 25°C

Table 39-10 provides current consumption in Wait mode under typical conditions.

Table 39-10. Typical Current Consumption in Wait Mode (Single Supply)

Wait Mode Consumption Refer to Figure 39-5		Typical Value V_{DDIO} = 1.8V		
RAM Powered	**Flash Conditions**	**VDDOUT Consumption (AMP1)**	**Total Consumption (AMP2)**	**Unit**
All RAM + 16 Kbytes SRAM I/D Cache	In Deep-power-down mode	20.7	22.0	µA
All RAM powered		17.5	19.6	
8 Kbytes powered		10	12.2	

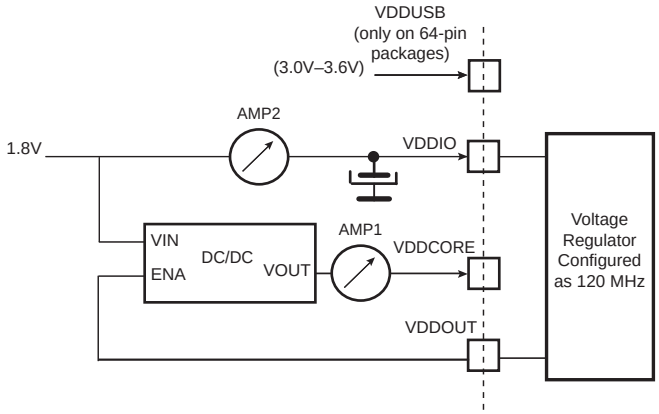
Table 39-11 provides timing wakeup from RAM in Wait mode under typical conditions.

Table 39-11. Typical Wakeup Timing in Wait Mode

Wait Mode Wakeup Time Refer to Figure 39-5			
Fast RC Oscillator (MHz)	**Flash Conditions**	**Wakeup Time**	**Unit**
24 MHz trimmed	All RAM	4.9	µs
16 MHz trimmed		6.5	
8 MHz		13	

39.4.2.2 Wait Mode Dual Supply

Figure 39-6. Wait Mode Measurement Setup (Dual Supply)



- 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
- RAM can be powered by block of 8/16/32 Kbytes in Wait mode
- BOD disabled
- V_{DDIO} = 1.8V (Dual supply)
- V_{DDCORE} = $V_{DDCOREEXT120}$
- Temperature = 25°C

Table 39-10 provides current consumption in Wait mode under typical conditions.

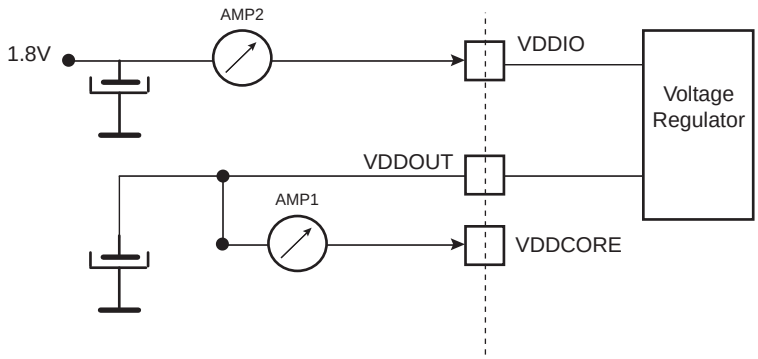
Table 39-12. Typical Current Consumption in Wait Mode (Dual Supply)

Wait Mode Consumption Refer to Figure 39-5		Typical Value V_{DDIO} = 1.8V		
RAM Powered	**Flash Conditions**	**VDDOUT Consumption (AMP1)**	**Total Consumption (AMP2)**	**Unit**
All RAM powered	In Deep-power-down mode	44	48	µA

39.4.3 Sleep Mode Power Consumption

Sleep mode configuration and measurements are defined below.

Figure 39-7. Measurement Setup for Sleep Mode



- Core clock off
- V_{DDIO} = 1.8V or 3.3V (single supply)
- Master clock (MCK) running at various frequencies with PLLA or the fast RC oscillator
- Fast startup through pins WKUP0–15
- All peripheral clocks deactivated
- Temperature = 25°C

Table 39-13 provides current consumption in Sleep mode under typical conditions.

Table 39-13. Typical Current Consumption in Sleep Mode versus Master Clock (MCK) Variation with PLLA or Fast RC

Sleep Mode Consumption		Typical Value V_{DDIO} = 1.8V	Typical Value V_{DDIO} = 3.3V	
MCK (MHz)	PLLA or Fast RC	Total Consumption (AMP2)	Total Consumption (AMP2)	Unit
120	PLL	8.5	8.6	mA
100	PLL	7.2	7.2	
96	PLL	6.9	6.9	
48	PLL	3.6	3.6	
24	PLL	2.0	2.0	
24	Fast RC	1.8	1.8	
16	Fast RC	1.2	1.2	
8	Fast RC	0.7	0.7	

39.4.4 Active Mode Power Consumption

Active mode power configuration and measurements are as follows for all tables:

- Temperature = 25°C
- Fibonacci or CoreMark algorithm runs from Flash memory with 128-bit access mode or from SRAM
- All peripheral clocks are deactivated
- Peripheral clocks are divided by 8
- PLLB clock source is PCK (PMC_USB.USBS = 1)
- Peripheral clock PCK3 is used by timers 0 and 1
- All interrupts, MPU and FPU are disabled
- Flash is OFF (PMC_FSMR.FFLPM = 1 and PMC_FSMR.FLPM = 0x1)
- Master Clock (MCK) runs at various frequencies with PLLA or the fast RC oscillator
- Current measurement on AMP1 (V_{DDCORE}) and total current on AMP2

Figure 39-8. Active Mode Measurement Setup

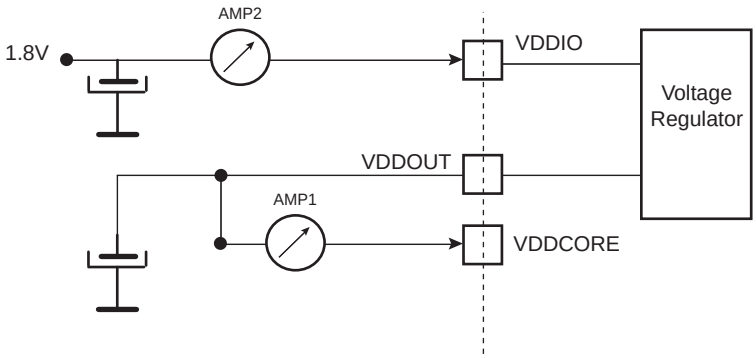


Table 39-14 and Table 39-16 provide active mode current consumption under typical conditions running the Fibonacci algorithm.

Table 39-15 and Table 39-17 provide active mode current consumption under typical conditions running the CoreMark algorithm.

Active mode power configuration and measurements are as follows for Table 39-14 and Table 39-15:

- V_{DDIO} = 3.3V (single supply)
- V_{DDCORE} = value of internal voltage regulator with settings @120 MHz

Table 39-14. Typical Current Consumption Running at V_{DDIO} = 3.3V

MCK (MHz)	PLL or Fast RC	Wait State	Fibonacci Algorithm			Unit
			128-bit Flash Access		SRAM	
			AMP1	AMP2	AMP2	
120	PLL	5	14.9	15.0	13.9	mA
100	PLL	4	12.5	12.6	11.6	
48	PLL	2	5.9	6.0	5.5	
24	PLL	1	3.0	3.0	2.8	
24	Fast RC	1	3.0	3.0	2.8	
16	Fast RC	0	2.1	2.1	1.9	
8	Fast RC	0	1.1	1.1	1.0	

Table 39-15. Typical Active Current Consumption at V_{DDIO} = 3.3V

MCK (MHz)	PLL or Fast RC	Wait State	CoreMark Algorithm			Unit
			128-bit Flash Access Cache Disabled	128-bit Flash Access Cache Enabled	SRAM	
			AMP2	AMP2	AMP2	
120	PLL	5	24.2	23.6	18.2	mA
100	PLL	4	21.0	19.8	15.3	
48	PLL	2	11.5	9.6	7.3	
24	PLL	1	6.1	4.9	3.7	
24	Fast RC	1	5.9	4.9	3.7	
16	Fast RC	0	3.9	3.3	2.5	
8	Fast RC	0	2.2	1.7	1.4	

Active mode power configuration and measurements are as follows for Table 39-16 and Table 39-17:

- V_{DDIO} = 1.8V (single supply)
- V_{DDCORE} = value of internal voltage regulator with settings @120 MHz

Table 39-16. Typical Current Consumption Running at V_{DDIO} = 1.8V

MCK (MHz)	PLL or Fast RC	Wait State	Fibonacci Algorithm			Unit
			128-bit Flash Access		SRAM	
			AMP1	AMP2	AMP2	
120	PLL	5	14.7	14.8	13.7	mA
100	PLL	4	12.3	12.4	11.5	
48	PLL	2	5.9	6.0	5.5	
24	PLL	1	3.0	3.0	2.8	
24	Fast RC	1	3.0	3.0	2.8	
16	Fast RC	0	2.1	2.1	1.9	
8	Fast RC	0	1.1	1.1	1.0	

Table 39-17. Typical Active Current Consumption at V_{DDIO} = 1.8V

MCK (MHz)	PLL or Fast RC	Wait State	CoreMark Algorithm			Unit
			128-bit Flash Access Cache Disabled	128-bit Flash Access Cache Enabled	SRAM	
			AMP2	AMP2	AMP2	
120	PLL	5	24.1	23.4	18.1	mA
100	PLL	4	20.9	19.6	15.2	
48	PLL	2	11.4	9.5	7.3	
24	PLL	1	6.1	4.9	3.6	
24	Fast RC	1	5.9	4.8	3.5	
16	Fast RC	0	3.9	3.2	2.5	
8	Fast RC	0	2.1	1.7	1.3	

39.4.5 Peripheral Power Consumption in Active Mode

The peripheral consumption configuration and measurements are as follows:

- V_{DDIO} = 1.8V or 3.0V
- V_{DDCORE} = value of internal voltage regulator with default settings
- Temperature = 25°C
- Frequency = 120 MHz

Table 39-18. Power Consumption on VDDCORE

Peripheral	Consumption (Typ) at 1.8V	Consumption (Typ) at 3.0V	Unit
PIO Controller A (PIOA)	2.6	2.7	µA/MHz
PIO Controller B (PIOB)	1.0	1.1	
UHP	7.2	7.3	
UDP	1.8	1.9	
CRCCU	0.01	0.01	
Timer Counter (TC0_CH0)	1.6	1.7	
Timer Counter (TC0_CH1)	0.9	0.9	
Timer Counter (TC0_CH2)	0.9	0.9	
Timer Counter (TC1_CH3)	1.4	1.5	
Timer Counter (TC1_CH4)	0.9	0.9	
Timer Counter (TC1_CH5)	0.9	0.9	
ADC	2.4	2.5	
PDM0	10.7	10.8	
PDM1	11.7	11.8	
MEM2MEM	0.67	0.73	
I2SC0	2.6	2.7	
I2SC1	2.4	2.5	
FLEXCOM0 in USART0	2.4	2.5	
FLEXCOM1 in USART1	2.4	2.5	
FLEXCOM2 in USART2	2.6	2.8	
FLEXCOM3 in TWI3	3.2	3.2	
FLEXCOM4 in TWI4	2.6	2.7	
FLEXCOM5 in SPI5	1.6	1.7	
FLEXCOM6 in TWI6	2.6	2.7	
FLEXCOM7 in SPI7	1.5	1.6	

39.5 Oscillator Characteristics

39.5.1 32 kHz RC Oscillator Characteristics

Table 39-19. 32 kHz RC Oscillator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
—	RC Oscillator Frequency	—	20	32	44	kHz
—	Frequency Supply Dependency	Typical at VDDIO = 3V	-3	—	3	%/V
—	Frequency Temperature Dependency	[-40°C : +85°C] versus 25°C	-10	—	10	%
Duty	Duty Cycle	—	45	50	55	%
t_{ON}	Startup Time	—	—	—	100	µs
I_{DDON}	Current Consumption	After startup time	—	540	860	nA
$I_{DDSTDBY}$	Standby Current Consumption	After startup time	—	10	150	nA

39.5.2 8/16/24 MHz RC Oscillators Characteristics

Table 39-20. 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	ACC accuracy at 25°C	16 MHz or 24 MHz output selected (2)	-1.5		1.5	%
ACC_8	8 MHz Total Accuracy	8 MHz output selected (1)(3)	-30	—	30	%
ACC_{16}	16 MHz Temperature Accuracy	16 MHz output selected (1)(4)	-11	—	3	%
		16 MHz output selected (1)(5)	-6	—	3	
		16 MHz output selected (1)(6)	-3.5	—	3	
ACC_{24}	24 MHz Temperature Accuracy	24 MHz output selected (1)(4)	-11	—	3	%
		24 MHz output selected (1)(5)	-6	—	3	
		24 MHz output selected (1)(6)	-3.5	—	3	
Duty	Duty Cycle	—	45	50	55	%
t_{ON}	Startup Time	Time after MOSCRCEN is set(7) when 90% of frequency is reached.	3	—	5.5	µs
t_{ONTRIM}	Stabilizing Time	—	—	5	—	µs
I_{DDON}	Active Current Consumption(4)	8 MHz	—	95	150	µA
		16 MHz	—	130	200	
		24 MHz	—	180	250	

Notes:
1. The frequency range can be configured in the Supply Controller registers.
2. Only if VDDCORE is connected to VDDOUT.
3. Not trimmed from factory.
4. After trimming from factory; for Wait mode and [-40°C : +85°C].
5. After trimming from factory; for Functional mode and [-40°C : +85°C]
6. After trimming from factory; for typical V_{DDCORE} (internal regulator) and [-40°C : +85°C]
7. With V_{DDCORE} @1.2V and T_A = 25°C. For more information on bit MOSCRCEN, refer to the PMC Clock Generator Main Oscillator register (CKGR_MOR).

The Total Accuracy of the RC 16/24 MHz is: ACC accuracy at 25°C + ACC temperature accuracy.

Example: Total Accuracy of RC 16 MHz : ACC_T = ACC+ ACC_{16} (with VDDCORE = internal regulator VDDOUT)
ACC_T at VDDCore typ. = -5%/+4.5% .

The 16/24 MHz fast RC oscillator is calibrated in production. This calibration can be read through the Get CALIB Bit command (see Section 23. “Enhanced Embedded Flash Controller (EEFC)”). The frequency can be trimmed by software through the Power Management Controller (PMC).

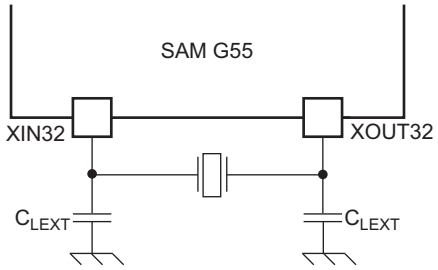
39.5.3 32.768 kHz Crystal Oscillator Characteristics

Table 39-21. 32.768 kHz Crystal Oscillator Characteristics

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
f_{FREQ}	Operating Frequency	Normal mode with crystal		25	32.768	40	kHz
Duty	Duty Cycle	—		40	50	60	%
t_{ON}	Startup Time	R_S(1) < 50 kΩ	$C_{CRYSTAL}$ = 12.5 pF	—	—	900	ms
			$C_{CRYSTAL}$ = 6 pF	—	—	300	
		R_S(1) < 100 kΩ	$C_{CRYSTAL}$ = 12.5 pF	—	—	1200	
			$C_{CRYSTAL}$ = 6 pF	—	—	500	
I_{DDON}	Current Consumption Without Trimming	R_S(1) < 50 kΩ	$C_{CRYSTAL}$ = 12.5 pF	—	—	850	nA
			$C_{CRYSTAL}$ = 6 pF	—	—	800	
		R_S(1) < 100 kΩ	$C_{CRYSTAL}$ = 12.5 pF	—	—	1100	
			$C_{CRYSTAL}$ = 6 pF	—	—	1200	
	Current Consumption After Trimming	R_S(1) < 50 kΩ	$C_{CRYSTAL}$ = 12.5 pF	—	—	680	nA
			$C_{CRYSTAL}$ = 6 pF	—	—	650	
		R_S(1) < 100 kΩ	$C_{CRYSTAL}$ = 12.5 pF	—	—	650	
			$C_{CRYSTAL}$ = 6 pF	—	—	750	
P_{ON}	Drive Level	—		—	—	0.2	µW
R_F	Internal Resistor	Between XIN32 and XOUT32		—	10	—	MΩ
C_{LEXT}	Maximum External Capacitor on XIN32 and XOUT32	—		—	—	24	pF
C_{PARA}	Internal Parasitic Capacitance	—		0.4	0.5	0.6	pF

Note: 1. R_S is the series resistor.

Figure 39-9. 32.268 kHz Crystal Oscillator Schematic



$C_{LEXT} = 2 \times (C_{CRYSTAL} - C_{PARA} - C_{PCB})$.

where C_{PCB} is the capacitance of the printed circuit board (PCB) track layout from the crystal to the SAM G55 pin.

39.5.4 32.768 kHz Crystal Characteristics

Table 39-22. Crystal Characteristics

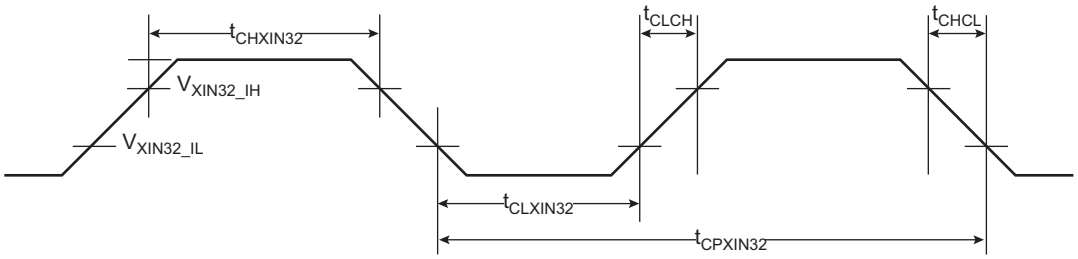
Symbol	Parameter	Conditions	Min	Typ	Max	Unit
ESR	Equivalent Series Resistor (R_S)	Crystal at 32.768 kHz	—	50	100	kΩ
C_m	Motional Capacitance	Crystal at 32.768 kHz	0.6	—	3	fF
C_{SHUNT}	Shunt Capacitance	Crystal at 32.768 kHz	0.6	—	2	pF
C_{LOAD}	Load Capacitance	Crystal at 32.768 kHz Max external capacitor 20 pF	6	—	12.5	pF

39.5.5 32.768 kHz XIN32 Clock Input Characteristics in Bypass Mode

Table 39-23. XIN32 Clock Electrical Characteristics (In Bypass Mode)

Symbol	Parameter	Conditions	Min	Max	Unit
$1/(t_{CPXIN32})$	XIN32 Clock Frequency	32.768 kHz crystal oscillator in Bypass mode (i.e., when SUPC_MR.OSCBYPASS = 1 and SUPC_CR.XTALSEL = 1)		44	kHz
$t_{CPXIN32}$	XIN32 Clock Period		22		µs
$t_{CHXIN32}$	XIN32 Clock High Half-period		11		µs
$t_{CLXIN32}$	XIN32 Clock Low Half-period		11		µs
t_{CLCH}	Rise Time		400		ns
t_{CHCL}	Fall Time		400		ns
C_i	XIN32 Input Capacitance			6	pF
R_{IN}	XIN32 Pull-down Resistor		3	5	MΩ
V_{XIN32_IL}	V_{XIN32} Input Low-level Voltage		-0.3	$0.3 \times V_{DDIO}$	V
V_{XIN32_IH}	V_{XIN32} Input High-level Voltage		$0.7 \times V_{DDIO}$	V_{DDIO} + 0.3	V

Figure 39-10. XIN32 Clock Timing



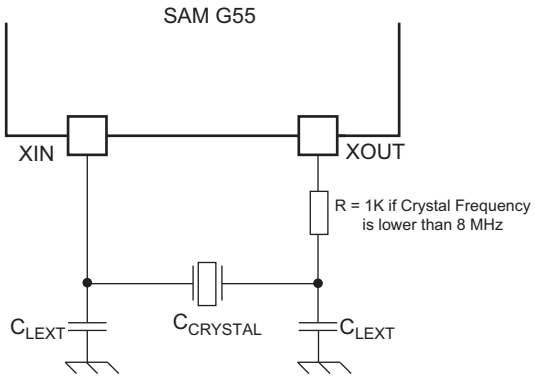
39.5.6 3 to 20 MHz Crystal Oscillator Characteristics

Table 39-24. 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
V_{DDIO}	Supply Voltage	—	1.62	3.3	3.60	V
V_{DDCORE}	Supply Voltage	Supplied by voltage regulator or $V_{DDCOREXT100}$ or $V_{DDCOREXT120}$	—	—	1.32	V
Duty	Duty Cycle	—	40	50	60	%
t_{ON}	Startup Time	3 MHz, C_{SHUNT} = 3 pF	—	—	14.5	ms
		8 MHz, C_{SHUNT} = 7 pF with C_m = 3 fF	—	—	5	
		12MHz, C_{SHUNT} = 7 pF with C_m = 3 fF	—	—	3.8	
		16 MHz, C_{SHUNT} = 7 pF with C_m = 8 fF	—	—	1.4	
		16 MHz, C_{SHUNT} = 7 pF with C_m = 1.6 fF	—	—	4.5	
		20 MHz, C_{SHUNT} = 7 pF	—	—	4.5	
I_{DD_ON}	Current Consumption (on VDDIO)	3 MHz[2]	—	230	350	µA
		8 MHz[3]	—	300	400	
		12 MHz	—	350	470	
		16 MHz[4]	—	390	470	
		20 MHz[5]	—	450	560	
	Current Consumption (on VDDCORE)	3 MHz[2]	—	6	7	µA
		8 MHz[3]	—	12	14	
		12 MHz	—	20	23	
		16 MHz[4]	—	20	23	
		20 MHz[5]	—	24	30	
P_{ON}	Drive Level	3 MHz	—	—	15	µW
		8 MHz	—	—	30	
		12 MHz, 16 MHz, 20 MHz	—	—	50	
R_f	Internal Resistor	Between XIN and XOUT	—	0.5	—	MΩ
$C_{INTLOAD}$	Internal Load Capacitance	Integrated load capacitance ((XIN)(GND) and (XOUT)(GND) in series)	7.5	9	10.5	pF
$C_{Crystal}$	Maximum External Capacitor on XIN and XOUT	Integrated load capacitance (XIN and XOUT in series)	12.5	—	17.5	pF
$C_{PARABYPASS}$	Internal Parasitic During Bypass	On XIN	—	5.5	6.3	pF
		On XOUT	—	2.9	3.3	
$R_{PARASTANDBY}$	Internal Impedance During Standby	MOSCXTEN = 0[6]	—	300	—	Ω

Notes:
1. R_S is the series resistor.
2. R_S = 100–200 Ω; C_{SHUNT} = 2.0–2.5 pF; C_m = 2 fF typ, 1.5 fF worst case, using 1 kΩ serial resistor on XOUT.
3. R_S = 50–100 Ω; C_{SHUNT} = 2.0–2.5 pF; C_m = 4 fF typ, 3 fF worst case.
4. R_S = 25–50 Ω; C_{SHUNT} = 2.5–3.0 pF; C_m = 7 fF typ, 5 fF worst case.
5. R_S = 20–50 Ω; C_{SHUNT} = 3.2–4.0 pF; C_m = 10 fF typ, 8 fF worst case.
6. For more information on the bit MOSCXTEN, refer to the PMC Clock Generator Main Oscillator Register (CKGR_MOR).

Figure 39-11. 3 to 20 MHz Crystal Oscillator Schematic



$C_{LEXT} = 2 \times (C_{CRYSTAL} - C_{INTLOAD} - C_{PCB})$

Where C_{PCB} is the capacitance of the printed circuit board (PCB) track layout from the crystal to the SAM G55 pin.

39.5.7 3 to 20 MHz Crystal Characteristics

Table 39-25. Crystal Characteristics

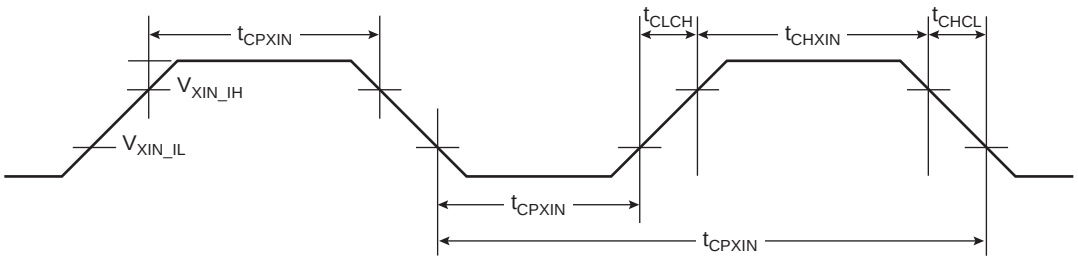
Symbol	Parameter	Conditions	Min	Typ	Max	Unit
ESR	Equivalent Series Resistor (R_S)	Fundamental at 3 MHz	—	—	200	W
		Fundamental at 8 MHz	—	—	100	
		Fundamental at 16 MHz	—	—	80	
		Fundamental at 20 MHz	—	—	50	
C_m	Motional Capacitance	—	3	—	8	fF
C_{SHUNT}	Shunt Capacitance	—	—	—	7	pF
C_{LOAD}	Load Capacitance	Max external capacitors: 17 pF	12.5	—	17.5	pF

39.5.8 3 to 20 MHz XIN Clock Input Characteristics in Bypass Mode

Table 39-26. XIN Clock Electrical Characteristics (In Bypass Mode)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$1/(t_{CPXIN})$	XIN Clock Frequency	3–20 MHz crystal oscillator is in Bypass mode	—	—	50	MHz
t_{CPXIN}	XIN Clock Period		20	—	—	ns
t_{CHXIN}	XIN Clock High Half-period		8	—	—	ns
t_{CLXIN}	XIN Clock Low Half-period		8	—	—	ns
t_{CLCH}	Rise Time		2.2	—	—	ns
t_{CHCL}	Fall Time		2.2	—	—	ns
V_{XIN_IL}	XIN Low-level Input Voltage		-0.3	—	Min [0.8V, 0.3 × V_{DDIO}]	V
V_{XIN_IH}	XIN High-level Input Voltage		Min [2.0V, 0.7 × V_{DDIO}]	—	V_{DDIO} + 0.3V	V

Figure 39-12. XIN Clock Timing



39.5.9 Crystal Oscillator Design Considerations

39.5.9.1 Choosing a Crystal

When choosing a crystal for the 32.768 kHz slow clock oscillator or for the 3–20 MHz oscillator, several parameters must be taken into account. Important parameters between crystal and SAM G55 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 selected according to the internal load capacitance (C_L) of the on-chip oscillator. A mismatch of the load capacitance will result in a frequency drift.
- Drive Level
  - Crystal drive level ≥ Oscillator drive level. A crystal drive level lower than the oscillator specification may damage the crystal.
- Equivalent Series Resistor (ESR)
  - Crystal ESR ≤ Oscillator ESR Max. A crystal with ESR value higher than that of 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 that of the oscillator may cause the oscillator to not start.

39.6 PLLA Characteristics

Table 39-27. PLLA Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{IN}	Input Frequency	—	20	—	300	kHz
f_{OUT}	Output Frequency	—	48	—	120	MHz
I_{PLLON}	Current Consumption	PLL is in Active mode	—	—	14	µA/MHz
I_{PLLOFF}	Current Consumption	PLL is in Standby mode	—	—	500	nA
t_{LOCK}	Lock PLL after Startup Time	Lock PLL	—	—	1.5	ms

39.7 PLLB Characteristics

Table 39-28. PLLB Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{IN}	Input Frequency	—	20	—	100	kHz
f_{OUT}	Output Frequency	—	24	—	48	MHz
I_{PLLON}	Current Consumption	PLL is in Active mode	—	—	12	µA/MHz
I_{PLLOFF}	Current Consumption	PLL is in Standby mode @ 25°C	—	—	500	nA
		PLL is in Standby mode	—	—	5	µA
t_{LOCK}	Lock PLL after Startup Time	Lock PLL	—	—	1.5	ms

39.8 12-bit ADC Characteristics

Table 39-29. Analog Power Supply Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{DDIO}	ADC Analog Supply	—	1.62	1.8	3.60	V
V_{DDCORE}	ADC Digital Supply	Internal voltage regulator or $V_{DDCOREXT100}$ or $V_{DDCOREXT120}$	—	—	1.32	
I_{VDDIO}	Current Consumption	Sleep mode (Clock off)	—	2	4	µA
		Normal mode	—	1.5	2.0	mA
$I_{VDDCORE}$	Current Consumption	Sleep mode (Clock off)	—	1	2	µA
		Normal mode	—	40	80	

Table 39-30. Channel Conversion Time and ADC Clock

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{ADC}	ADC Clock Frequency	No missing code	0.1	—	10	MHz
t_{CP_ADC}	ADC Clock Period	—	100	—	10000	ns
t_{CONV}	ADC Conversion Time	—	1	—	—	t_{cp_ADC}
f_{SAMPLE}	Sampling Rate	12-bit mode	—	—	500	ksps
t_{START}	ADC Startup Time	From OFF to Normal mode	—	—	4	µs
—	ADC Inherent Conversion Time (from start of conversion to end of conversion)	—		20		ADCCLCK cycles

39.8.1 ADC Resolution

39.8.1.1 Differential Mode Conditions

- Temperature = 25°C
- f_{ADC} = 10 MHz
- OSR: Number of averaged samples
- Differential Mode
- V_{DDIO} = 1.8V or 3.0V
- V_{DDCORE} = value of internal voltage regulator

Table 39-31. ADC Resolution Following Digital Averaging (Typical Value) at 1.8V

Parameter Averaging Resolution (ADC_EMR.OSR)	Oversampling Ratio	Mode (bits)	INL (LSB)	DNL (LSB)	SNR (dB)	THD (dB)	ENOB (bits)
OSR = 0	1	12	+/-0.5	-0.5/+0.3	68.4	-86.7	11.1
OSR = 1	4	13	-0.9/+1	-0.6/0.5	72.1	-86.8	11.7
OSR = 2	16	14	+/-1.7	+/-0.7	75.2	-88.1	12.2
OSR = 3	64	15	-3.5/3.4	-1.4/1.6	76.5	-88.1	12.4
OSR = 4	256	16	-8/7.3	-4.3/4.4	76.9	-87.9	12.4

Table 39-32. ADC Resolution Following Digital Averaging (Typical Value) at 3.0V

Parameter Averaging Resolution (ADC_EMR.OSR)	Oversampling Ratio	Mode (bits)	INL (LSB)	DNL (LSB)	SNR (dB)	THD (dB)	ENOB (bits)
OSR = 0	1	12	+/-0.4	-0.4/0.2	70.7	-84.7	11.4
OSR = 1	4	13	+/-0.9	+/-0.7	74.2	-86.7	12.0
OSR = 2	16	14	-1.7/1.9	-1/1.3	77.6	-87.9	12.5
OSR = 3	64	15	-3.6/3.5	-1.8/1.9	79.1	-88.2	12.8
OSR = 4	256	16	-8.1/6.8	-4.8/4.1	79.6	-88	12.8

39.8.1.2 Single Mode Conditions

- Temperature = 25°C
- f_{ADC} = 10 MHz
- OSR: Number of averaged samples
- Single Mode
- V_{DDIO} = 1.8V or 3.0V
- V_{DDCORE} = value of internal voltage regulator

Table 39-33. ADC Resolution Following Digital Averaging (Typical Value) at 1.8V

Parameter Averaging Resolution OSR (ADC_EMR)	Oversampling Ratio	Mode (bits)	INL (LSB)	DNL (LSB)	SNR (dB)	THD (dB)	ENOB (bits)
OSR = 0	1	12	-0.8/+0.9	-0.6/0+.4	57.4	-73.7	9.2
OSR = 1	4	13	+/-1.6	-0.8/+1	67.8	-73.7	10.8
OSR = 2	16	14	-3.2/+3.1	+/-0.9	70.2	-73.8	11.1
OSR = 3	64	15	-6.3/+6.4	+/-1.9	69	-73.6	11.0
OSR = 4	256	16	-12.3/+12.9	-3.0/+3.6	72.6	-76.7	11.5

Table 39-34. ADC Resolution Following Digital Averaging (Typical Value) at 3.0V

Parameter Averaging Resolution OSR (ADC_EMR)	Oversampling Ratio	Mode (bits)	INL (LSB)	DNL (LSB)	SNR (dB)	THD (dB)	ENOB (bits)
OSR = 0	1	12	-0.8/+0.7	-0.5/+0.3	58.9	-75.7	9.5
OSR = 1	4	13	-1.6/1+.3	-0.8/+0.6	70.5	-75.4	11.2
OSR = 2	16	14	-3/+2.6	+/-0.9	73.2	-75.4	11.5
OSR = 3	64	15	-6.1/+5.2	-1.1/+1.4	70.7	-75.6	11.2
OSR = 4	256	16	-12.2/+9.8	-2/+2.3	75.9	-78	12.0

39.8.2 Static Performance Characteristics

Table 39-35. Static Performance Characteristics 12-bit Mode INL, DNL

Parameter	Conditions	Min	Typ	Max	Unit
Native ADC Resolution	—	—	—	12	bit
Resolution with Digital Averaging	See Analog-to-Digital Converter (ADC)	—	—	16	bit
Integral Non-linearity (INL)	Single mode	-2.5	+/-1	+2.5	LSB
	Differential Mode	-2	+/-1	+2	
Differential Non-linearity (DNL)	Single Mode	-1	+/-0.5	+1	LSB
	Differential Mode	-1	+/-0.5	+1	

Table 39-36. Static Performance Characteristics Gain and Error Offset 12-bit Mode

Parameter	Conditions	Min	Typ	Max	Unit
Offset Error	Single Mode	-3	+/-1	+3	LSB
	Differential Mode	-2	+/-1	+2	
Gain Error	Single Mode	-3	+/-1	+3	
	Differential Mode	-2	+/-1	+2	

39.8.3 Dynamic Performance Characteristics

Table 39-37. Dynamic Performance Characteristics in Single Mode 12-bit Mode (1)

Parameter	Conditions	Min	Typ	Max	Unit
Signal to Noise Ratio (SNR)	—	—	64	—	dB
Total Harmonic Distortion (THD)	—	—	-70	—	dB
Signal to Noise and Distortion (SINAD)	—	—	63	—	dB
Effective Number of Bits (ENOB)	—	—	10	—	bit

Note: 1. ADC Clock (f_{ADC}) = 10 MHz, f_{SAMPLE} = 500 ksps, f_{IN} = 62 kHz, FFT using 512 points or more, frequency band = [1 kHz, 250 kHz] – Nyquist conditions fulfilled.

Table 39-38. Dynamic Performance Characteristics in Differential Mode 12-bit Mode (1)

Parameter	Conditions	Min	Typ	Max	Unit
Signal to Noise Ratio (SNR)	—	—	70	—	dB
Total Harmonic Distortion (THD)	—	—	-74	—	dB
Signal to Noise and Distortion (SINAD)	—	—	68	—	dB
Effective Number of Bits (ENOB)	—	—	11	—	bit

Note: 1. ADC Clock (f_{ADC}) = 10 MHz, f_{SAMPLE} = 500 ksps, f_{IN} = 62 kHz, FFT using 1024 points or more, frequency band = [1 kHz, 250 kHz] – Nyquist conditions fulfilled.

39.8.4 External Reference Voltage

V_{ADVREF} is an external reference voltage applied on the pin ADVREF (available on 64-pin packages only). If ADC not used it is recommended to connect it to VDDIO.

The quality of the reference voltage V_{ADVREF} is critical to the performance of the ADC. A DC variation of the reference voltage V_{ADVREF} is converted to a gain error by the ADC. The noise generated by V_{ADVREF} is converted by the ADC to count noise.

Table 39-39. ADVREF Electrical Characteristics

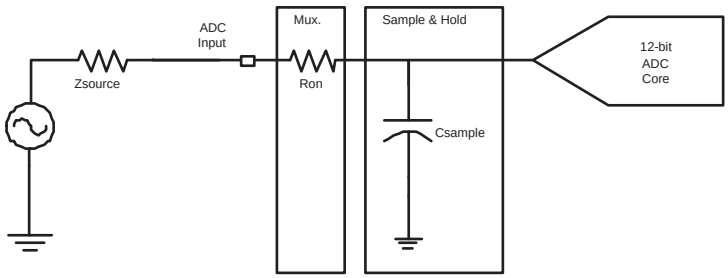
Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{ADVREF}	ADVREF Voltage Range	Full operational	1.62	–	3.6	V
V_n	Input Voltage Noise	Bandwidth from 1 kHz to 1 MHz	–	–	100	µVrms
R_{ADVREF}	ADVREF Input DC Impedance	Reference resistor bridge(1)	5	8	11	kΩ

Notes: 1. When the ADC is in Sleep mode, the ADVREF impedance has a minimum of 1 MΩ.

39.8.5 Track and Hold Time versus Source Output Impedance

Figure 39-13 shows a simplified acquisition path.

Figure 39-13. Simplified Acquisition Path



During the tracking phase, the ADC must track the input signal during the tracking time shown below:

12-bit mode: $t_{TRACKTIM} = 0.12 \times Z_{SOURCE} + 250$

With $t_{TRACKTIM}$ expressed in ns and Z_{SOURCE} expressed in ohms.

Table 39-40. 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 V_{DDIO} without destruction or over-consumption.

39.8.6 Autotest Values

Table 39-41. Autotest Mode Values

ADC_ACR.AUTOTEST Configuration	Code
NO_AUTOTEST	–
OFFSET_ ERROR	2048
GAIN_ERROR_HIGH	3840
GAIN_ERROR_LOW	256

39.9 AC Characteristics

39.9.1 External Reset

Table 39-42. I/O Characteristics

Symbol	Parameter	Conditions	Min	Max	Unit
t_{EXT}	Minimum reset pulse width (pulse to apply on NRST pin to guarantee the reset)		1	—	SCLK

39.9.2 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 V_{DDIO} - 100 mV
- Minimum output swing: 100 mV to V_{DDIO} - 100 mV
- Addition of rising and falling time inferior to 75% of the period

Table 39-43. I/O Characteristics

Symbol	Parameter	Conditions	Min	Max	Unit
FreqMax1	Pin Group 1[1] Maximum Output Frequency	25 pF	—	25	MHz
$PulseminH_1$	Pin Group 1 [1] High Level Pulse Width		20	—	ns
$PulseminL_1$	Pin Group 1 [1] Low Level Pulse Width		20	—	
FreqMax2	Pin Group 2[2]Maximum Output Frequency with Drive HIGH	10 pF	—	40	MHz
		25 pF	—	20	
	Pin Group 2[2]Maximum Output Frequency with Drive LOW	10 pF	—	65	MHz
		25 pF	—	33	
$PulseminH_2$	Pin Group 2[2] High Level Pulse Width with Drive HIGH	10 pF	12.5	—	ns
		25 pF	25	—	
	Pin Group 2[2] High Level Pulse Width with Drive LOW	10 pF	7.7	—	ns
		25 pF	15	—	
$PulseminL_2$	Pin Group 2[2] Low Level Pulse Width with Drive HIGH	10 pF	12.5	—	ns
		25 pF	25	—	
	Pin Group 2[2] Low Level Pulse Width with Drive LOW	10 pF	7.7	—	ns
		25 pF	15	—	
FreqMax3	Pin Group 3[3] Maximum Output Frequency	10 pF	—	65	MHz
		30 pF	—	40	
$PulseminH_3$	Pin Group 3[3] High Level Pulse Width	10 pF	7.7	—	ns
		30 pF	12.5	—	
$PulseminL_3$	Pin Group 3[3] Low Level Pulse Width	10 pF	7.7	—	ns
		30 pF	12.5	—	

Notes:
1. Pin Group 1 = PA21, 22 if USB used is Full Speed mode
2. Pin Group 2 = All others, NRST
3. Pin Group 3 = PA3, PA4, PA14

39.9.3 PDM Characteristics

Figure 39-14. PDM Mode

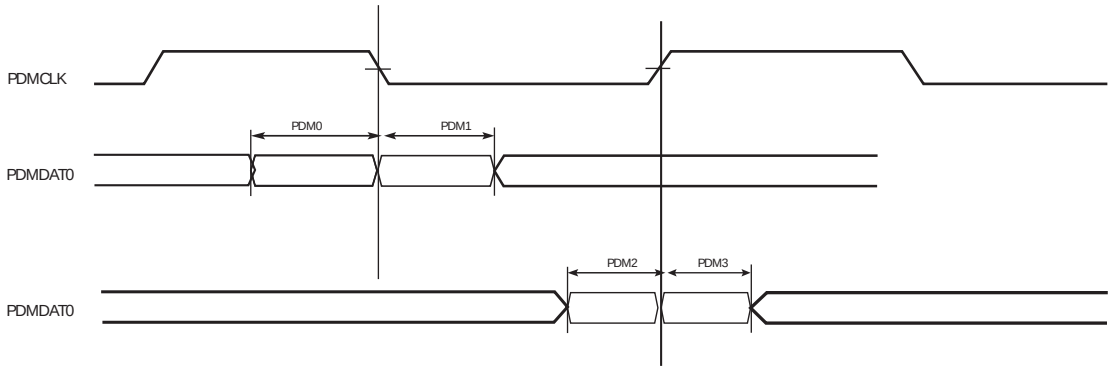


Table 39-44. PDM Characteristics

Symbol	Parameter	Conditions	Min	Max	Unit
PDMCLK	Clock Frequency	—	1	4	MHz
PDM_0	Data Setup Falling Edge	1.8V domain	4.0	—	ns
		3.3V domain	4.0	—	ns
PDM_1	Data Hold Falling Edge	1.8V domain	0	—	ns
		3.3V domain	0	—	ns
PDM_2	Data Setup Rising Edge	1.8V domain	4.0	—	ns
		3.3V domain	4.0	—	ns
PDM_3	Data Hold Rising Edge	1.8V domain	0	—	ns
		3.3V domain	0	—	ns

Notes: 1. 1.8V domain: V_{DDIO} from 1.62V to 2V, maximum external capacitor = 20 pF
2. 3.3V domain: V_{DDIO} from 2.85V to 3.60, maximum external capacitor = 20 pF

39.9.4 SPI Characteristics

In Figure 39-16 “SPI Master Mode with (CPOL = NCPHA = 0) or (CPOL = NCPHA = 1)” and Figure 39-17 “SPI Master Mode with (CPOL = 0 and NCPHA=1) or (CPOL = 1 and NCPHA = 0)” below, the MOSI line shifting edge is represented with a hold time = 0. However, it is important to note that for this device, the MISO line is sampled prior to the MOSI line shifting edge. As shown in Figure 39-15 “MISO Capture in Master Mode”, the device sampling point extends the propagation delay (t_p) for slave and routing delays to more than half the SPI clock period, whereas the common sampling point allows only less than half the SPI clock period.

As an example, an SPI Slave working in Mode 0 is safely driven if the SPI Master is configured in Mode 0.

Figure 39-15. MISO Capture in Master Mode

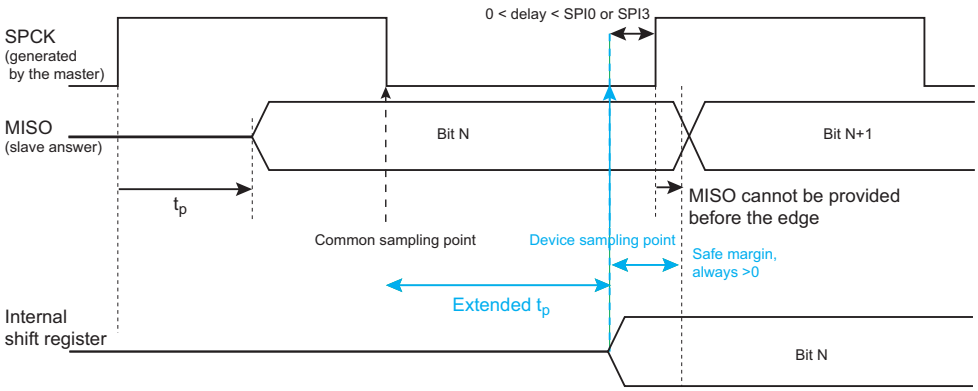


Figure 39-16. SPI Master Mode with (CPOL = NCPHA = 0) or (CPOL = NCPHA = 1)

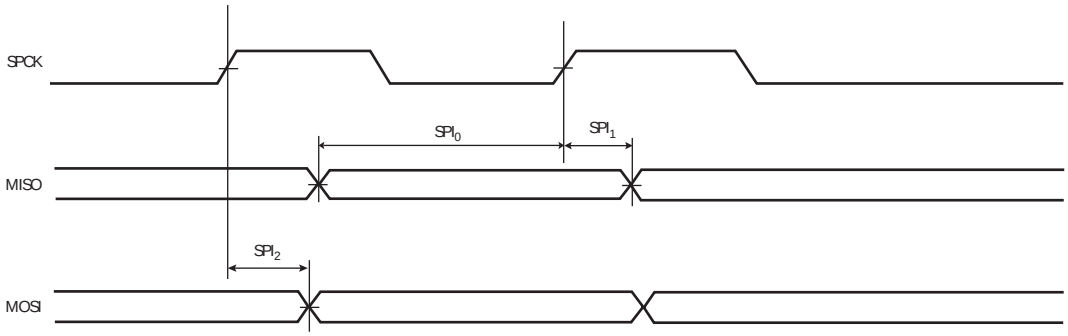


Figure 39-17. SPI Master Mode with (CPOL = 0 and NCPHA=1) or (CPOL = 1 and NCPHA = 0)

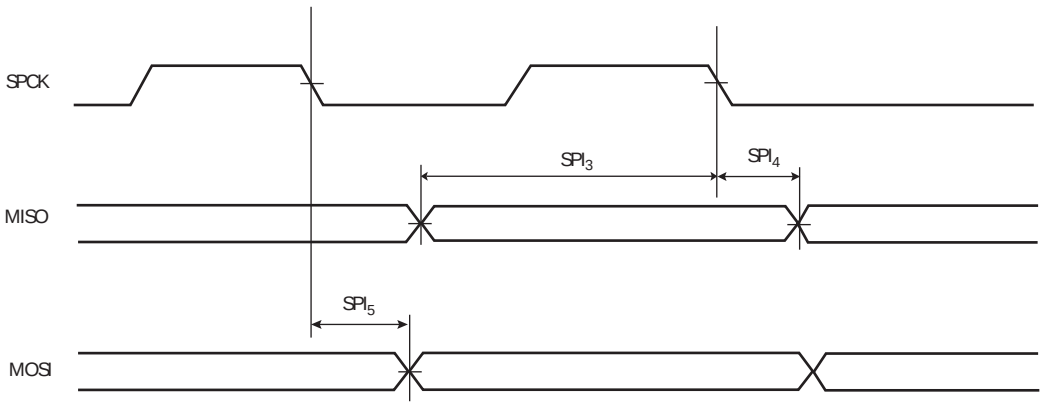


Figure 39-18. SPI Slave Mode with (CPOL = 0 and NCPHA = 1) or (CPOL = 1 and NCPHA = 0)

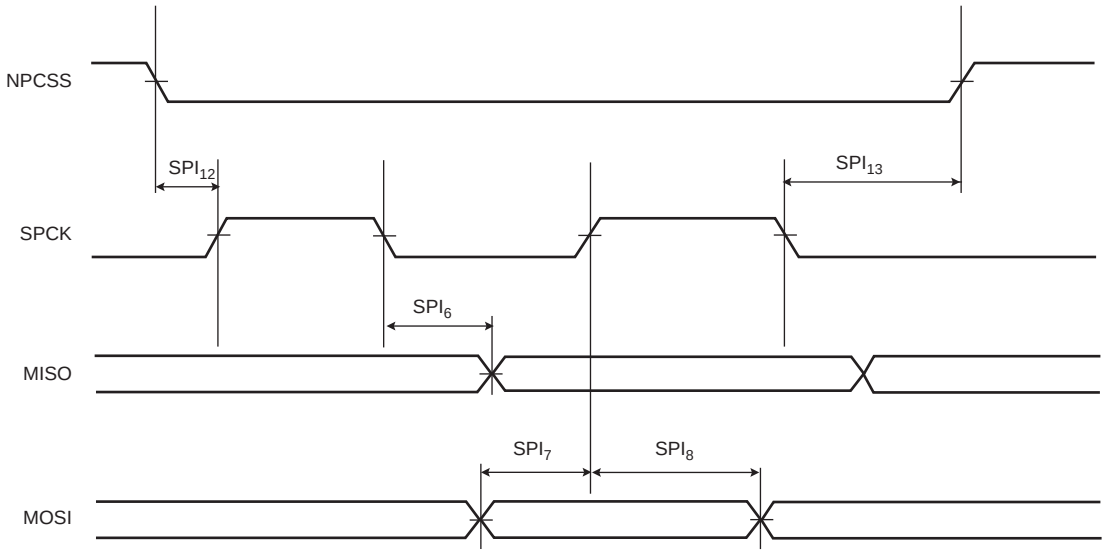
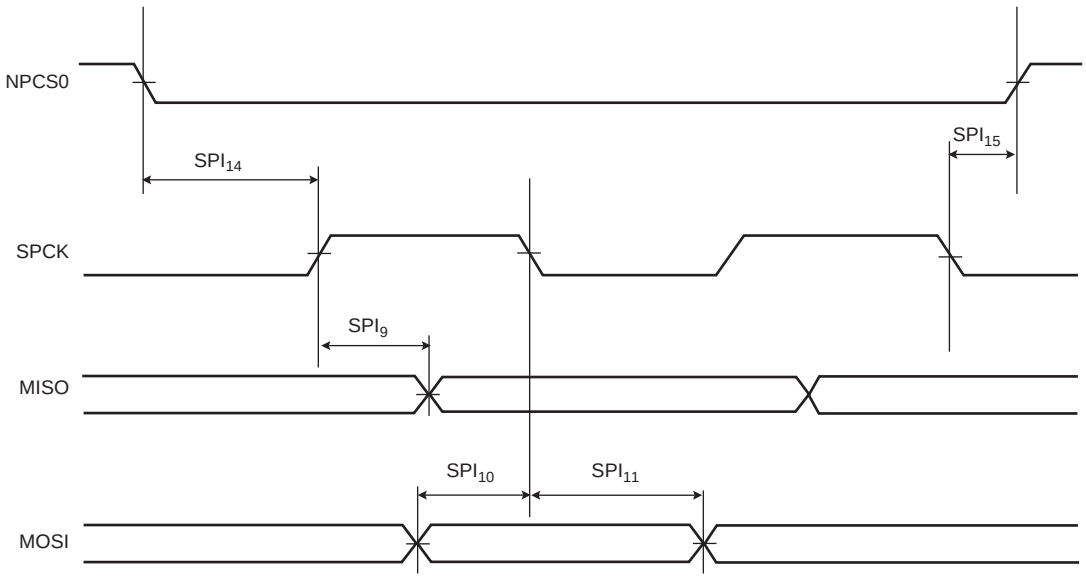


Figure 39-19. SPI Slave Mode with (CPOL = NCPHA = 0) or (CPOL = NCPHA = 1)



39.9.4.1 Maximum SPI Frequency

The following formulae give maximum SPI frequency in master Read and Write modes and in slave Read and Write modes.

Master Write Mode

The SPI sends data only to a slave device, e.g., to an LCD. The limit is given by SPI_2 (or SPI_5) timing. The SPI provides a maximum frequency greater than the maximum pad speed (see Section 39.9.2 “I/O Characteristics”) and thus the maximum SPI frequency is equivalent to the maximum pad speed.

Master Read Mode

$$f_{SPCK}Max = \frac{1}{SPI_0(orSPI_3) + t_{VALID}}$$

t_{VALID} is the slave time response to output data after detecting an SPCK edge. For a non-volatile memory with t_{VALID} (or t_V) = 12 ns Max.

f_{SPCK}max = 34.7 MHz @ V_{DDIO} = 3.3V.

Slave Read Mode

In Slave mode, SPCK is the input clock for the SPI. The maximum SPCK frequency is given by setup and hold timings SPI_7/SPI_8(or SPI_{10}/SPI_{11}). Since this gives a frequency well above the pad limit, the limit in slave Read mode is given by SPCK pad.

Slave Write Mode

$$f_{SPCK}Max = \frac{1}{2x(SPI_{6max}(orSPI_{9max}) + t_{SETUP}}$$

For I/O domain and SPI6, f_{SPCK}Max = 20 MHz. t_{SETUP} is the setup time from the master before sampling data.

39.9.4.2 SPI Timings

Table 39-45. SPI Timings

Symbol	Parameter	Conditions	Min	Max	Unit
SPI_0	MISO Setup time before SPCK rises (master)	1.8V domain[1]	19.3	—	ns
		3.3V domain[2]	16.8	—	ns
SPI_1	MISO Hold time after SPCK rises (master)	1.8V domain[1]	0	—	ns
		3.3V domain[2]	0	—	ns
SPI_2	SPCK rising to MOSI Delay (master)	1.8V domain[1]	0	4.8	ns
		3.3V domain[2]	0	3.2	ns
SPI_3	MISO Setup time before SPCK falls (master)	1.8V domain[1]	21.6	—	ns
		3.3V domain[2]	17.7	—	ns
SPI_4	MISO Hold time after SPCK falls (master)	1.8V domain[1]	0	—	ns
		3.3V domain[2]	0	—	ns
SPI_5	SPCK falling to MOSI Delay (master)	1.8V domain[1]	0	6.1	ns
		3.3V domain[2]	0	4.1	ns
SPI_6	SPCK falling to MISO Delay (slave)	1.8V domain[1]	4.6	20.3	ns
		3.3V domain[2]	4.0	16.3	ns
SPI_7	MOSI Setup time before SPCK rises (slave)	1.8V domain[1]	2.6	—	ns
		3.3V domain[2]	2.4	—	ns
SPI_8	MOSI Hold time after SPCK rises (slave)	1.8V domain[1]	4.1	—	ns
		3.3V domain[2]	4.1	—	ns
SPI_9	SPCK rising to MISO Delay (slave)	1.8V domain[1]	4.6	20.6	ns
		3.3V domain[2]	4.2	16.9	ns
SPI_{10}	MOSI Setup time before SPCK falls (slave)	1.8V domain[1]	3.0	—	ns
		3.3V domain[2]	3.0	—	ns
SPI_{11}	MOSI Hold time after SPCK falls (slave)	1.8V domain[1]	4.0	—	ns
		3.3V domain[2]	3.7	—	ns
SPI_{12}	NPCS setup to SPCK rising (slave)	1.8V domain[1]	6.0	—	ns
		3.3V domain[2]	6.0	—	ns
SPI_{13}	NPCS hold after SPCK falling (slave)	1.8V domain[1]	3.7	—	ns
		3.3V domain[2]	3.7	—	ns
SPI_{14}	NPCS setup to SPCK falling (slave)	1.8V domain[1]	6.4	—	ns
		3.3V domain[2]	6.4	—	ns
SPI_{15}	NPCS hold after SPCK falling (slave)	1.8V domain[1]	3.9	—	ns
		3.3V domain[2]	3.9	—	ns

Notes: 1. 1.8V domain: V_{DDIO} from 1.62V to 2V, maximum external capacitor = 20 pF.
2. 3.3V domain: V_{DDIO} from 2.85V to 3.60V, maximum external capacitor = 40 pF.

Note that in SPI master mode, the SAM G55 does not sample the data (MISO) on the edge opposite from where data clocks out (MOSI) but on the same edge. This is shown in Figure 39-16 and Figure 39-17.

39.9.5 USART in SPI Mode Timings

Figure 39-20. USART SPI Master Mode

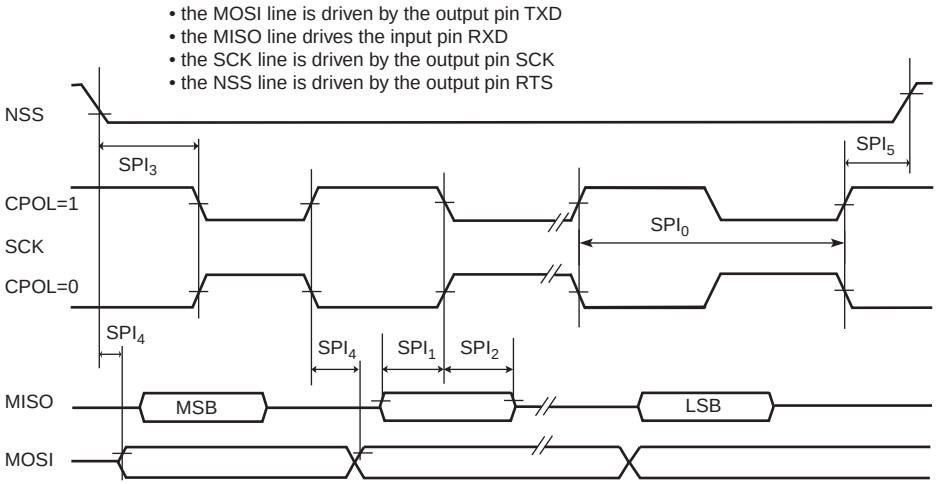


Figure 39-21. USART SPI Slave Mode (Mode 1 or 2)

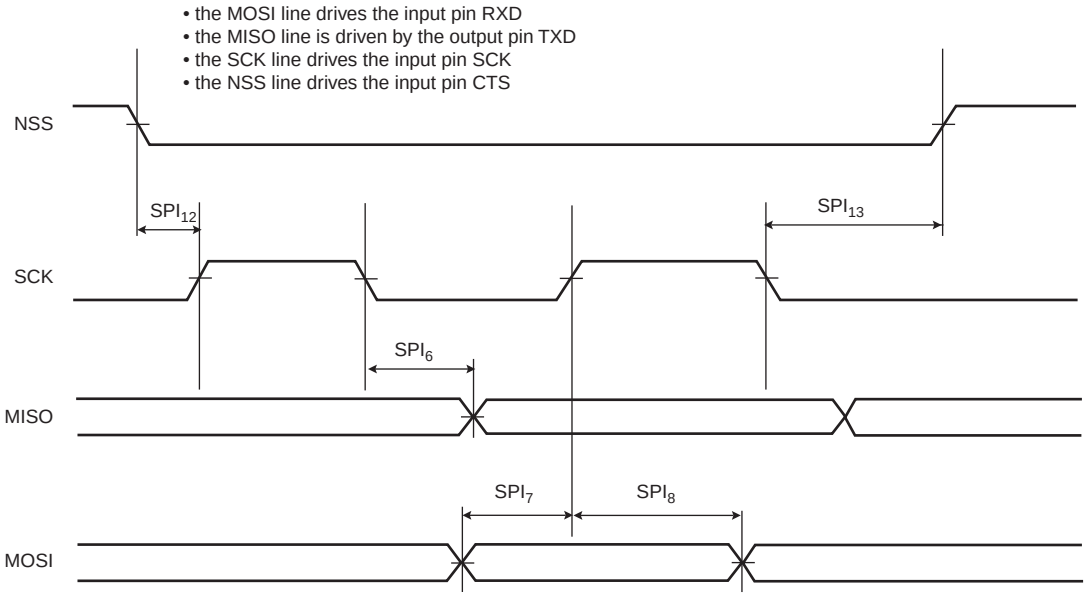
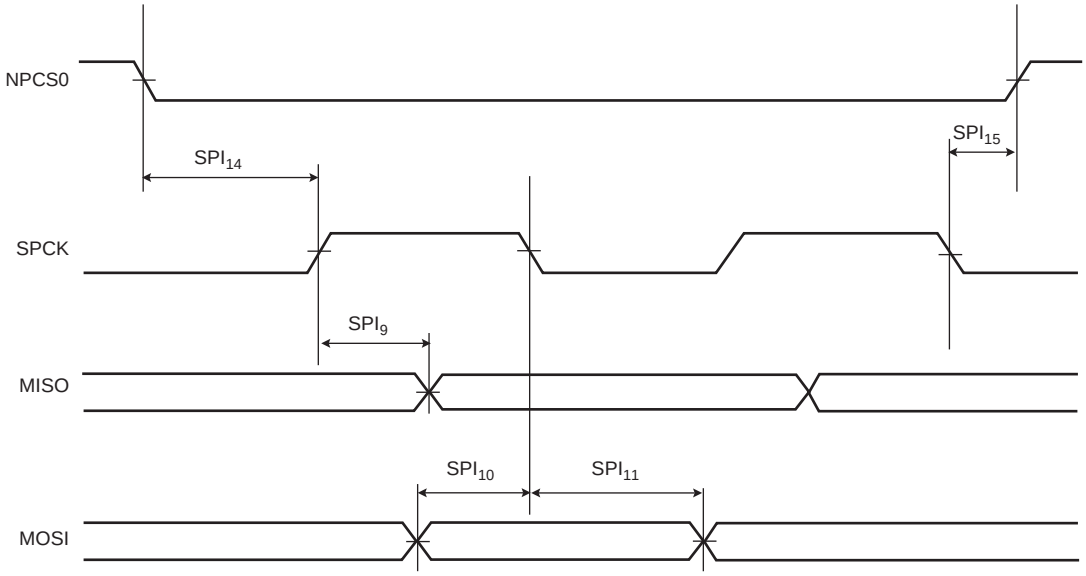


Figure 39-22. USART SPI Slave Mode (Mode 0 or 3)



39.9.5.1 USART SPI TImings

Table 39-46. USART SPI Timings

Symbol	Parameter	Conditions	Min	Max	Unit
Master Mode					
SPI_0	SCK Period	1.8V domain(1)	MCK/6	—	ns
		3.3V domain(2)	MCK/6	—	ns
SPI_1	Input Data Setup Time	1.8V domain(1)	0.5 × MCK + 4.2	—	ns
		3.3V domain(2)	0.5 × MCK + 4.2	—	ns
SPI_2	Input Data Hold Time	1.8V domain(1)	1.5 × MCK	—	ns
		3.3V domain(2)	1.5 × MCK	—	ns
SPI_3	Chip Select Active to Serial Clock	1.8V domain(1)	1.5 × SPCK - 3	—	ns
		3.3V domain(2)	1.5 × SPCK - 2.5	—	ns
SPI_4	Output Data Setup Time	1.8V domain(1)	0	12.4	ns
		3.3V domain(2)	0	10	ns
SPI_5	Serial Clock to Chip Select Inactive	1.8V domain(1)	1 × SPCK - 1.8	—	ns
		3.3V domain(2)	1 × SPCK - 1.5	—	ns
Slave Mode					
SPI_6	SCK falling to MISO	1.8V domain(1)	4	21	ns
		3.3V domain(2)	3.4	14.5	ns
SPI_7	MOSI Setup time before SCK rises	1.8V domain(1)	2 × MCK + 2.3	—	ns
		3.3V domain(2)	2 × MCK + 2.0	—	ns
SPI_8	MOSI Hold time after SCK rises	1.8V domain(1)	1.5	—	ns
		3.3V domain(2)	1.5	—	ns
SPI_9	SCK rising to MISO	1.8V domain(1)	4.0	19.5	ns
		3.3V domain(2)	3.5	14.5	ns
SPI_{10}	MOSI Setup time before SCK falls	1.8V domain(1)	2 × MCK + 2.8	—	ns
		3.3V domain(2)	2 × MCK + 2.8	—	ns
SPI_{11}	MOSI Hold time after SCK falls	1.8V domain(1)	0.8	—	ns
		3.3V domain(2)	0.5	—	ns
SPI_{12}	NPCS0 setup to SCK rising	1.8V domain(1)	2.5 × MCK + 0.3	—	ns
		3.3V domain(2)	2.5 × MCK + 0.2	—	ns
SPI_{13}	NPCS0 hold after SCK falling	1.8V domain(1)	1.5 × MCK + 1.3	—	ns
		3.3V domain(2)	1.5 × MCK + 1.0	—	ns
SPI_{14}	NPCS0 setup to SCK falling	1.8V domain(1)	2.5 × MCK + 0.6	—	ns
		3.3V domain(2)	2.5 × MCK + 0.5	—	ns
SPI_{15}	NPCS0 hold after SCK rising	1.8V domain(1)	1.5 × MCK +1.9	—	ns
		3.3V domain(2)	1.5 × MCK +1.8	—	ns

Notes: 1. 1.8V domain: V_{DDIO} from 1.62V to 2V, maximum external capacitor = 20 pF
2. 3.3V domain: V_{DDIO} from 2.85V to 3.60V, maximum external capacitor = 40 pF

39.9.6 Two-wire Serial Interface Characteristics

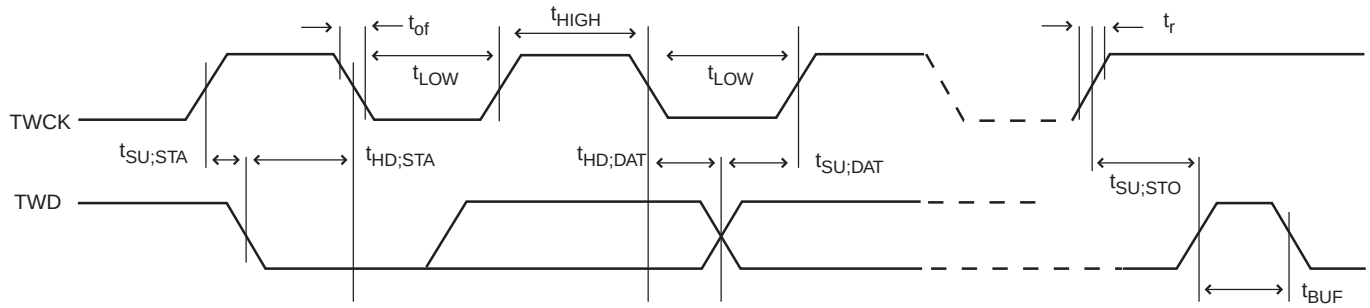
Table 39-47 provides the requirements for devices connected to the Two-wire Serial Bus and the compliance of the device with them. For timing symbols, refer to Figure 39-23.

Table 39-47. Two-wire Serial Bus Requirements

Symbol	Parameter	Condition	Min	Max	Unit
V_{HYST}	Hysteresis of Schmitt Trigger Inputs	—	0.150	—	V
V_{OL}	Low-level Output Voltage	3 mA sink current	—	0.4	V
t_R	Rise Time for both TWD and TWCK	—	$20 + 0.1C_B$ (1)	300	ns
t_{OF}	Output Fall Time from V_{IHmin} to V_{ILmax}	10 pF < C_b < 400 pF See Figure 39-23	$20 + 0.1C_B$ (1)	250	ns
C_i (1)	Capacitance for each I/O Pin	—	—	10	pF
f_{TWCK}	TWCK Clock Frequency	—	0	400	kHz
R_P	Value of Pull-up Resistor	$f_{TWCK} \le 100$ kHz	$\frac{V_{VDDIO} - 0.4V}{3mA}$	$\frac{1000ns}{C_B}$	Ω
		$f_{TWCK} > 100$ kHz	$\frac{V_{VDDIO} - 0.4V}{3mA}$	$\frac{300ns}{C_B}$	Ω
t_{LOW}	Low Period of the TWCK Clock	—	(2)	—	µs
t_{HIGH}	High Period of the TWCK Clock	—	(3)	—	µs
$t_{HD;STA}$	Hold Time (repeated) START Condition	—	t_{HIGH}	—	µs
$t_{SU;STA}$	Setup Time for a Repeated Start Condition	—	t_{HIGH}	—	µs
$t_{HD;DAT}$	Data Hold Time	—	0	(HOLD +3) × t_{CP_MCK} (4)	µs
$t_{SU;DAT}$	Data Setup Time	—	t_{LOW} - ((HOLD +3) × t_{CP_MCK} (4))	—	ns
$t_{SU;STO}$	Setup Time for STOP Condition	—	t_{HIGH}	—	µs
$t_{HD;STA}$	Hold Time (repeated) START Condition	—	t_{HIGH}	—	µs

Notes:
1. C_B = capacitance of one bus line in pF. Per I2C standard, C_B max = 400 pF
2. The TWCK low period is defined as follows: $t_{LOW} = ((CLDIV \times 2^{CKDIV}) + 4) \times t_{MCK}$
3. The TWCK high period is defined as follows: $t_{HIGH} = ((CHDIV \times 2^{CKDIV}) + 4) \times t_{MCK}$
4. The field HOLD is defined in the TWI Clock Waveform Generator Register (TWI_CWGR). t_{CP_MCK} = MCK bus period.

Figure 39-23. Two-wire Serial Bus Timing



39.9.7 High-speed Two-wire Serial Interface Characteristics

Table 39-48 provides the requirements for devices connected to the Two-wire Serial Bus.

For timing symbols, refer to Figure 39-23.

Table 39-48. High-Speed Two-wire Serial Bus Requirements

Symbol	Parameter	Condition	Min	Max	Unit
f_{TWCK}	TWCK Clock Frequency	Capacitive load			
		C_B = 100 pF (max)	—	3.4	MHz
		C_B = 400 pF[1]	—	1.7	
V_{HYST}	Hysteresis of Schmitt Trigger Inputs	—	0.1 × V_{DDIO}	—	V
V_{OL}	Low-level Output Voltage	—	—	0.2 × V_{DDIO}	V
t_{Rd}	Rise Time for both TWD/TWCK (After Acknowledge)	Capacitive load			
		C_B = 100 pF (max)	10	80	ns
		C_B = 400 pF[1]	20	160	
t_{OF}	Output Fall Time from V_{IHmin} to V_{ILmax}	Capacitive load			
		C_B =100 pF (max)	10	40	ns
		C_B = 400 pF[1]	20	80	
C_I	Capacitance for Each I/O Pin	—	—	10	pF
R_P	Value of Pull-up Resistor	$C_B \leq 100$ pF	$\frac{V_{VDDIO} - 0.4V}{3mA}$	$\frac{1000ns}{C_B}$	Ω
		100 pF < $C_B \leq$ 400 pF	$\frac{V_{VDDIO} - 0.4V}{3mA}$	$\frac{300ns}{C_B}$	Ω
$t_{HD;DAT}$	Data Hold Time	Capacitive load from 10 pF to 100 pF	0	70	µs
		Capacitive load C_B = 400 pF[1]	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[1]	10	—	ns

Note: 1. For bus line loads C_B between 100 pF and 400 pF, the timing parameters must be linearly interpolated.

39.9.8 Embedded Flash Characteristics

The maximum operating frequency is given in Table 39-49 below but is limited by the embedded Flash access time when the processor is fetching code out of it. Table 39-49 and Table 39-50 below give the device maximum and typical operating frequency depending on the field FWS of the EEFC Flash Mode Register (EEFC_FMR). This field defines the number of wait states required to access the embedded Flash memory.

Table 39-49. Embedded Flash Wait State at VDDIO 1.62V (Max Value)

FWS	Read Operations	Maximum Operating Frequency (MHz)
0	1 cycle	20
1	2 cycles	40
2	3 cycles	60
3	4 cycles	80
4	5 cycles	100
5	6 cycles	120

The following characteristics are applicable at 25°C unless otherwise specified.

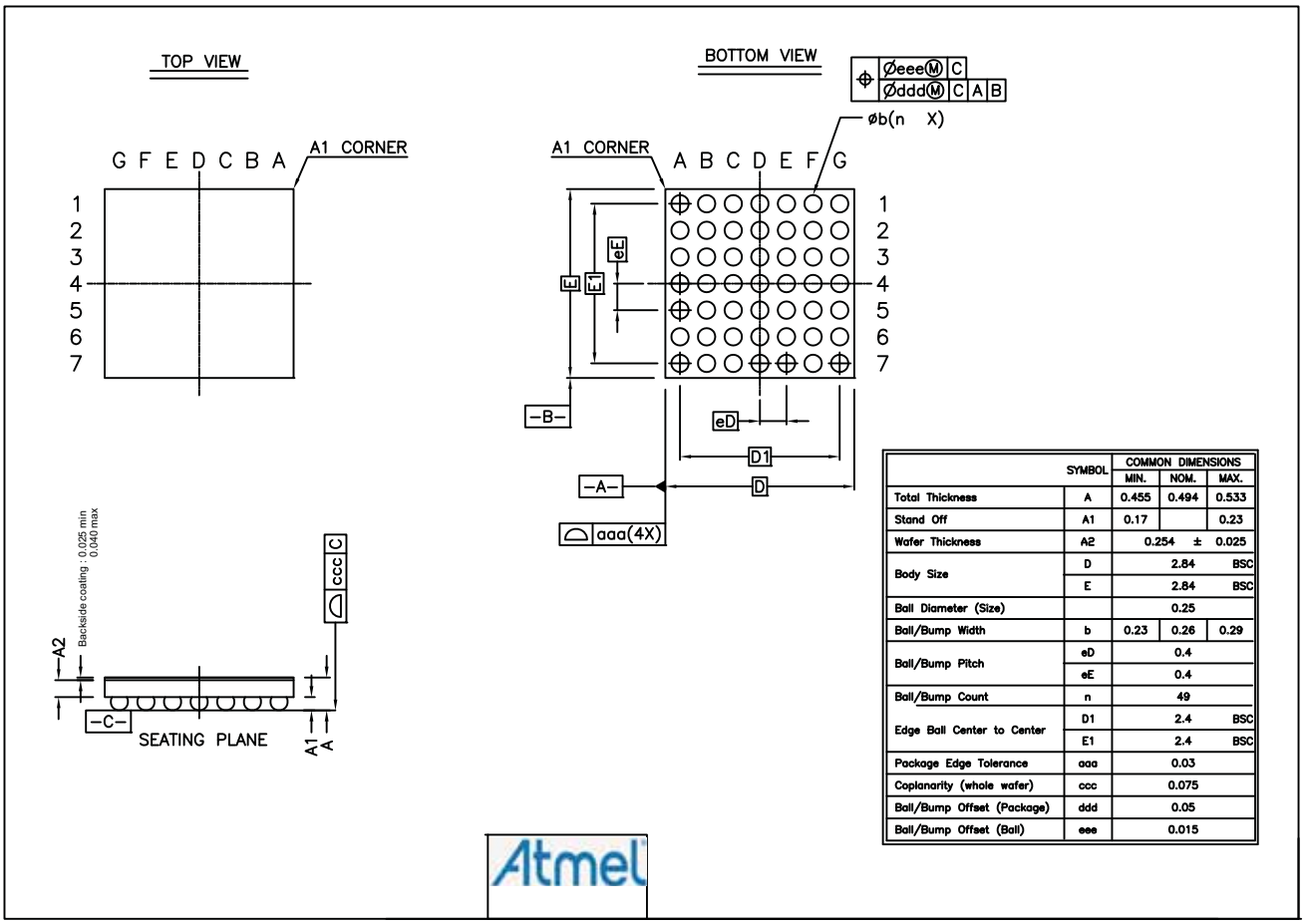
Table 39-50. AC Flash Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
I_{CC}	Flash Active Current on VDDCORE	Random 128-bit read at max frequency	—	16	20	mA
		Random 64-bit read at max frequency	—	8	10	
		Program	—	3	5	
		Erase	—	3	5	
I_{CC20}	Flash Active Current on VDDIO	Random 128-bit read at max frequency	—	6	10	mA
		Random 64-bit read at max frequency	—	6	10	
		Program	—	10	15	
		Erase	—	10	15	
–	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 8Kbytes)	—	80	200	ms
		Erase sector mode (Sector of 112/128 Kbytes)	—	800	1500	ms
		Lock/Unlock time per region	—	—	—	ms
–	Full Chip Erase	512 Kbytes	—	3	6	s
–	Data Retention	Not powered or powered	—	20		years
–	Erase Pin Debounce Time	Erase pin at high level	200	—	—	ms
–	Endurance	Write/Erase cycles per page, block or sector at temp = 85°C	10K	—		cycles

40. Mechanical Characteristics

40.1 49-lead WLCSP Package

Figure 40-1. 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 40-1. Device and 49-ball WLCSP Package Maximum Weight

SAMG55	8.444	mg

Table 40-2. Package Reference

JEDEC Drawing Reference	na
JESD97 Classification	e1

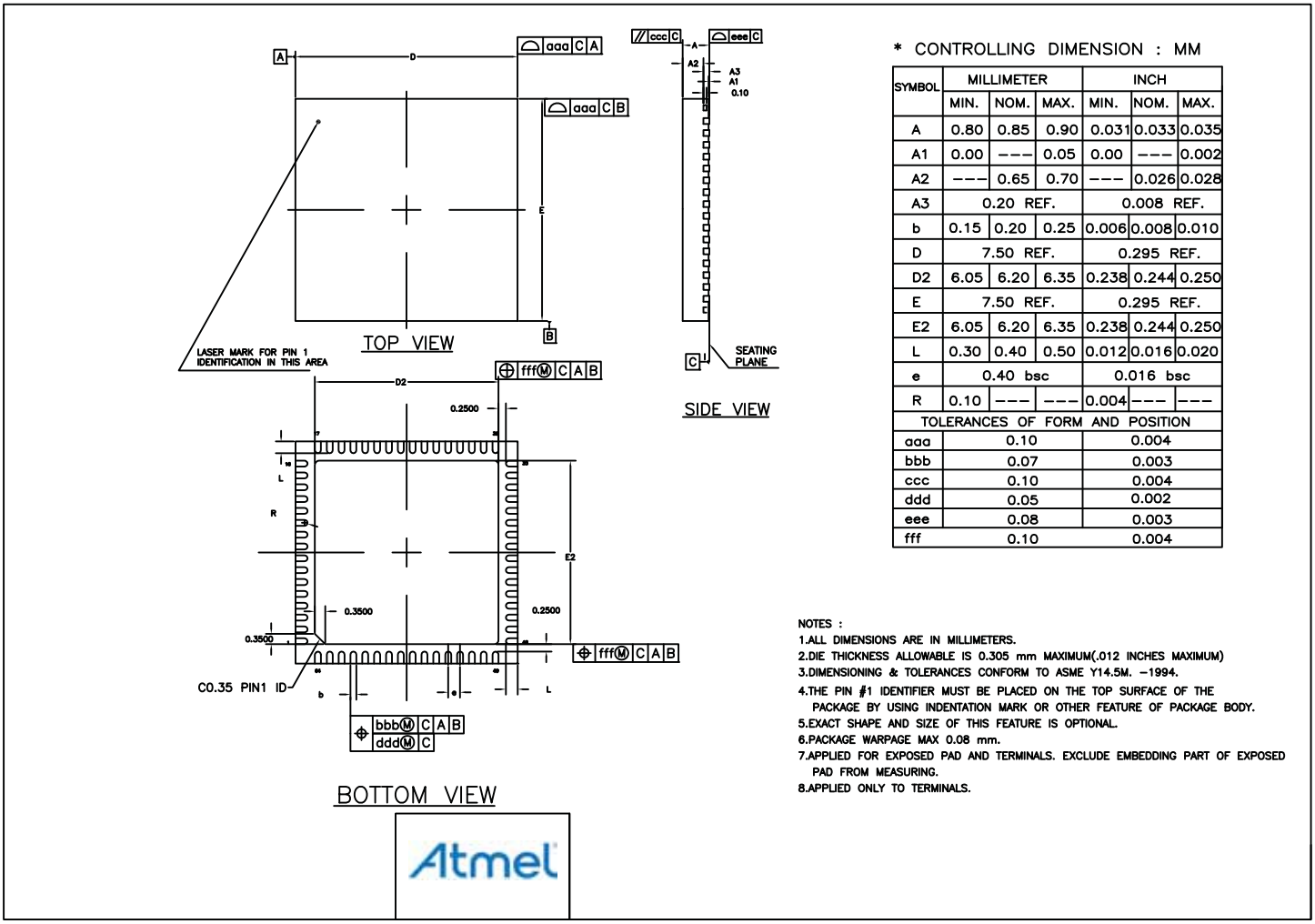
Table 40-3. 49-ball WLCSP Package Characteristics

Moisture Sensitivity Level	1

This package respects the recommendations of the NEMI User Group.

40.2 64-lead QFN Package

Figure 40-2. 64-lead QFN Package Mechanical Drawing



* CONTROLLING DIMENSION : MM

SYMBOL	MILLIMETER			INCH		
	MIN.	NOM.	MAX.	MIN.	NOM.	MAX.
A	0.80	0.85	0.90	0.031	0.033	0.035
A1	0.00	---	0.05	0.00	---	0.002
A2	---	0.65	0.70	---	0.026	0.028
A3	0.20 REF.			0.008 REF.		
b	0.15	0.20	0.25	0.006	0.008	0.010
D	7.50 REF.			0.295 REF.		
D2	6.05	6.20	6.35	0.238	0.244	0.250
E	7.50 REF.			0.295 REF.		
E2	6.05	6.20	6.35	0.238	0.244	0.250
L	0.30	0.40	0.50	0.012	0.016	0.020
e	0.40 bsc			0.016 bsc		
R	0.10	---	---	0.004	---	---
TOLERANCES OF FORM AND POSITION						
aaa	0.10			0.004		
bbb	0.07			0.003		
ccc	0.10			0.004		
ddd	0.05			0.002		
eee	0.08			0.003		
fff	0.10			0.004		

NOTES :
1.ALL DIMENSIONS ARE IN MILLIMETERS.
2.DIE THICKNESS ALLOWABLE IS 0.305 mm MAXIMUM(.012 INCHES MAXIMUM)
3.DIMENSIONING & TOLERANCES CONFORM TO ASME Y14.5M. -1994.
4.THE PIN #1 IDENTIFIER MUST BE PLACED ON THE TOP SURFACE OF THE PACKAGE BY USING INDENTATION MARK OR OTHER FEATURE OF PACKAGE BODY.
5.EXACT SHAPE AND SIZE OF THIS FEATURE IS OPTIONAL.
6.PACKAGE WARPAGE MAX 0.08 mm.
7.APPLIED FOR EXPOSED PAD AND TERMINALS. EXCLUDE EMBEDDING PART OF EXPOSED PAD FROM MEASURING.
8.APPLIED ONLY TO TERMINALS.

Table 40-4. Device and 64-lead QFN Package Maximum Weight

160	mg

Table 40-5. Package Reference

JEDEC Drawing Reference	MO-220
JESD97 Classification	e3

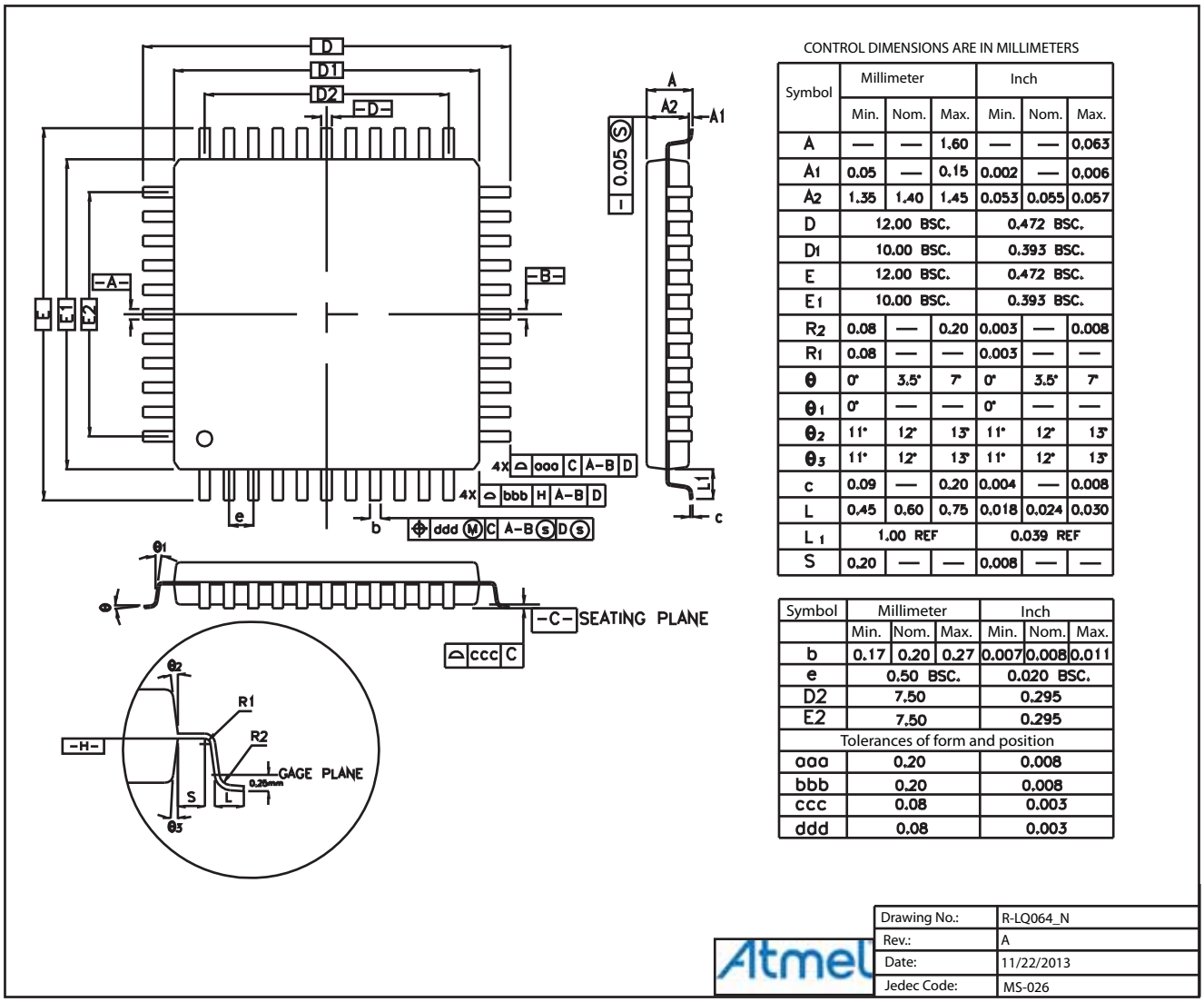
Table 40-6. 64-lead QFN Package Characteristics

Moisture Sensitivity Level	3

This package respects the recommendations of the NEMI User Group.

40.3 64-lead LQFP Mechanical Characteristics

Figure 40-3. 64-lead LQFP Package Drawing



Symbol	Millimeter			Inch		
	Min.	Nom.	Max.	Min.	Nom.	Max.
A	—	—	1.60	—	—	0.063
A1	0.05	—	0.15	0.002	—	0.006
A2	1.35	1.40	1.45	0.053	0.055	0.057
D	12.00 BSC.			0.472 BSC.		
D1	10.00 BSC.			0.393 BSC.		
E	12.00 BSC.			0.472 BSC.		
E1	10.00 BSC.			0.393 BSC.		
R2	0.08	—	0.20	0.003	—	0.008
R1	0.08	—	—	0.003	—	—
θ	0°	3.5°	7°	0°	3.5°	7°
θ1	0°	—	—	0°	—	—
θ2	11°	12°	13°	11°	12°	13°
θ3	11°	12°	13°	11°	12°	13°
c	0.09	—	0.20	0.004	—	0.008
L	0.45	0.60	0.75	0.018	0.024	0.030
L1	1.00 REF			0.039 REF		
S	0.20	—	—	0.008	—	—

Symbol	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	7.50			0.295		
E2	7.50			0.295		
Tolerances of form and position						
aaa	0.20			0.008		
bbb	0.20			0.008		
ccc	0.08			0.003		
ddd	0.08			0.003		

Drawing No.:	R-LQ064_N
Rev.:	A
Date:	11/22/2013
Jedec Code:	MS-026

Table 40-7. Device and 64-lead LQFP Package Maximum Weight

88.62	mg

Table 40-8. LQFP Package Reference

JEDEC Drawing Reference	MS-026
J-STD-609 Classification	e3

Table 40-9. 64-lead LQFP Package Characteristics

Moisture Sensitivity Level	3

This package respects the recommendations of the NEMI User Group.

40.4 Soldering Profile

Table 40-10 gives the recommended soldering profile from J-STD-020C.

Table 40-10. 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.

40.5 Packaging Resources

Land Pattern Definition.

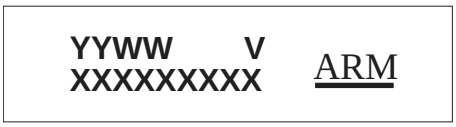
Refer to the following IPC Standards:

- 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/

41. Marking

All devices are marked with the Atmel logo and the ordering code.

Additional marking may be in one of the following formats:



where

- “YY”: manufactory year
- “WW”: manufactory week
- “V”: revision
- “XXXXXXXXX”: lot number

42. Ordering Information

Table 42-1. SAM G55 Ordering Information

Ordering Code	MRL	Package	Carrier Type	Operating Temperature Range
ATSAMG55G19A-UUT	A	WLCSP49	Reel	Industrial -40°C to 85°C
ATSAMG55G19B-UUT	B			
ATSAMG55J19A-MU	A	QFN64	Tray	Industrial -40°C to 85°C
ATSAMG55J19B-MU	B			
ATSAMG55J19A-MUT	A		Reel	
ATSAMG55J19B-MUT	B			
ATSAMG55J19A-AU	A	LQFP64	Tray	Industrial -40°C to 85°C
ATSAMG55J19B-AU	B			
ATSAMG55J19A-AUT	A		Reel	
ATSAMG55J19B-AUT	B			

43. Errata

Errata is described in the following sections:

43.1 Revision A Parts

Errata in this section is applicable to the devices listed in the following table:

Ordering Code
ATSAMG55G19A-UUT
ATSAMG55J19A-MU
ATSAMG55J19A-MUT
ATSAMG55J19A-AU
ATSAMG55J19A-AUT

43.1.1 USB

Issue: USB functionality not guaranteed over the whole temperature range

The USB device and host functionalities are guaranteed only in a commercial temperature range 0°C to +70°C.

Workaround: None.

43.1.2 Analog-to-Digital Converter (ADC)

Issue: INL and DNL values at 25°C

In Single-ended mode at 25°C, the INL and DNL values are as follows:

Table 43-1. Static Performance Characteristics 12-bit Mode INL, DNL

Parameter	Conditions	Min	Typ	Max	Unit
Integral Non-linearity (INL)	Single-ended mode	-3.3	±0.8	+3.4	LSB
Differential Non-linearity (DNL)		-2	-0.7 / +0.4	+1.2	LSB

Workaround: None.

Issue: INL and DNL values from 40°C to 85°C

In Single-ended mode from 40°C to 85°C, the INL and DNL values are as follows:

Table 43-2. Static Performance Characteristics 12-bit Mode INL, DNL

Parameter	Conditions	Min	Typ	Max	Unit
Integral Non-linearity (INL)	Single-ended mode	-5.5	±0.8	+5.5	LSB
Differential Non-linearity (DNL)		-2	-0.7 / +0.4	+1.2	LSB

Workaround: None.

43.1.3 Asynchronous Partial Wakeup (SleepWalking) (PMC)

Issue: SleepWalking limitation

If a wakeup request from the WKUP pins, an RTT event or an RTC event occurs simultaneously with the wakeup of a peripheral that has SleepWalking enabled and simultaneously entering wait mode, the device is not able to wake up.

Workaround: None.

Issue: PMC Asynchronous Wakeup

Using more than one wakeup event (WKUP or internal event such as RTC, etc.) may lead to a system deadlock.

Workaround: None

43.1.4 SUPC

Issue: No Write Protection On SUPC_WUIR

The System Controller Write Protection Mode register (SYSC_WPMR) does not work on the Supply Controller Wakeup Inputs Register (SUPC_WUIR). The SUPC_WUIR is not write-protected.

Workaround: None

43.1.5 TWI/TWIHS

Issue: TWI/TWIHS Clear command does not work

Bus reset using the “CLEAR” bit of the TWIHS control register does not work correctly during a bus busy state.

Workaround:

When the TWI master detects the SDA line stuck in low state, the procedure to recover is:

1. Reconfigure the SDA/SCL lines as PIO.
2. Try to assert a logic 1 on the SDA line (PIO output = 1) .
3. Read the SDA line state. If the PIO state is a logic 0 then generate a clock pulse on SCL (1-0-1 transition).
4. Read the SDA line state. If the SDA line = 0, go to Step 3; if SDA = 1, go to Step 5
5. Generate a STOP condition.
6. Reconfigure SDA/SCL PIOs as peripheral.

43.1.6 CMCC

Issue: RAM CMCC usage limitation

The 16 Kbytes of cache RAM cannot be allocated simultaneously to the RAM Cache and to the SRAM on I/D bus. If the cache is used, up to 14 Kbytes can be unusable (depending on the selected cache size). If the cache function is required to optimize the device performances, the size must be set to 8 Kbytes.

Workaround: None

43.2 Revision B Parts

Errata in this section is applicable to the devices listed in the following table:

Ordering Code
ATSAMG55G19B-UUT
ATSAMG55J19B-MU
ATSAMG55J19B-MUT
ATSAMG55J19B-AU
ATSAMG55J19B-AUT

43.2.1 SUPC

Issue: No write protection on SUPC_WUIR

The System Controller Write Protection Mode register (SYSC_WPMR) does not work on the Supply Controller Wakeup Inputs Register (SUPC_WUIR). The SUPC_WUIR is not write-protected.

Workaround: None

43.2.2 TWI/TWIHS

Issue: TWI/TWIHS Clear command does not work

Bus reset using the “CLEAR” bit of the TWIHS control register does not work correctly during a bus busy state.

Workaround:

When the TWI master detects the SDA line stuck in low state the procedure to recover is:

1. Reconfigure the SDA/SCL lines as PIO.
2. Try to assert a Logic 1 on the SDA line (PIO output = 1).
3. Read the SDA line state. If the PIO state is a Logic 0 then generates a clock pulse on SCL (1-0-1 transition).
4. Read the SDA line state. If the SDA line = 0, go to Step 3; if SDA = 1, go to Step 5.
5. Generate a STOP condition.
6. Reconfigure SDA/SCL PIOs as peripheral.

43.2.3 CMCC

Issue: RAM CMCC usage limitation

The 16 Kbytes of cache RAM cannot be allocated simultaneously to the RAM Cache and to the SRAM on I/D bus. If the cache is used, up to 14 Kbytes can be unusable (depending on the selected cache size). If the cache function is required to optimize the device performances, the size must be set to 8 Kbytes.

Workaround: None

44. Revision History

In the tables that follow, the most recent version of the document appears first.

Table 44-1. SAM G55 Datasheet Rev. 11289F Revision History

Issue Date	Changes
27-May-16	Minor editorial and formatting changes throughout
	"Features" "USB 2.0 Device" changed to "Crystal-less USB 2.0 Device"
	Section 1. "Configuration Summary" Table 1-1 "Configuration Summary": removed instance of "TWIHS"
	Section 2. "Block Diagram" Figure 2-1 "SAM G55 Block Diagram": repositioned 'VUSB' input and renamed to 'VDDUSB'
	Section 3. "Signal Description" Table 3-1 "Signal Description List": renamed 'VUSB' to 'VDDUSB'; inserted row "Pulse Density Modulation Interface Controller - PDMICx"; "USB OHCI/FS/IC" changed to "USB OHCI/FS"
	Section 5. "Power Considerations" Updated Figure 5-2 "Single Supply", Figure 5-3 "Dual Supply" and Table 5-1 "Low-power Mode Configuration Summary"
	Section 6. "Product Mapping" Figure 6-1 "SAM G55 Product Mapping": updated addresses for Internal SRAM blocks
	Section 8. "Memories" Figure 8-2 "Flash Sector Organization": "A sector size is 64 KBytes" changed to "A sector size is 128 Kbytes" Updated Section 8.3.1.8 "Calibration Bits"
	Section 11. "Debug and Test Features" Section 11.6.4 "Debug Architecture": in first paragraph, "four functional units" corrected to "fivie functional units" Updated Section 11.6.9 "IEEE 1149.1 JTAG Boundary Scan" Section 11.6.10 "ID Code Register": updated descriptions for bit [0] and for field PART NUMBER
	Section 12. "Chip Identifier (CHIPID)" Updated Table 12-1 "SAM G55 Chip ID Registers" with MRL B chip identifiers
	Section 14. "Cortex-M Cache Controller (CMCC)" Section 14.2 "Embedded Characteristics": bullet "Write through cache operations, read allocate" changed to "Write accesses forwarded, cache state not modified. Allocate on read."
	Section 15. "Bus Matrix (MATRIX)" Section 15.2 "Embedded Characteristics": added bullet "Automatic clock-off mode for power reduction" Section 15.6.1.4 "Fixed Priority Arbitration": in last sentence, deleted instance of "and MATRIX_PRBS" Table 15-4 "Register Mapping": - added registers MATRIX_MCFG3, MATRIX_MCFG4, MATRIX_SCFG4 and MATRIX_PRAS4 - defined offset range 0x00A4–0x010C as "reserved" Section 15.9.1 "Bus Matrix Master Configuration Registers": index [x = 0..2] updated to [x = 0..4] Section 15.9.2 "Bus Matrix Slave Configuration Registers": index [x = 0..3] updated to [x = 0..4] Section 15.9.3 "Bus Matrix Priority Registers A For Slaves": modified title (was "Bus Matrix Priority Registers For Slaves"); index MATRIX_PRAS0..MATRIX_PRAS3 updated to MATRIX_PRAS0..MATRIX_PRAS4; added field M4PR

Table 44-1. SAM G55 Datasheet Rev. 11289F Revision History (Continued)

Issue Date	Changes
27-May-16	Section 16. “Parallel Input/Output Controller (PIO)” Table 16-4 “Register Mapping”: - “Peripheral Select Register 1” corrected to “Peripheral ABCD Select Register 1” - “Peripheral Select Register 2” corrected to “Peripheral ABCD Select Register 2” Section 16.6.24 “PIO Peripheral ABCD Select Register 1”: added addresses Section 16.6.25 “PIO Peripheral ABCD Select Register 2”: corrected addresses
	Section 19. “Reset Controller (RSTC)” ‘Slow crystal’ changed to ‘32.768 kHz crystal’ throughout; instances of “is set” changed to “is written to 1”; instances of “is reset” changed to “is written to 0” Reworked Section 19.1 “Description” and Section 19.2 “Embedded Characteristics” Updated Figure 19-1 “Reset Controller Block Diagram” and Figure 19-3 “General Reset Timing Diagram” Updated Section 19.4.2.1 “NRST Signal or Interrupt” Updated Section 19.4.3.3 “32.768 kHz Crystal Oscillator Failure Detection Reset” and Section 19.4.3.4 “Watchdog Reset” Updated Figure 19-6 “Software Reset Timing Diagram” and Figure 19-7 “User Reset Timing Diagram”
	Section 20. “Watchdog Timer (WDT)” Section 20.4 “Functional Description”: changed “To prevent a software deadlock that continuously triggers the watchdog, the reload of the watchdog must occur...” to read “The reload of the watchdog must occur...”
	Section 23. “Enhanced Embedded Flash Controller (EEFC)” Section 23.4.3.6 “Calibration Bit”: changed information on oscillators that are calibrated in production
	Section 24. “Timer Counter (TC)” Reformatted and renamed Table 24-2 “Channel Signal Description” Updated Section 24.6.3 “Clock Selection” and Section 24.6.9 “Transfer with PDC in Capture Mode”
	Section 25. “Supply Controller (SUPC)” Table 25-2 “Register Mapping”: SUPC_MR reset value 0x00E0_5A00 corrected to 0x00E0_5400 Added “This register can only be written if the WPEN bit is cleared in the System Controller Write Protection Mode Register (SYSC_WPMR)” in Section 25.5.3 “Supply Controller Control Register” and Section 25.5.4 “Supply Controller Supply Monitor Mode Register” Section 25.5.5 “Supply Controller Mode Register”: - added “This register can only be written if the WPEN bit is cleared in the System Controller Write Protection Mode Register (SYSC_WPMR).” - added note “Bits 23 and 14 must always be written to ‘1’.” - in bitmap cell 23, renamed “ONE” to “ONEA” and added corresponding bit description - added bit ONE in bitmap cell 14 - modified VDDSEL bit description Section 25.5.9 “Supply Controller Power Mode Register”: modified ECPWRS bit description
	Section 26. “Real-time Clock (RTC)” Updated Section 26.5.6 “Updating Time/Calendar” Reworked Figure 26-5 “Calibration Circuitry Waveforms” AD channel index ‘7’ replaced with generic ‘n’ in Section 26.5.8 “Waveform Generation” Modified Figure 26-6 “Waveform Generation for ADC Trigger Event” Section 26.6.1 “RTC Control Register”: added ‘3’ to CALEVSEL field values Section 26.6.2 “RTC Mode Register”: added fields TPERIOD and THIGH

Table 44-1. SAM G55 Datasheet Rev. 11289F Revision History (Continued)

Issue Date	Changes
27-May-16	Section 28. “General Purpose Backup Registers (GPBR)” Updated Section 28.1 “Description” Updated Section 28.2 “Embedded Characteristics” Section 28.3.1 “General Purpose Backup Register x”: updated GPBR_VALUE field description
	Section 29. “Flexible Serial Communication Controller (FLEXCOM)” Updated Table 29-1 “I/O Line Description” Section 29.5.3 “Interrupt Sources”: changed title (was “Interrupt”) Table 29-2 “FLEXCOM Configuration”: updated TWI function names Section 29.7 “Flexible Serial Communication Controller (FLEXCOM) User Interface”: modified title (was “Flexible Serial Communication Unit (FLEXCOM) User Interface”) Section 29.7.1 “FLEXCOM Mode Register”: removed mention of “RS485” from OPMODE field description
	Section 30. “Universal Synchronous Asynchronous Receiver Transceiver (USART)” Section 30.5 “Product Dependencies”: instances of “FX_MR” corrected to “FLEX_MR” Section 30.6.1.2 “Fractional Baud Rate in Asynchronous Mode”: deleted sentence “This feature is only available when using USART normal mode.” Section 30.6.3.8 “Receiver Timeout”: deleted redundant paragraphs on STTTO and RETTO. Section 30.6.4 “ISO7816 Mode”: "...to the value 0x5 for protocol T = 1" changed to "...to the value 0x6 for protocol T = 1"
	Section 31. “Serial Peripheral Interface (SPI)” Section 31.6 “Product Dependencies”: instances of “FX_MR” corrected to “FLEX_MR” Section 31.7.5 “SPI Comparison on Received Character”: “The comparison trigger event is restarted by writing a 1 to the REQCLR bit in SPI_CR” changed to “The comparison trigger event is restarted by setting the REQCLR bit in SPI_CR if SleepWalking mode is disabled”
	Section 32. “Two-wire Interface (TWI)” Change all instances of “TWIHS” to “TWI” Deleted bullet on SMBALERT in Section 32.6.3.10 “SMBus Mode” and Section “SMBus Mode”
	Section 33. “Inter-IC Sound Controller (I2SC)” Section 33.6.3 “Master, Controller and Slave Modes”: updated last sentence Section 33.8.2 “Inter-IC Sound Controller Mode Register”: updated first sentence below bitmap
	Section 34. “Pulse Density Modulation Interface Controller (PDMIC)” Section 34.6 “Functional Description”: removed section “Buffer Structure” Section 34.6.2.6 “Gain and Offset Compensation”: updated “dgain” bullet Section 34.7.3 “PDMIC Converted Data Register”: updated DATA field description Section 34.7.8 “PDMIC DSP Configuration Register 0”: updated OSR field description
	Section 36. “USB Host Port (UHP)” Section 36.1 “Description”: changed content of third paragraph Added Section 36.2 “Embedded Characteristics” Section 36.3 “Block Diagram”: deleted warning referencing connecting a pull-down to DP; deleted content about port overcurrent protection Updated Section 36.6 “Typical Connection” Added address for Section 36.7.22 “UHP HC Port 1 Status and Control Register” and Section 36.7.23 “UHP HC Port 2 Status and Control Register”

Table 44-1. SAM G55 Datasheet Rev. 11289F Revision History (Continued)

Issue Date	Changes
27-May-16	Section 37. “USB Device Port (UDP)” Updated Section 37.4.1 “I/O Lines” Section 37.5 “Typical Connection”: removed second figure “Board Schematic to Interface Device Peripheral” Updated Section 37.5.2 “VBUS Monitoring”
	Section 38. “Analog-to-Digital Converter (ADC)” Section 38.1 “Description”: at end of section, deleted paragraphs referencing digital error correction circuit and ADC timings Updated Section 38.2 “Embedded Characteristics” Figure 38-1 “Analog-to-Digital Converter Block Diagram”: added captions “all channels trigger” and “last channel trigger” Section 38.5 “Product Dependencies”: - in Section 38.5.3 “I/O Lines”, “ADC_ADTRG” corrected to “ADTRG” - deleted sections “Timer Triggers” and “Conversion Performances” - added Section 38.5.4 “Hardware Triggers” Updated Section 38.6.1 “Analog-to-Digital Conversion” Updated Section 38.6.4 “Conversion Resolution” Updated Section 38.6.6 “Conversion Triggers” Section 38.6.8 “Comparison Window”: instance of ADC_SR corrected to ADC_ISR Replaced section “Differential Inputs” with Section 38.6.9 “Differential and Single-ended Input Modes” Removed section “Input Offset” Updated Section 38.6.10 “ADC Timings” Updated Section 38.6.11 “Last Channel Specific Measurement Trigger” Revised Section 38.6.12 “Enhanced Resolution Mode and Digital Averaging Function” Updated Section 38.6.13 “Asynchronous and Partial Wakeup (SleepWalking)” Table 38-7 “Register Mapping”: renamed “Channel Offset Register” to “Channel Differential Input Register” Section 38.7.2 “ADC Mode Register”: updated TRACKTIM field description. Section 38.7.12 “ADC Last Channel Trigger Mode Register”: updated CMPMOD field description Section 38.7.15 “ADC Extended Mode Register”: updated CMPMODE field description Section 38.7.17 “ADC Channel Differential Input Register”: changed title (was “ADC Channel Offset Register”) removed OFFx bits; updated DIFFx bit description Section 38.7.19 “ADC Analog Control Register”: added address; updated AUTOTEST field description
	Section 39. “Electrical Characteristics” Table 39-3 “DC Characteristics”: updated note on using PA21–PA22 (USB pads) Updated Figure 39-4 “Backup Mode Measurement Setup” Updated Figure 39-6 “Wait Mode Measurement Setup (Dual Supply)” Removed column “FS (kSps)” from Table 39-31 “ADC Resolution Following Digital Averaging (Typical Value) at 1.8V”, Table 39-32 “ADC Resolution Following Digital Averaging (Typical Value) at 3.0V”, Table 39-33 “ADC Resolution Following Digital Averaging (Typical Value) at 1.8V” and Table 39-34 “ADC Resolution Following Digital Averaging (Typical Value) at 3.0V” Added Section 39.8.6 “Autotest Values” Added Section 39.9.1 “External Reset” Updated Table 39-43 “I/O Characteristics”
	Section 42. “Ordering Information” Table 42-1 “SAM G55 Ordering Information”: added MRL B ordering codes

Table 44-1. SAM G55 Datasheet Rev. 11289F Revision History (Continued)

Issue Date	Changes
27-May-16	Section 43. “Errata” Added Section 43.2 “Revision B Parts”

Table 44-2. SAM G55 Datasheet Rev. 11289E Revision History

Issue Date	Changes
30-Nov-15	Updated ”Description”. Modified ”Features” (Note in “Core” section & “Up to 48 I/O lines” instead of “Up to 32 I/O lines” in “I/O “section) Updated Figure 2-1 SAM G55 Block Diagram Modified comments on VDDCORE, DM and DP in Table 3-1 “Signal Description List”. Replaced PDMCLK0 with PDMIC_CLK and PDMDAT0 with PDMIC_DAT in Figure 2-1 SAM G55 Block Diagram, Table 3-1 “Signal Description List” and Table 9-3 “Multiplexing on PIO Controller A (PIOA)” Modified note 3 in Table 9-3 “Multiplexing on PIO Controller A (PIOA)” and note 2 in Table 9-4 “Multiplexing on PIO Controller B (PIOB)”.
	Modified Section 5.1 “Power Supplies” Added Section 5.2.1 “VDDIO Versus VDDCORE” Modified Section 5.3 “Voltage Regulator” and Section 5.4 “Typical Powering Schematics” Updated Section 5.5.2 “Backup Mode”, Section 5.5.3 “Wait Mode” and Section 5.5.5 “Low-power Mode Configuration Summary”
	Removed row for chip name “SAM G55N19” in Table 12-1 “SAM G55 Chip ID Registers” Removed 100-lead version in ARCH description (Section 12.3.1 “Chip ID Register”) Modified Section 7. “Bootloader” Modified Section 8.1 “Internal SRAM”, Section 8.3.1.5 “Unique Identifier” and Section 8.3.1.8 “Calibration Bits” Added DM and DP in Table 9-6 “System I/O Configuration Pin List”
	Section 15. ”Bus Matrix (MATRIX)” Section 15.3 ”Master/Slave Management”: in tables, harmonized naming for masters Section 15.8 ”Register Write Protection”: updated text about clearing the WPVS flag Section 15.9.9 ”Bus Matrix Write Protection Status Register”: updated description of bit WPVS
	Section 17. ”Clock Generator” Modified Figure 17.2 “Embedded Characteristics” Figure 17-1 “Clock Generator Block Diagram”: updated Clock Generator circuitry Updated Section 17.5.7 ”Switching Main Clock between the RC Oscillator and the Crystal Oscillator” Updated Section 17.5.6 ”Main Clock Frequency Counter”
	Section 18. ”Power Management Controller (PMC)” Figure 18-1 “General Clock Block Diagram”: updated Clock Generator circuitry Added information on 32768Hz crystal oscillator frequency monitor: updated Section 18.2 ”Embedded Characteristics”, added Section 18.16 ”32768 Hz Crystal Oscillator Frequency Monitor”, added XT32KFME bit in Section 18.20.7 ”PMC Clock Generator Main Oscillator Register”, added XT32KERR in Section 18.20.14 ”PMC Interrupt Enable Register”, Section 18.19 ”Register Write Protection”: added “PMC Clock Generator Main Clock Frequency Register” to list of protectable registers Section 18.20.15 ”PMC Interrupt Disable Register” and Section 18.20.16 ”PMC Status Register”. Section 18.20.7 ”PMC Clock Generator Main Oscillator Register”: updated notes in CFDEN bit description. Section 18.20.8 ”PMC Clock Generator Main Clock Frequency Register”: updated MAINF bit description Section 18.13 ”Fast Startup”: added one step and inserted warning

Table 44-2. SAM G55 Datasheet Rev. 11289E Revision History (Continued)

Issue Date	Changes
30-Nov-15	Section 20. ”Watchdog Timer (WDT)”
	Updated Section 20.5.3 ”Watchdog Timer Status Register”
	Section 20.5.2 ”Watchdog Timer Mode Register”: updated note on modification of WDT_MR values.
	Added “When the WDDIS bit is set, the fields WDV and WDD must not be modified.” in Section 20.4 ”Functional Description” and Section 20.5.2 ”Watchdog Timer Mode Register” (WDDIS bit description)
	Replaced “Idle mode” with “Sleep mode (Idle mode)” in Section 20.1 ”Description”and with “Sleep mode” in Section 20.4 ”Functional Description”
	Section 23. ”Enhanced Embedded Flash Controller (EEFC)”
	Section 23.5.1 ”EEFC Flash Mode Register”: added missing cross-reference to Section 6.5 ”EEFC Write Protection Mode Register”
	Modified Section 23.2 ”Embedded Characteristics”
	Added Figure 23-1 Flash Memory Areas
	Section 24. ”Timer Counter (TC)”
	Section 24.1 ”Description”: modified first paragraph
	Section 24.7.3 ”TC Channel Mode Register: Waveform Mode”: in ‘Name’ line, replaced “(WAVE = 1)” with “(WAVEFORM_MODE)
	Moved Table 24-1 “Timer Counter Clock Assignment” from Section 24.1 ”Description” to Section 24.3 ”Block Diagram”
	Section 24.5.2 ”Power Management” added “of each channel” after "to enable the Timer Counter clock"
	Section 24.5.3 ”Interrupt Sources”: added “per channel” after "The TC has an interrupt line"
	Section 24.6.16 ”Register Write Protection”: added "The Timer Counter clock of the first channel must be enabled to access TC_WPMR."
	Section 24.7.16 ”TC Write Protection Mode Register” updated WPEN bit description
	Section 26. ”Supply Controller (SUPC)”
	Table 26-2 “Register Mapping”: Modified SUPC_PWMR reset value (0x00FF_180A instead of 0x0000_0000)
	Added VDDSEL and VRVDD in Section 26.5.5 ”Supply Controller Mode Register”
	Modified Section 26.5.9 ”Supply Controller Power Mode Register”
	Section 26. ”Real-time Clock (RTC)”
	Updated Section 26.1 ”Description”“ and Section 26.5 ”Functional Description”(removed references to the 20th century)
	Updated Section 26.5.7 ”RTC Accurate Clock Calibration”
	Updated Section 26.2 ”Embedded Characteristics”.
	Deleted “All non-significant bits read zero.” from the following registers:
	- Section 26.6.3 ”RTC Time Register”
	- Section 26.6.4 ”RTC Calendar Register”
	Figure 26-4 “Calibration Circuitry Waveforms”: corrected two instances of “3,906 ms” to “3.906 ms”
	Section 26-2 ”Register Mapping”: added offset 0xCC as reserved
	Section 26.6.1 ”RTC Control Register”: updated descriptions of value ‘0’ for bits UPDTIM and UPDCA
	Section 28. ”General Purpose Backup Registers (GPBR)”
	Rephrased partly Section 2. ”Description” and Section 3. ”Embedded Characteristics”
	Section 30. ”Flexible Serial Communication Controller (FLEXCOM)”
	Added Table 30-2 “FLEXCOM configuration”

Table 44-2. SAM G55 Datasheet Rev. 11289E Revision History (Continued)

Issue Date	Changes
30-Nov-15	Section 30. “Universal Synchronous Asynchronous Receiver Transceiver (USART)”
	Removed references to RS485 mode
	Section 30.5.1 “I/O Lines”: removed second paragraph (“To prevent the TXD line from falling... must also be enabled
	Section 30.7.17 ”USART Channel Status Register”, Section 30.7.18 ”USART Channel Status Register (SPI_MODE)”, Section 30.7.19 ”USART Channel Status Register (LIN_MODE)”: corrected description of bit “ENDRX: End of RX Buffer” (“has not reached” replaced with “has reached” for 1:)
	Section 30.6.10.5 ”Character Transmission”: after first paragraph, inserted new paragraph “The chip select line is de-asserted for a period equivalent to three bits between the transmission of two data.”
	Section 30.6.8 ”USART Comparison Function on Received Character”: corrected instance of “US_RDR” to “US_RHR”
	Figure 30-39 Receive Holding Register Management: updated to add RHR details
	Section 30.7.1 “USART Control Register”: updated RTSEN and RTSDIS bit descriptions
	Section 30.7.23 ”USART Baud Rate Generator Register”: restructured equations in CD field description
	Section 30.7.25 ”USART Transmitter Timeguard Register”: restructured equations in TG field description
	Added NSSE bit in Section 30.7.6 “USART Interrupt Enable Register (SPI_MODE)”, Section 30.7.10 ”USART Interrupt Disable Register (SPI_MODE)”, Section 30.7.14 ”USART Interrupt Mask Register (SPI_MODE)” and
	Section 30.7.18 ”USART Channel Status Register (SPI_MODE)”
	Added NSS bit in Section 30.7.18 ”USART Channel Status Register (SPI_MODE)”
	Added Warning in Section 30.6.1.2 ”Fractional Baud Rate in Asynchronous Mode” and Section 30.7.23 ”USART Baud Rate Generator Register”
	Added Figure 30-28 RTS line software control when US_MR.USART_MODE = 2”
	Updated USART_MODE desciption in Section 30.7.3 “USART Mode Register”
	Added CLKO bit in Section 30.7.4 “USART Mode Register (SPI_MODE)”
	Updated RTSDIS description in Section 30.7.1 “USART Control Register”
	Updated Section 30.6.11.8 ”Slave Node Synchronization” (modified Oversampling parameter:
	“OVER = 0 => 16X or OVER = 1 => 8X” instead of “Over=0 => 16X or Over=0 => 8X”)

Table 44-2. SAM G55 Datasheet Rev. 11289E Revision History (Continued)

Issue Date	Changes
30-Nov-15	Section 32. "Two-wire Interface (TWI)" Between Section 32.3 "Block Diagram"and Section 32.4 "I/O Lines Description", removed section "Application Block Diagram" Section 32.6.3 "Master Mode" and Section 32.6.5 "Slave Modes": removed subsection "Application Block Diagram" Section 32.7.6 "TWI Status Register": in TXRDY bit description, added links to figures for "TXRDY behavior in Master mode" Updated Section 32.6.5.6 "Alternative Command" Section 32.2 "Embedded Characteristics": added note "See Table 2-1 for details on compatibility with I²C Standard." Section 32.5.2 "Power Management": changed "FX_MR" to " FLEXCOM Mode Regiser (FLEX_MR)" Section 32.5.3 "Interrupt Sources": corrected "SPI interrupt line" to "TWI interrupt line"; corrected "FX_MR" to "FLEX_MR" Table 32-5 "Register Mapping": removed reset value for TWI_THR (register is write-only); added reset value for TWI_ACR Section 32.6.3.4 "Master Transmitter Mode": added note describing how TXRDY flag can be cleared Section "Read Sequence": added note describing how TXRDY flag can be cleared Section 32.6.3.2 "Programming Master Mode": added register names Section 32.6.3.10 "SMBus Mode": replaced instance of "will send/check the PEC field" with "will send/check the PEC field of the TWI frame" Updated Table 32-2, "I/O Lines Description" Section 32.7.5 "TWI Clock Waveform Generator Register": renamed SCL to TWCK in descriptions of fields CLDIV, CHDIV, and CKDIV Section 32.7.6 "TWI Status Register": updated descriptions of bits SCL and SDA Updated Section 32.6.5.5 "High-Speed Slave Mode" Removed CKSRC bit description in Section 32.7.5 "TWI Clock Waveform Generator Register"
	Section 33. "Inter-IC Sound Controller (I2SC)" Updated explanations regarding calculation of the master clock frequency in Section 33.6.5 "Serial Clock and Word Select Generation" and in IMCKDIV and IMCKFS field descriptions in Section 33.8.2 "Inter-IC Sound Controller Mode Register" Section 33.8.2 "Inter-IC Sound Controller Mode Register": Modified IWS description and FORMAT field description and added note 2 to MCKDIV field description
	Section 34. "Pulse Density Modulation Interface Controller (PDMIC)" Section 34.7.7 "PDMIC Interrupt Status Register": added "(cleared by writing PDMIC_RCR or PDMIC_RNCR)" to descriptions of bits ENDRX and RXBUF Table 34-4 "Register Mapping": added row for reserved offset range 0x08–0x10 Updated Section 34.7.7 "PDMIC Interrupt Status Register" PDMDAT' renamed to 'PDMIC_DAT' and 'PDMCLK' renamed to 'PDMIC_CLK' in Section 34.3 "Block Diagram", Table 34-1 "PDMIC Pin Description", Section 34.1 "Description", Section 34.6.1.3 "Restrictions" and Section 34.7.2 "PDMIC Mode Register"

Table 44-2. SAM G55 Datasheet Rev. 11289E Revision History (Continued)

Issue Date	Changes
30-Nov-15	Section 37. ”USB Device Port (UDP)”
	Section ”Using Endpoints With Ping-pong Attribute”: Replaced Bank 0 with Bank 1 in step 12
	Section 37.7.10 ”UDP Endpoint Control and Status Register (CONTROL_BULK)”: below warning, inserted code
	Section 37.7.7 ”UDP Interrupt Status Register”: in EPxINT bit description, corrected two instances of “Endpoint0 Interrupt” to “Endpointx Interrupt”
	Section 37.7.10 ”UDP Endpoint Control and Status Register (CONTROL_BULK)”: reorganized order of EPTYPE field configuration values and added value ‘4’ (reserved)
	Section 37.7.11 ”UDP Endpoint Control and Status Register (ISOCHRONOUS)”: reorganized order of EPTYPE field configuration values and added value ‘4’ (reserved)
	Modified Section 37.4.1 ”I/O Lines”
	Section 39. ”Analog-to-Digital Converter (ADC)”
	Renamed “Hold time” to “Transfer time”
	Section 39.2 ”Embedded Characteristics”: replaced “Hz” with “sps” in Conversion Rate Characteristics: replaced “Hz” with “sps” in Conversion Rate Characteristics
	In text below Section 39-8 ”Independent Trigger Measurement for Last Channel (ADC_CHSR[LCI] = 0 and ADC_MR.TRGEN = 1)”, replaced “rate defined by the RTC OUT1 field” with “rate defined by the RTCOUT1”
	Figure 39-9 Only Last Channel Measurement Triggered at Low Speed (ADC_CHSR[LCI] = 0 and ADC_MR.TRGEN = 0): replaced “RTCOUT1 field” with “RTCOUT
	Revised Section 39.6.1 ”Analog-to-Digital Conversion”
	Added Section 39.6.2 ”ADC Clock”
	Updated Section 39.6.3 ”ADC Reference Voltage” and changed title (was “Conversion Reference”)
	Section 39.7.2 ”ADC Mode Register”: updated description of ‘TRANSFER’’ field
	Section 39.7.4 ”ADC Channel Enable Register”: updated CHx field description
	Removed Section “Analog Inputs”
	Modified Section 39.5.3 ”I/O Lines”
	Modified information about USCHx fields in:
	- Section 39.6.7 ”Sleep Mode and Conversion Sequencer”
	- Section 39.7.3 ”ADC Channel Sequence 1 Register”
	Updated Section 39.6.12 ”Last Channel Specific Measurement Trigger”
	Updated Section 39-7 ”Same Trigger for All Channels (ADC_CHSR[LCI] = 1 and ADC_MR.TRGEN = 1)”, Section 39-8 ”Independent Trigger Measurement for Last Channel (ADC_CHSR[LCI] = 0 and ADC_MR.TRGEN = 1)” and Section 39-9 ”Only Last Channel Measurement Triggered at Low Speed (ADC_CHSR[LCI] = 0 and ADC_MR.TRGEN = 0)”
	Section 39. “Electrical Characteristics”
	Updated Table 39-2 “Recommended Operating Conditions”, Table 39-3 “DC Characteristics”, Table 39-4 “VDDCORE Voltage Regulator Characteristics”
	Modified Section 39.4.1 “Backup Mode Current Consumption”, Section 39.4.2 “Wait Mode Power Consumption”, Section 39.4.2.2 “Wait Mode Dual Supply”, Section 39.4.4 “Active Mode Power Consumption”, Section 39.6 “PLLA Characteristics”, Section 39.7 “PLLB Characteristics”, Section 39.5.2 “8/16/24 MHz RC Oscillators Characteristics”, Section 39.5.6 “3 to 20 MHz Crystal Oscillator Characteristics”, Table 39-29 “Analog Power Supply Characteristics”, Section 39.8.4 “External Reference Voltage”, Table 39-44 “PDM Characteristics”, Section 39.9.4 “SPI Characteristics”, Section 39.9.4.1 “Maximum SPI Frequency”, Section 39.9.4.2 “SPI Timings”, Section 39.9.5.1 “USART SPI TImings” (Table 39-44 title)
	Added Section 39.5.5 “32.768 kHz XIN32 Clock Input Characteristics in Bypass Mode”
	Section 43. “Errata”
	Added Section Issue: “No Write Protection On SUPC_WUIR”, Section Issue: “PMC Asynchronous Wakeup”, Section 43.1.5 “TWI/TWIHS” and Section 43.1.6 “CMCC”

Table 44-3. SAM G55 Datasheet Rev. 11289D Revision History

Issue Date	Changes
15-Jun-15	Removed “Preliminary” marking.
	Modified Section ”Description”
	Updated Figure 2-1 “SAM G55 Block Diagram”(GPBR)
	Added note to PB0/PB15 in Table 3-1 “Signal Description List”
	Added note to Section 4.2.1 “64-lead QFN / LQFP Pinout”
	Modified “Flash Wait States” information in Section 5.5.3 “Wait Mode”
	Replaced “Processor and architecture” section with Section 6. “Product Mapping” (removed redundant information) Modified Figure 6-1 SAM G55 Product Mapping (information added in “Boot Memory”)
	Section 7. “Bootloader” Updated section
	Section 8. “Memories” Modified Section 8.3.1.5 “Unique Identifier”
	Section 9. “Peripherals” Added notes to Extra functions in Table 9-3 “Multiplexing on PIO Controller A (PIOA)” and Table 9-4 “Multiplexing on PIO Controller B (PIOB)” Added Section 9.3 ”System I/O Lines”
	Section 10. “Real-time Event Management” Replaced AFEC with ADC in Table 10-1, “Real-time Event Mapping List”
	Section 11. “Debug and Test Features” Added Section 11.6.2 “NRST Pin” and Section 11.6.3 “ERASE Pin”
	Section 38. “Analog-to-Digital Converter (ADC)” Modified conversion rate in Section 38.2 “Embedded Characteristics”
	Section 39. “Electrical Characteristics” Updated Table 39-20, “8/16/24 MHz RC Oscillator Characteristics” and Section 39.4.2 “Wait Mode Power Consumption” Section 39.4.4 “Active Mode Power Consumption”: removed table “Typical Current Consumption Running at V_{VDDIO} = 1.8V and $V_{VDDCORE}$ = default settings” and text. Table 39-3, “DC Characteristics”: added reference to REXT in note 1. Modified f_{IN} min value in Table 39-27, “PLLA Characteristics” and Table 39-28, “PLLB Characteristics” Updated Section 39.5.2 “8/16/24 MHz RC Oscillators Characteristics” Removed note on bit LOWRES in Table 39-31, “ADC Resolution Following Digital Averaging (Typical Value) at 1.8V”, Table 39-32, “ADC Resolution Following Digital Averaging (Typical Value) at 3.0V”, Table 39-33, “ADC Resolution Following Digital Averaging (Typical Value) at 1.8V”, Table 39-34, “ADC Resolution Following Digital Averaging (Typical Value) at 3.0V”
	Section 40. “Mechanical Characteristics” Updated Table 40-6 “64-lead QFN Package Characteristics”
	Section 42. “Ordering Information” Replaced ATSAMG55J19-A-AUT with ATSAMG55J19A-AUT in Table 42-1 “SAM G55 Ordering Information”.
	Section 43. “Errata” Modified Section Issue: “INL and DNL values at 25°C”: added temperature of 25°C. Added new Section Issue: “INL and DNL values from 40°C to 85°C”.

Table 44-4. SAM G55 Datasheet Rev. 11289C Revision History

Issue Date	Changes
14-Jan-15	Added “Preliminary Status” marking.

Table 44-5. SAM G55 Datasheet Rev. 11289B Revision History

Issue Date	Changes
13-Jan-15	Section 5. “Power Considerations” Added Section 5.2 “Powerup Considerations”
	Section 18. “Power Management Controller (PMC)” Section 18.19.18 “PMC Fast Startup Mode Register”: added bit FFLPM (Force Flash Low-power Mode)
	Section 39. “Electrical Characteristics” Updated and restructured Section 39.4.4 “Active Mode Power Consumption”. Updated Table 39-20 “8/16/24 MHz RC Oscillator Characteristics”.

Table 44-6. SAM G55 Datasheet Rev. 11289A Revision History

Issue Date	Changes
19-Dec-14	First issue.

Table of Contents

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**