# SAM3N Series

## Atmel | SMART ARM-based MCU

**DATASHEET**

## Description

The Atmel® | SMART SAM3N series is a member of a family of Flash microcontrollers based on the high performance 32-bit ARM® Cortex®-M3 RISC processor. It operates at a maximum speed of 48 MHz and features up to 256 Kbytes of Flash and up to 24 Kbytes of SRAM. The peripheral set includes 2 USARTs, 2 UARTs, 2 TWIs, 3 SPIs, as well as a PWM timer, two 3-channel general-purpose 16-bit timers, an RTC, a 10-bit ADC, and a 10-bit DAC.

The SAM3N devices have three software-selectable low-power modes: Sleep, Wait and Backup. In Sleep mode, the processor is stopped while all other functions can be kept running. In Wait mode, all clocks and functions are stopped but some peripherals can be configured to wake up the system based on predefined conditions. In Backup mode, only the RTC, RTT, 256-bit GPBR, and wake-up logic are running.

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

The SAM3N series is ready for capacitive touch thanks to the Atmel QTouch® library, offering an easy way to implement buttons, wheels and sliders.

The SAM3N device is an entry-level general purpose microcontroller. That makes the SAM3N the ideal starting point to move from 8-/16-bit to 32-bit microcontrollers.

It operates from 1.62V to 3.6V and is available in 48-pin, 64-pin and 100-pin QFP, 48-pin and 64-pin QFN, and 100-pin BGA packages.

The SAM3N series is the ideal migration path from the SAM3S for applications that require a reduced BOM cost. The SAM3N series is pin-to-pin compatible with the SAM3S series. Its aggressive price point and high level of integration pushes its scope of use far into cost-sensitive, high-volume applications.

# 1. Features

- Core
  - ARM Cortex-M3 revision 2.0 running at up to 48 MHz
  - Thumb®-2 Instruction Set
  - 24-bit SysTick Counter
  - Nested Vector Interrupt Controller
- Pin-to-pin compatible with SAM7S legacy products (48/64-pin versions) and SAM3S (48/64/100-pin versions)
- Memories
  - From 16 to 256 Kbytes embedded Flash, 128-bit wide access, memory accelerator, single plane
  - From 4 to 24 Kbytes embedded SRAM
  - 16 Kbytes ROM with embedded bootloader routines (UART) and IAP routines
- System
  - Embedded voltage regulator for single supply operation
  - Power-on-Reset (POR), Brown-out Detector (BOD) and Watchdog for safe operation
  - Quartz or ceramic resonator oscillators: 3 to 20 MHz main power with Failure Detection and optional low power 32.768 kHz for RTC or device clock
  - High precision 8/12 MHz factory trimmed internal RC oscillator with 4 MHz default frequency for device startup. In-application trimming access for frequency adjustment
  - Slow Clock Internal RC oscillator as permanent low-power mode device clock
  - One PLL up to 130 MHz for device clock
  - Up to 10 Peripheral DMA (PDC) channels
- Low Power Modes
  - Sleep, Wait, and Backup modes, down to 1.2 µA in Backup mode with RTC, RTT, and 256-bit GPBR
- Peripherals
  - Up to 2 USARTs with RS-485 and SPI mode support. One USART (USART0) has ISO7816, IrDA® and PDC support in addition
  - Two 2-wire UARTs
  - Two 2-wire Interfaces (I2C compatible)
  - One SPI
  - Up to two 3-channel 16-bit Timer Counters with capture, waveform, compare and PWM mode, Quadrature Decoder Logic and 2-bit Gray Up/Down Counter for Stepper Motor
  - 4-channel 16-bit PWM
  - 32-bit low-power Real-time Timer (RTT)
  - Low-power Real-time Clock (RTC) with calendar and alarm features
  - Up to 16 channels, 384 ksps 10-bit ADC
  - One 500 ksps 10-bit DAC
  - Register Write Protection
- I/O
  - Up to 79 I/O lines with external interrupt capability (edge or level sensitivity), debouncing, glitch filtering and on-die Series Resistor Termination
  - Three 32-bit Parallel Input/Output Controllers
- Packages
  - 100-lead LQFP – 14 x 14 mm, pitch 0.5 mm
  - 100-ball TFBGA – 9 x 9 mm, pitch 0.8 mm
  - 64-lead LQFP – 10 x 10 mm, pitch 0.5 mm
  - 64-pad QFN – 9 x 9 mm, pitch 0.5 mm
  - 48-lead LQFP – 7 x 7 mm, pitch 0.5 mm
  - 48-pad QFN – 7 x 7 mm, pitch 0.5 mm

## 1.1 Configuration Summary

The SAM3N series devices differ in memory size, package and features list. Table 1-1 summarizes the configurations.

**Table 1-1. Configuration Summary**

| Device | Flash (Kbytes) | SRAM (Kbytes) | Package | Number of PIOs | ADC Channels | Timer Channels | PDC Channels | USART | DAC |
|---|---|---|---|---|---|---|---|---|---|
| SAM3N4A | 256 | 24 | LQFP48<br>QFN48 | 34 | 8 | 6(1) | 8 | 1 | _ |
| SAM3N4B | 256 | 24 | LQFP64<br>QFN64 | 47 | 10 | 6(2) | 10 | 2 | 1 |
| SAM3N4C | 256 | 24 | LQFP100<br>BGA100 | 79 | 16 | 6 | 10 | 2 | 1 |
| SAM3N2A | 128 | 16 | LQFP48<br>QFN48 | 34 | 8 | 6(1) | 8 | 1 | _ |
| SAM3N2B | 128 | 16 | LQFP64<br>QFN64 | 47 | 10 | 6(2) | 10 | 2 | 1 |
| SAM3N2C | 128 | 16 | LQFP100<br>BGA100 | 79 | 16 | 6 | 10 | 2 | 1 |
| SAM3N1A | 64 | 8 | LQFP48<br>QFN48 | 34 | 8 | 6(1) | 8 | 1 | _ |
| SAM3N1B | 64 | 8 | LQFP64<br>QFN64 | 47 | 10 | 6(2) | 10 | 2 | 1 |
| SAM3N1C | 64 | 8 | LQFP100<br>BGA100 | 79 | 16 | 6 | 10 | 2 | 1 |
| SAM3N0A | 32 | 8 | LQFP48<br>QFN48 | 34 | 8 | 6(1) | 8 | 1 | _ |
| SAM3N0B | 32 | 8 | LQFP64<br>QFN64 | 47 | 10 | 6(2) | 10 | 2 | 1 |
| SAM3N0C | 32 | 8 | LQFP100<br>BGA100 | 79 | 16 | 6 | 10 | 2 | 1 |
| SAM3N00A | 16 | 4 | LQFP48<br>QFN48 | 34 | 8 | 6(1) | 8 | 1 | _ |
| SAM3N00B | 16 | 4 | LQFP64<br>QFN64 | 47 | 10 | 6(2) | 10 | 2 | 1 |

Notes: 1. Only two TC channels are accessible through the PIO.
2. Only three TC channels are accessible through the PIO.

# 2. SAM3N Block Diagram

**Figure 2-1. SAM3N 100-pin version Block Diagram**

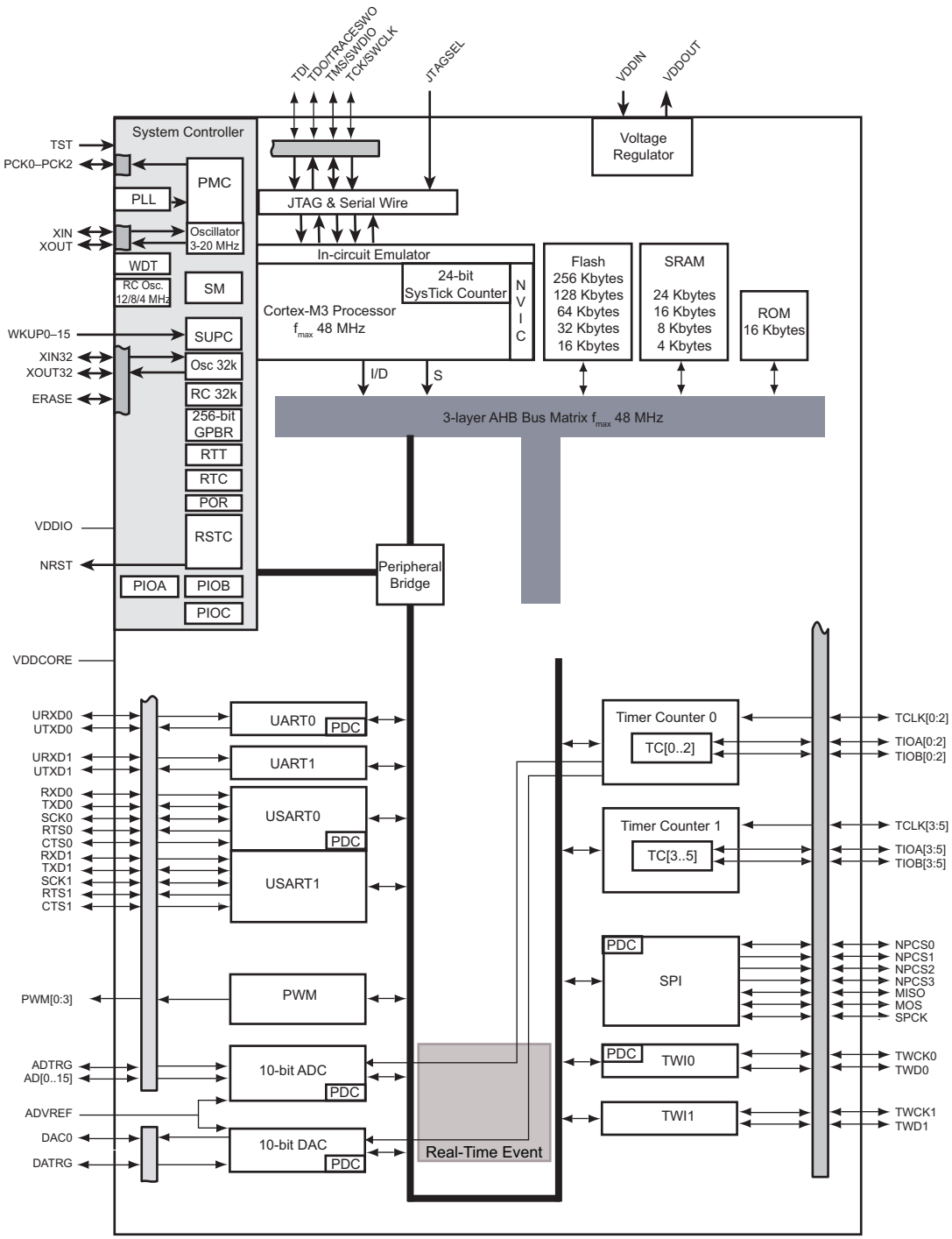


**Figure 2-2. SAM3N 64-pin version Block Diagram**

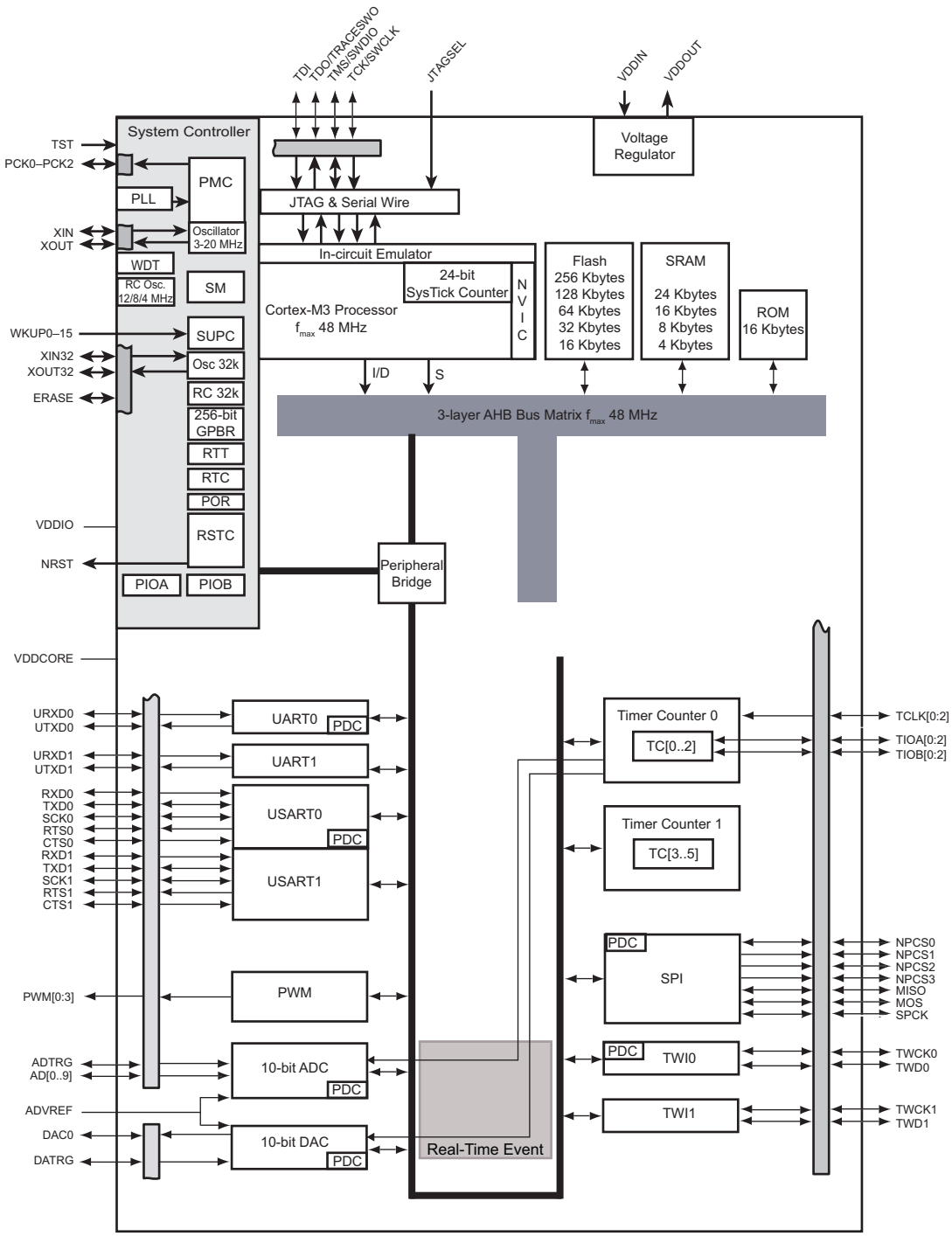
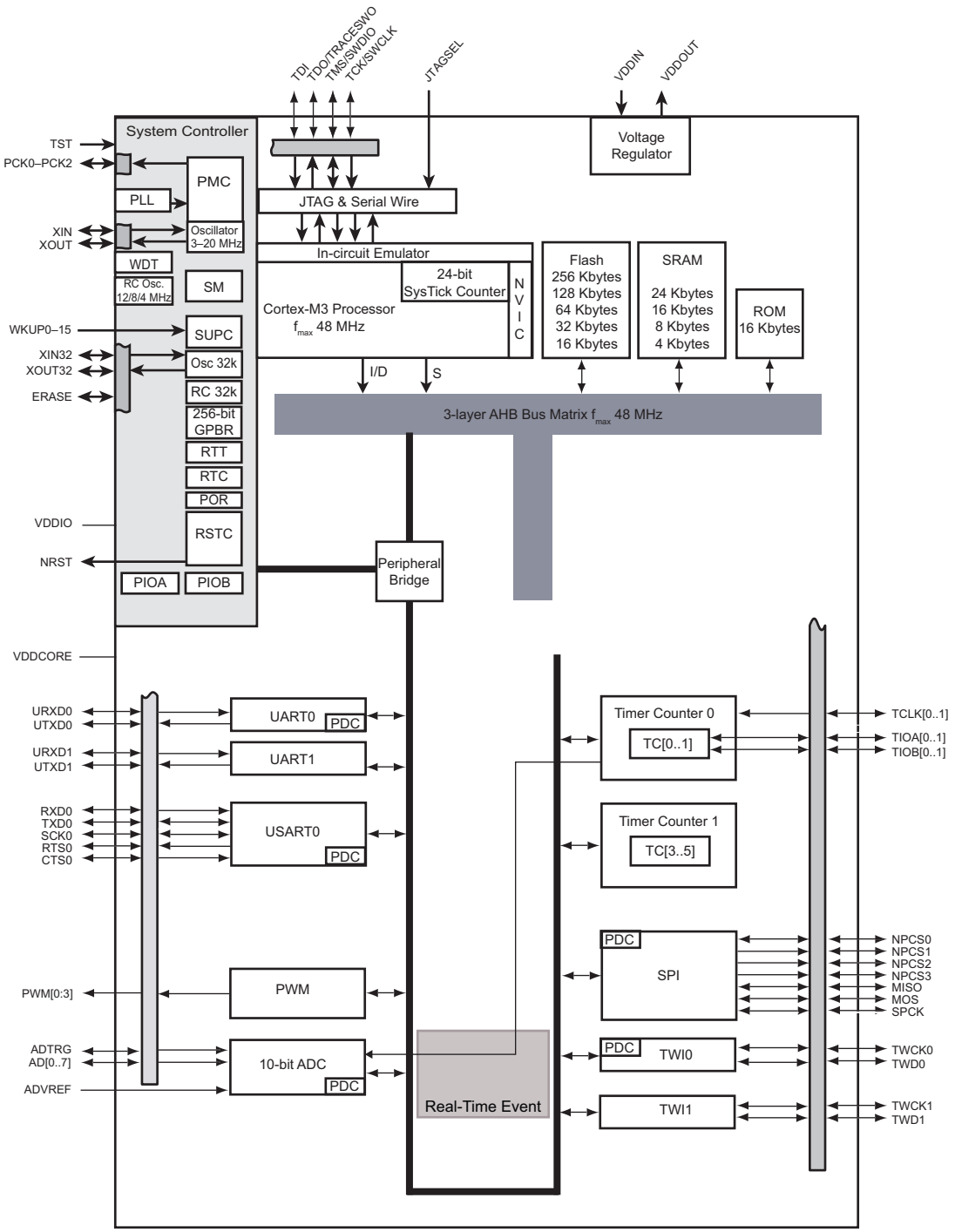


**Figure 2-3. SAM3N 48-pin version Block Diagram**



# 3. Signal Description

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

**Table 3-1. Signal Description List**

| Signal Name | Function | Type | Active Level | Voltage Reference | Comments |
|---|---|---|---|---|---|
| **Power Supplies** | | | | | |
| VDDIO | Peripherals I/O Lines Power Supply | Power | | | 1.62V to 3.6V |
| VDDIN | Voltage Regulator, ADC and DAC Power Supply | Power | | | 1.8V to 3.6V(3) |
| VDDOUT | Voltage Regulator Output | Power | | | 1.8V Output |
| VDDPLL | Oscillator and PLL Power Supply | Power | | | 1.65 V to 1.95V |
| VDDCORE | Power the core, the embedded memories and the peripherals | Power | | | 1.65V to 1.95V<br>Connected externally to VDDOUT |
| GND | Ground | Ground | | | |
| **Clocks, Oscillators and PLLs** | | | | | |
| XIN | Main Oscillator Input | Input | | VDDIO | Reset State:<br>- PIO Input<br>- Internal Pull-up disabled(4)<br>- Schmitt Trigger enabled(1) |
| XOUT | Main Oscillator Output | Output | | | |
| XIN32 | Slow Clock Oscillator Input | Input | | | |
| XOUT32 | Slow Clock Oscillator Output | Output | | | |
| PCK0–PCK2 | Programmable Clock Output | Output | | | Reset State:<br>- PIO Input<br>- Internal Pull-up enabled<br>- Schmitt Trigger enabled(1) |
| **ICE and JTAG** | | | | | |
| TCK/SWCLK | Test Clock/Serial Wire Clock | Input | | VDDIO | Reset State:<br>- SWJ-DP Mode<br>- Internal pull-up disabled(1)<br>- Schmitt Trigger enabled(1) |
| TDI | Test Data In | Input | | | |
| TDO/TRACESWO | Test Data Out/Trace Asynchronous Data Out | Output | | | |
| TMS/SWDIO | Test Mode Select /Serial Wire Input/Output | Input / I/O | | | |
| JTAGSEL | JTAG Selection | Input | High | | Permanent Internal pull-down |
| **Flash Memory** | | | | | |
| ERASE | Flash and NVM Configuration Bits Erase Command | Input | High | VDDIO | Reset State:<br>- Erase Input<br>- Internal pull-down enabled<br>- Schmitt Trigger enabled(1) |
| **Reset/Test** | | | | | |
| NRST | Microcontroller Reset | I/O | Low | VDDIO | Permanent Internal pull-up |
| TST | Test Mode Select | Input | | VDDIO | Permanent Internal pull-down |

**Table 3-1. Signal Description List (Continued)**

| Signal Name | Function | Type | Active Level | Voltage Reference | Comments |
|---|---|---|---|---|---|
| **Universal Asynchronous Receiver Transceiver - UARTx** | | | | | |
| URXDx | UART Receive Data | Input | | | |
| UTXDx | UART Transmit Data | Output | | | |
| **PIO Controller - PIOA - PIOB - PIOC** | | | | | |
| PA0–PA31 | Parallel IO Controller A | I/O | | VDDIO | Reset State:<br>- PIO or System IOs(2)<br>- Internal pull-up enabled<br>- Schmitt Trigger enabled(1) |
| PB0–PB14 | Parallel IO Controller B | I/O | | | |
| PC0–PC31 | Parallel IO Controller C | I/O | | | |
| **Universal Synchronous Asynchronous Receiver Transmitter - USARTx** | | | | | |
| SCKx | USARTx Serial Clock | I/O | | | |
| TXDx | USARTx Transmit Data | I/O | | | |
| RXDx | USARTx Receive Data | Input | | | |
| RTSx | USARTx Request To Send | Output | | | |
| CTSx | USARTx Clear To Send | Input | | | |
| **Timer Counter - TC** | | | | | |
| TCLKx | TC Channel x External Clock Input | Input | | | |
| TIOAx | TC Channel x I/O Line A | I/O | | | |
| TIOBx | TC Channel x I/O Line B | I/O | | | |
| **Pulse Width Modulation Controller - PWM** | | | | | |
| PWMx | PWM Waveform Output for channel x | Output | | | |
| **Serial Peripheral Interface - SPI** | | | | | |
| MISO | Master In Slave Out | I/O | | | |
| MOSI | Master Out Slave In | I/O | | | |
| SPCK | SPI Serial Clock | I/O | | | |
| SPI_NPCS0 | SPI Peripheral Chip Select 0 | I/O | Low | | |
| SPI_NPCS1–SPI_NPCS3 | SPI Peripheral Chip Select | Output | Low | | |
| **Two-Wire Interface - TWIx** | | | | | |
| TWDx | TWIx Two-wire Serial Data | I/O | | | |
| TWCKx | TWIx Two-wire Serial Clock | I/O | | | |
| **Analog** | | | | | |
| ADVREF | ADC and DAC Reference | Analog | | | |
| **10-bit Analog-to-Digital Converter - ADC** | | | | | |
| AD0–AD15 | Analog Inputs | Analog | | | |
| ADTRG | ADC Trigger | Input | | VDDIO | |
| **Digital-to-Analog Converter Controller - DACC** | | | | | |
| DAC0 | DACC Channel Analog Output | Analog | | | |
| DATRG | DACC Trigger | Input | | VDDIO | |

**Table 3-1. Signal Description List (Continued)**

| Signal Name | Function | Type | Active Level | Voltage Reference | Comments |
|---|---|---|---|---|---|
| **Fast Flash Programming Interface - FFPI** | | | | | |
| PGMEN0–PGMEN2 | Programming Enabling | Input | | VDDIO | |
| PGMM0–PGMM3 | Programming Mode | Input | | | |
| PGMD0–PGMD15 | Programming Data | I/O | | | |
| PGMRDY | Programming Ready | Output | High | | |
| PGMNVALID | Data Direction | Output | Low | | |
| PGMNOE | Programming Read | Input | Low | | |
| PGMCK | Programming Clock | Input | | | |
| PGMNCMD | Programming Command | Input | Low | | |

Notes:
1. Schmitt triggers can be disabled through PIO registers.
2. Some PIO lines are shared with System IOs.
3. See Section 5.4 “Typical Powering Schematics” for restriction on voltage range of analog cells.
4. TDO pin is set in input mode when the Cortex-M3 Core is not in debug mode. Thus the internal pull-up corresponding to this PIO line must be enabled to avoid current consumption due to floating input.

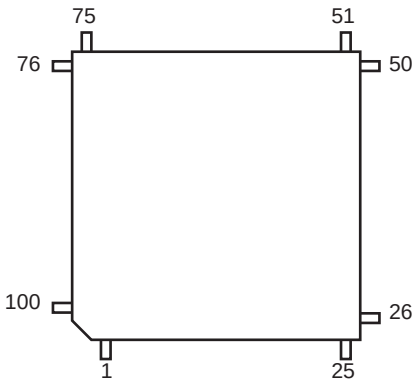
# 4. Package and Pinout

SAM3N4/2/1/0/00 series is pin-to-pin compatible with SAM3S products. Furthermore SAM3N4/2/1/0/00 devices have new functionalities referenced in italic in Table 4-1, Table 4-3 and Table 4-4.

## 4.1 SAM3N4/2/1/0/00C Package and Pinout

### 4.1.1 100-lead LQFP Package Outline

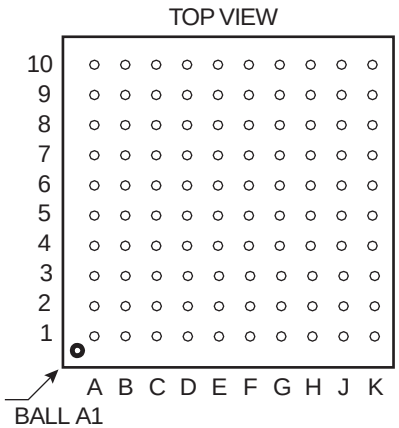
**Figure 4-1. Orientation of the 100-lead LQFP Package**




See Section 37. “Mechanical Characteristics” for mechanical drawings and specifications.

### 4.1.2 100-ball TFBGA Package Outline

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




The 100-ball TFBGA package respects Green Standards.

See Section 37. “Mechanical Characteristics” for mechanical drawings and specifications.

### 4.1.3 100-Lead LQFP Pinout

**Table 4-1. 100-lead LQFP SAM3N4/2/1/0/00C Pinout**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | ADVREF | 26 | GND | 51 | TDI/PB4 | 76 | TDO/TRACESWO/PB5 |
| 2 | GND | 27 | VDDIO | 52 | PA6/PGMNOE | 77 | JTAGSEL |
| 3 | PB0/AD4 | 28 | PA16/PGMD4 | 53 | PA5/PGMRDY | 78 | PC18 |
| 4 | PC29/AD13 | 29 | PC7 | 54 | PC28 | 79 | TMS/SWDIO/PB6 |
| 5 | PB1/AD5 | 30 | PA15/PGMD3 | 55 | PA4/PGMNCMD | 80 | PC19 |
| 6 | PC30/AD14 | 31 | PA14/PGMD2 | 56 | VDDCORE | 81 | PA31 |
| 7 | PB2/AD6 | 32 | PC6 | 57 | PA27 | 82 | PC20 |
| 8 | PC31/AD15 | 33 | PA13/PGMD1 | 58 | PC8 | 83 | TCK/SWCLK/PB7 |
| 9 | PB3/AD7 | 34 | PA24 | 59 | PA28 | 84 | PC21 |
| 10 | VDDIN | 35 | PC5 | 60 | NRST | 85 | VDDCORE |
| 11 | VDDOUT | 36 | VDDCORE | 61 | TST | 86 | PC22 |
| 12 | PA17/PGMD5/AD0 | 37 | PC4 | 62 | PC9 | 87 | ERASE/PB12 |
| 13 | PC26 | 38 | PA25 | 63 | PA29 | 88 | PB10 |
| 14 | PA18/PGMD6/AD1 | 39 | PA26 | 64 | PA30 | 89 | PB11 |
| 15 | PA21/AD8 | 40 | PC3 | 65 | PC10 | 90 | PC23 |
| 16 | VDDCORE | 41 | PA12/PGMD0 | 66 | PA3 | 91 | VDDIO |
| 17 | PC27 | 42 | PA11/PGMM3 | 67 | PA2/PGMEN2 | 92 | PC24 |
| 18 | PA19/PGMD7/AD2 | 43 | PC2 | 68 | PC11 | 93 | PB13/DAC0 |
| 19 | PC15/AD11 | 44 | PA10/PGMM2 | 69 | VDDIO | 94 | PC25 |
| 20 | PA22/AD9 | 45 | GND | 70 | GND | 95 | GND |
| 21 | PC13/AD10 | 46 | PA9/PGMM1 | 71 | PC14 | 96 | PB8/XOUT |
| 22 | PA23 | 47 | PC1 | 72 | PA1/PGMEN1 | 97 | PB9/*PGMCK/XIN* |
| 23 | PC12/AD12 | 48 | PA8/XOUT32/PGMM0 | 73 | PC16 | 98 | VDDIO |
| 24 | PA20/AD3 | 49 | PA7/XIN32/PGMNVALID | 74 | PA0/PGMEN0 | 99 | PB14 |
| 25 | PC0 | 50 | VDDIO | 75 | PC17 | 100 | VDDPLL |

### 4.1.4 100-ball TFBGA Pinout

**Table 4-2. 100-ball TFBGA SAM3N4/2/1/0/00C Pinout**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A1 | PB1 | C6 | PB7 | F1 | PA18/PGMD6 | H6 | PC4 |
| A2 | PC29 | C7 | PC16 | F2 | PC26 | H7 | PA11/PGMM3 |
| A3 | VDDIO | C8 | PA1/PGMEN1 | F3 | VDDOUT | H8 | PC1 |
| A4 | PB9/PGMCK | C9 | PC17 | F4 | GND | H9 | PA6/PGMNOE |
| A5 | PB8 | C10 | PA0/PGMEN | F5 | VDDIO | H10 | PB4 |
| A6 | PB13 | D1 | PB3 | F6 | PA27 | J1 | PC15 |
| A7 | PB11 | D2 | PB0 | F7 | PC8 | J2 | PC0 |
| A8 | PB10 | D3 | PC24 | F8 | PA28 | J3 | PA16/PGMD4 |
| A9 | PB6 | D4 | PC22 | F9 | TST | J4 | PC6 |
| A10 | JTAGSEL | D5 | GND | F10 | PC9 | J5 | PA24 |
| B1 | PC30 | D6 | GND | G1 | PA21 | J6 | PA25 |
| B2 | ADVREF | D7 | VDDCORE | G2 | PC27 | J7 | PA10/PGMM2 |
| B3 | GNDANA | D8 | PA2/PGMEN2 | G3 | PA15/PGMD3 | J8 | GND |
| B4 | PB14 | D9 | PC11 | G4 | VDDCORE | J9 | VDDCORE |
| B5 | PC21 | D10 | PC14 | G5 | VDDCORE | J10 | VDDIO |
| B6 | PC20 | E1 | PA17/PGMD5 | G6 | PA26 | K1 | PA22 |
| B7 | PA31 | E2 | PC31 | G7 | PA12/PGMD0 | K2 | PC13 |
| B8 | PC19 | E3 | VDDIN | G8 | PC28 | K3 | PC12 |
| B9 | PC18 | E4 | GND | G9 | PA4/PGMNCMD | K4 | PA20 |
| B10 | PB5 | E5 | GND | G10 | PA5/PGMRDY | K5 | PC5 |
| C1 | PB2 | E6 | NRST | H1 | PA19/PGMD7 | K6 | PC3 |
| C2 | VDDPLL | E7 | PA29 | H2 | PA23 | K7 | PC2 |
| C3 | PC25 | E8 | PA30 | H3 | PC7 | K8 | PA9/PGMM1 |
| C4 | PC23 | E9 | PC10 | H4 | PA14/PGMD2 | K9 | PA8/PGMM0 |
| C5 | PB12 | E10 | PA3 | H5 | PA13/PGMD1 | K10 | PA7/PGMVALID |

## 4.2 SAM3N4/2/1/0/00B Package and Pinout

**Figure 4-3. Orientation of the 64-pad QFN Package**

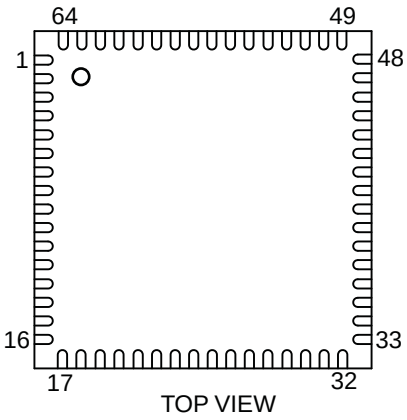
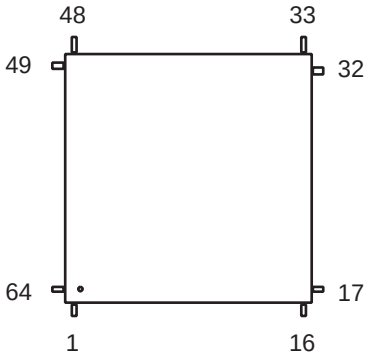



**Figure 4-4. Orientation of the 64-lead LQFP Package**




See Section 37. “Mechanical Characteristics” for mechanical drawings and specifications.

### 4.2.1 64-Lead LQFP and QFN Pinout

64-pin version SAM3N devices are pin-to-pin compatible with SAM3S products. Furthermore, SAM3N products have new functionalities shown in italic in Table 4-3.

**Table 4-3. 64-pin SAM3N4/2/1/0/00B Pinout**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | ADVREF | 17 | GND | 33 | *TDI/PB4* | 49 | *TDO/TRACESWO/PB5* |
| 2 | GND | 18 | VDDIO | 34 | PA6/PGMNOE | 50 | JTAGSEL |
| 3 | *PB0/AD4* | 19 | PA16/PGMD4 | 35 | PA5/PGMRDY | 51 | *TMS/SWDIO/PB6* |
| 4 | *PB1AD5* | 20 | PA15/PGMD3 | 36 | PA4/PGMNCMD | 52 | PA31 |
| 5 | *PB2/AD6* | 21 | PA14/PGMD2 | 37 | PA27/PGMD15 | 53 | *TCK/SWCLK/PB7* |
| 6 | *PB3/AD7* | 22 | PA13/PGMD1 | 38 | PA28 | 54 | VDDCORE |
| 7 | VDDIN | 23 | PA24/PGMD12 | 39 | NRST | 55 | *ERASE/PB12* |
| 8 | VDDOUT | 24 | VDDCORE | 40 | TST | 56 | *PB10* |
| 9 | PA17/PGMD5/AD0 | 25 | PA25/PGMD13 | 41 | PA29 | 57 | *PB11* |
| 10 | PA18/PGMD6/AD1 | 26 | PA26/PGMD14 | 42 | PA30 | 58 | VDDIO |
| 11 | PA21/PGMD9/AD8 | 27 | PA12/PGMD0 | 43 | PA3 | 59 | *PB13/DAC0* |
| 12 | VDDCORE | 28 | PA11/PGMM3 | 44 | PA2/PGMEN2 | 60 | GND |
| 13 | PA19/PGMD7/AD2 | 29 | PA10/PGMM2 | 45 | VDDIO | 61 | *XOUT/PB8* |
| 14 | PA22/PGMD10/AD9 | 30 | PA9/PGMM1 | 46 | GND | 62 | *XIN/PGMCK/PB9* |
| 15 | PA23/PGMD11 | 31 | PA8/XOUT32/PGMM0 | 47 | PA1/PGMEN1 | 63 | *PB14* |
| 16 | PA20/PGMD8/AD3 | 32 | PA7/XIN32/XOUT32/ PGMNVALID | 48 | PA0/PGMEN0 | 64 | VDDPLL |

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

## 4.3 SAM3N4/2/1/0/00A Package and Pinout

**Figure 4-5. Orientation of the 48-pad QFN Package**

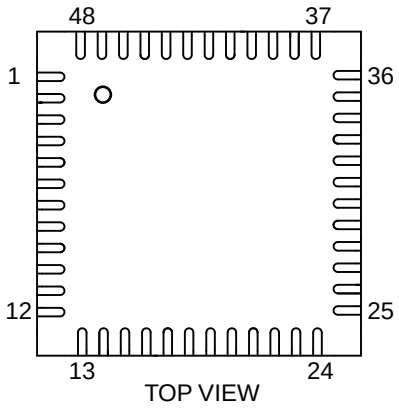
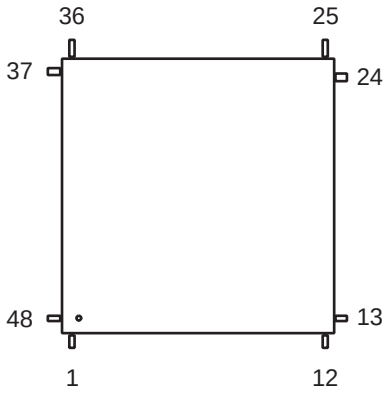



**Figure 4-6. Orientation of the 48-lead LQFP Package**




See Section 37. “Mechanical Characteristics” for mechanical drawings and specifications.

### 4.3.1 48-Lead LQFP and QFN Pinout

**Table 4-4. 48-pin SAM3N4/2/1/0/00A Pinout**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | ADVREF | 13 | VDDIO | 25 | *TDI/PB4* | 37 | *TDO/TRACESWO/PB5* |
| 2 | GND | 14 | PA16/PGMD4 | 26 | PA6/PGMNOE | 38 | JTAGSEL |
| 3 | *PB0/AD4* | 15 | PA15/PGMD3 | 27 | PA5/PGMRDY | 39 | *TMS/SWDIO/PB6* |
| 4 | *PB1/AD5* | 16 | PA14/PGMD2 | 28 | PA4/PGMNCMD | 40 | *TCK/SWCLK/PB7* |
| 5 | *PB2/AD6* | 17 | PA13/PGMD1 | 29 | NRST | 41 | VDDCORE |
| 6 | *PB3/AD7* | 18 | VDDCORE | 30 | TST | 42 | *ERASE/PB12* |
| 7 | VDDIN | 19 | PA12/PGMD0 | 31 | PA3 | 43 | *PB10* |
| 8 | VDDOUT | 20 | PA11/PGMM3 | 32 | PA2/PGMEN2 | 44 | *PB11* |
| 9 | PA17/PGMD5/AD0 | 21 | PA10/PGMM2 | 33 | VDDIO | 45 | *XOUT/PB8* |
| 10 | PA18/PGMD6/AD1 | 22 | PA9/PGMM1 | 34 | GND | 46 | *XIN/P/PB9/GMCK* |
| 11 | PA19/PGMD7/AD2 | 23 | PA8/XOUT32/PGMM0 | 35 | PA1/PGMEN1 | 47 | VDDIO |
| 12 | PA20/AD3 | 24 | PA7/XIN32/PGMNVALID | 36 | PA0/PGMEN0 | 48 | VDDPLL |

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

# 5. Power Considerations

## 5.1 Power Supplies

The SAM3N product has several types of power supply pins:

- VDDCORE pins: Power the core, including the processor, the embedded memories and the peripherals. Voltage ranges from 1.62V to 1.95V.
- VDDIO pins: Power the peripherals I/O lines, backup part, 32 kHz crystal oscillator and oscillator pads. Voltage ranges from 1.62V to 3.6V
- VDDIN pin: Voltage Regulator, ADC and DAC Power Supply. Voltage ranges from 1.8V to 3.6V for the Voltage Regulator.
- VDDPLL pin: Powers the PLL, the Fast RC and the 3 to 20 MHz oscillators. Voltage ranges from 1.62V to 1.95V.

## 5.2 Power-up Considerations

### 5.2.1 VDDIO Versus VDDCORE

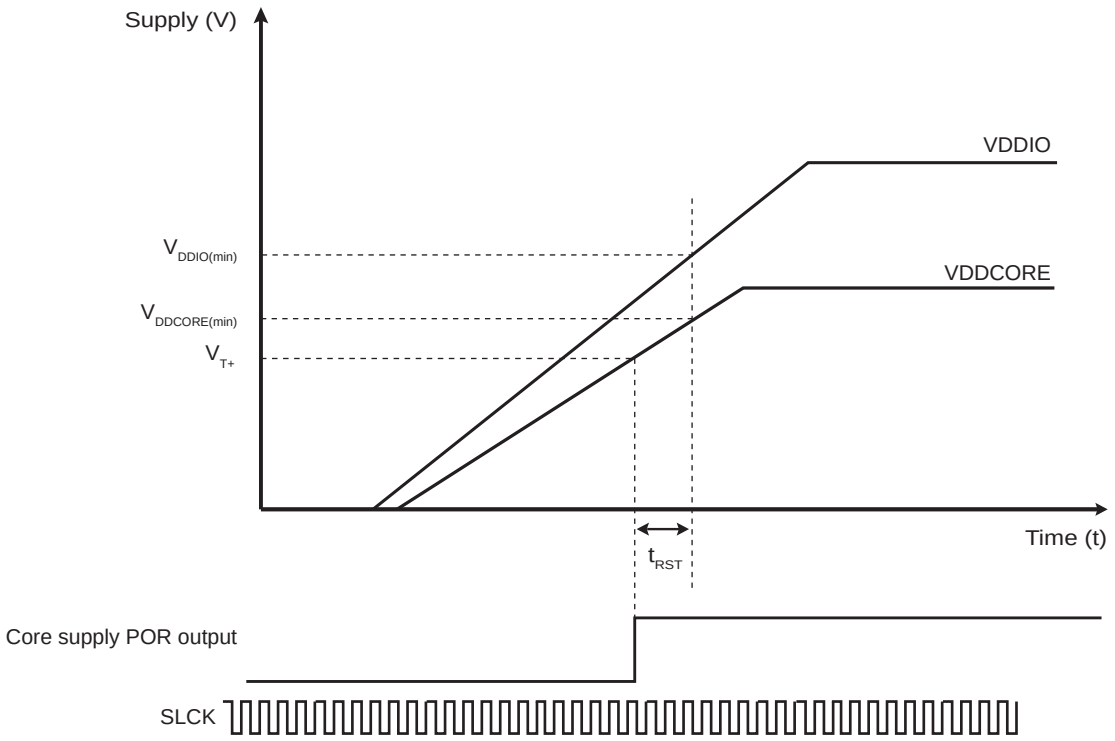
$V_{DDIO}$ must always be higher or equal to $V_{DDCORE}$.

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

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

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

**Figure 5-1. VDDCORE and VDDIO Constraints at Startup**



### 5.2.2 VDDIO Versus VDDIN

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

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

## 5.3 Voltage Regulator

The SAM3N embeds a voltage regulator that is managed by the Supply Controller.

This internal regulator is intended to supply the internal core of SAM3N. It features two different operating modes:

- In Normal mode, the voltage regulator consumes less than 700 µA static current and draws 60 mA of output current. Internal adaptive biasing adjusts the regulator quiescent current depending on the required load current. In Wait mode quiescent current is only 7 µA.
- In Backup mode, the voltage regulator consumes less than 1 µA while its output (VDDOUT) is driven internally to GND. The default output voltage is 1.80 V and the start-up time to reach Normal mode is less than 100 µs.

For adequate input and output power supply decoupling/bypassing, refer to Table 36-3 ”1.8V Voltage Regulator Characteristics”.

## 5.4 Typical Powering Schematics

The SAM3N supports a 1.62–3.6 V single supply mode. The internal regulator input connected to the source and its output feeds VDDCORE. Figure 5-2 shows the power schematics.

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

**Figure 5-2. Single Supply**

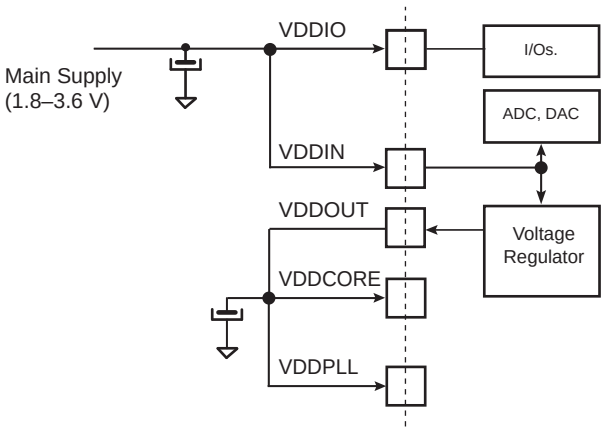
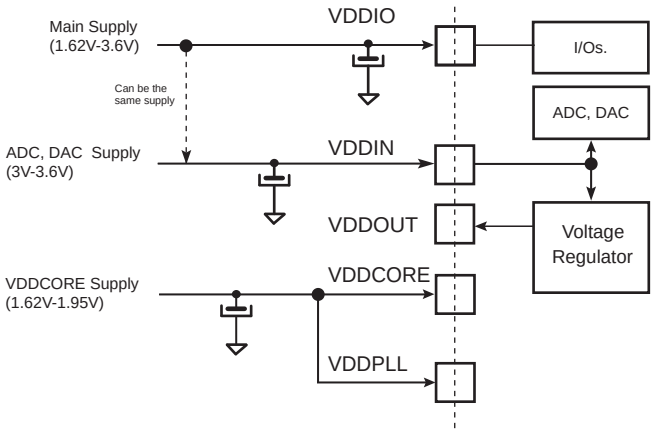


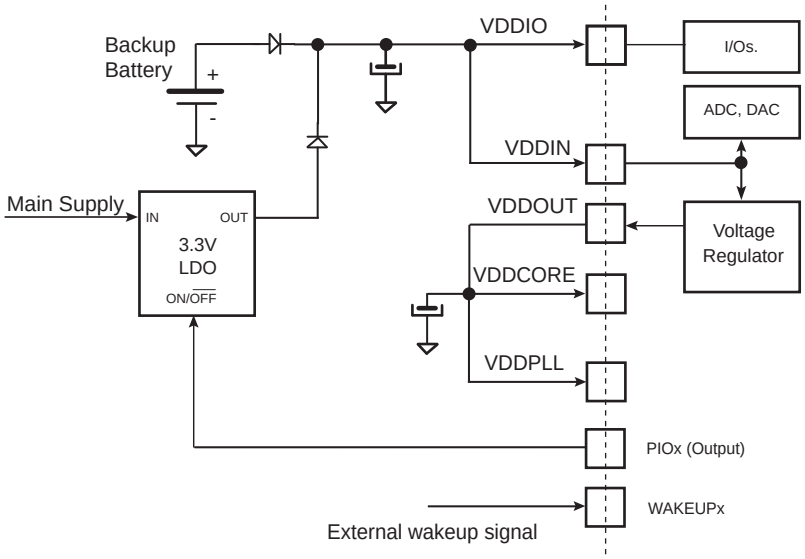
**Figure 5-3. Core Externally Supplied**




Note: Restrictions:
- With Main Supply < 3V, ADC and DAC are not usable.
- With Main Supply ≥ 3V, all peripherals are usable.

Figure 5-4 provides an example of the powering scheme when using a backup battery. Since the PIO state is preserved when in backup mode, any free PIO line can be used to switch off the external regulator by driving the PIO line at low level (PIO is input, pull-up enabled after backup reset). External wake-up of the system can be from a push button or any signal. See Section 5.7 “Wake-up Sources” for further details.

**Figure 5-4. Core Externally Supplied (Backup Battery)**




Notes: 1. The two diodes provide a “switchover circuit” (for illustration purpose) between the backup battery and the main supply when the system is put in backup mode.
2. Restrictions:
- With Main Supply < 3V, ADC and DAC are not usable.
- With Main Supply ≥ 3V, all peripherals are usable.

## 5.5 Active Mode

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

## 5.6 Low Power Modes

The various low-power modes of the SAM3N are described below.

### 5.6.1 Backup Mode

The purpose of backup mode is to achieve the lowest power consumption possible in a system that is performing periodic wakeups to carry out tasks but not requiring fast startup time (< 0.1ms). Total current consumption is 3 µA typical.

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

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

The SAM3N can be woken up from this mode through pins WKUP0–15, the supply monitor (SM), the RTT or RTC wake-up event.

Backup mode can be entered by using the WFE instruction.

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

1. Write a 1 to the SLEEPDEEP bit in the Cortex-M3 processor System Control Register (SCR) (refer to Section 11.21.7 “System Control Register”).
2. Execute the WFE instruction of the processor.

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

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

### 5.6.2 Wait Mode

The purpose of the wait mode is to achieve very low power consumption while maintaining the whole device in a powered state for a startup time of less than 10 µs. Current consumption in Wait mode is typically 15 µA (total current consumption) if the internal voltage regulator is used or 8 µA if an external regulator is used.

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

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

The procedure to enter Wait Mode is the following:

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

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

### 5.6.3 Sleep Mode

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

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

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

### 5.6.4 Low Power Mode Summary Table

The modes detailed above are the main low power modes. Each part can be set to on or off separately and wake up sources can be individually configured. Table 5-1 on page 21 shows a summary of the configurations of the low power modes.

**Table 5-1. Low Power Mode Configuration Summary**

| Mode | SUPC, 32 kHz Osc., RTC, RTT, GPBR, POR (Backup Region) | Regulator | Core Memory Peripherals | Mode Entry | Potential Wake-up Sources | Core at Wake-up | PIO State While in Low Power Mode | PIO State at Wake-up | Consumption (2) (3) | Wake-up Time(1) |
|---|---|---|---|---|---|---|---|---|---|---|
| Backup | ON | OFF | OFF<br>(Not powered) | WFE<br>+ SLEEPDEEP = 1 | WKUP0–15 pins<br>RTC alarm<br>RTT alarm<br>SM alarm | Reset | Previous state saved | PIOA & PIOB & PIOC inputs with pull ups | 3 µA typ(4) | < 0.1 ms |
| Wait | ON | ON | Powered<br>(Not clocked) | WFE<br>+ SLEEPDEEP = 0<br>+ LPM bit = 1 | Any event from<br>- Fast startup through pins WKUP0–15<br>- RTC alarm<br>- RTT alarm<br>- SM alarm | Clocked back | Previous state saved | Unchanged | 5 µA/15 µA (5) | < 10 µs |
| Sleep | ON | ON | Powered(7)<br>(Not clocked) | WFE or WFI<br>+ SLEEPDEEP = 0<br>+ LPM bit = 0 | Entry mode = WFI Interrupt Only;<br>Entry mode = WFE<br>Any enabled interrupt<br>and/or<br>any event from<br>- Fast startup through pins WKUP0–15<br>- RTC alarm<br>- RTT alarm<br>- SM alarm | Clocked back | Previous state saved | Unchanged | (6) | (6) |

Notes:
1. When considering wake-up time, the time required to start the PLL is not taken into account. Once started, the device works with the 4/8/12 MHz Fast RC oscillator. The user has to add the PLL start-up time if it is needed in the system. The wake-up time is defined as the time taken for wake up until the first instruction is fetched.
2. The external loads on PIOs are not taken into account in the calculation.
3. Supply Monitor current consumption is not included.
4. Total current consumption.
5. 5 µA on VDDCORE, 15 µA for total current consumption (using internal voltage regulator), 8 µA for total current consumption (without using internal voltage regulator).
6. Depends on MCK frequency.
7. In this mode the core is supplied and not clocked but some peripherals can be clocked.

## 5.7 Wake-up Sources

The wake-up events allow the device to exit Backup mode. When a wake-up event is detected, the Supply Controller performs a sequence which automatically reenables the core power supply and the SRAM power supply, if they are not already enabled. See Figure 17-4, "Wake Up Sources" on page 255.

## 5.8 Fast Startup

The SAM3N 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 19 wake-up inputs (WKUP0 to 15 + SM + RTC + RTT).

The fast restart circuitry (shown in Figure 25-3, "Fast Startup Circuitry" on page 337), is fully asynchronous and provides a fast start-up signal to the Power Management Controller. As soon as the fast start-up signal is asserted, the PMC automatically restarts the embedded 4 MHz fast RC oscillator, switches the master clock on this 4 MHz clock and reenables the processor clock.

# 6. Input/Output Lines

The SAM3N has several kinds of input/output (I/O) lines such as general purpose I/Os (GPIO) and system I/Os. GPIOs can have alternate functionality due to multiplexing capabilities of the PIO controllers. The same PIO line can be used whether in IO mode or by the multiplexed peripheral. System I/Os include pins such as test pins, oscillators, erase or analog inputs.

## 6.1 General Purpose I/O Lines

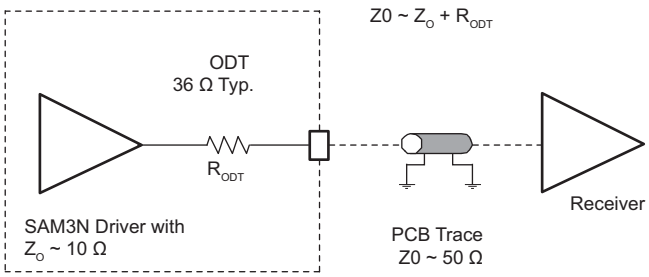
GPIO Lines are managed by PIO Controllers. All I/Os have several input or output modes such as pull-up or pull-down, input Schmitt triggers, multi-drive (open-drain), glitch filters, debouncing or input change interrupt. Programming of these modes is performed independently for each I/O line through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

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

The SAM3N embeds high speed pads able to handle up to 45 MHz for SPI clock lines and 35 MHz on other lines. See Section 36.8 “AC Characteristics” for more details. Typical pull-up and pull-down value is 100 kΩ for all I/Os.

Each I/O line also embeds an ODT (On-Die Termination) (see Figure 6-1). ODT consists of an internal series resistor termination scheme for impedance matching between the driver output (SAM3N) and the PCB trace impedance preventing signal reflection. The series resistor helps to reduce I/O switching current (di/dt) thereby reducing in turn, EMI. It also decreases overshoot and undershoot (ringing) due to inductance of interconnect between devices or between boards. In conclusion ODT helps diminish signal integrity issues.

**Figure 6-1. On-Die Termination**



## 6.2 System I/O Lines

Table 6-1 lists the SAM3N system I/O lines shared with PIO lines. These pins are software configurable as general purpose I/O or system pins. At startup, the default function of these pins is always used.

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

| CCFG_SYSIO Bit No. | Default Function After Reset | Other Function | Constraints For Normal Start | Configuration |
|---|---|---|---|---|
| 12 | ERASE | PB12 | Low Level at startup(1) | In Matrix User Interface Registers (refer to System I/O Configuration Register in Section 22. "Bus Matrix (MATRIX)") |
| 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 17.4.2 "Slow Clock Generator".
3. Refer to Section 24.5.3 "3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator".

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

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

At startup, SWJ-DP pins are configured in SWJ-DP mode to allow connection with debugging probe. Please refer to Section 12. "Debug and Test Features".

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

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

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

The asynchronous TRACE output (TRACESWO) is multiplexed with TDO. So the asynchronous trace can only be used with SW-DP, not JTAG-DP. For more information about SW-DP and JTAG-DP switching, please refer to Section 12. "Debug and Test Features".

## 6.3 Test Pin

The TST pin is used for JTAG Boundary Scan Manufacturing Test or Fast Flash programming mode of the SAM3N series. The TST pin integrates a permanent pull-down resistor of about 15 kΩ to GND, so that it can be left unconnected for normal operations. To enter fast programming mode, see Section 20. "Fast Flash Programming Interface (FFPI)". For more on the manufacturing and test mode, refer to Section 12. "Debug and Test Features".

## 6.4 NRST Pin

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

## 6.5 ERASE Pin

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

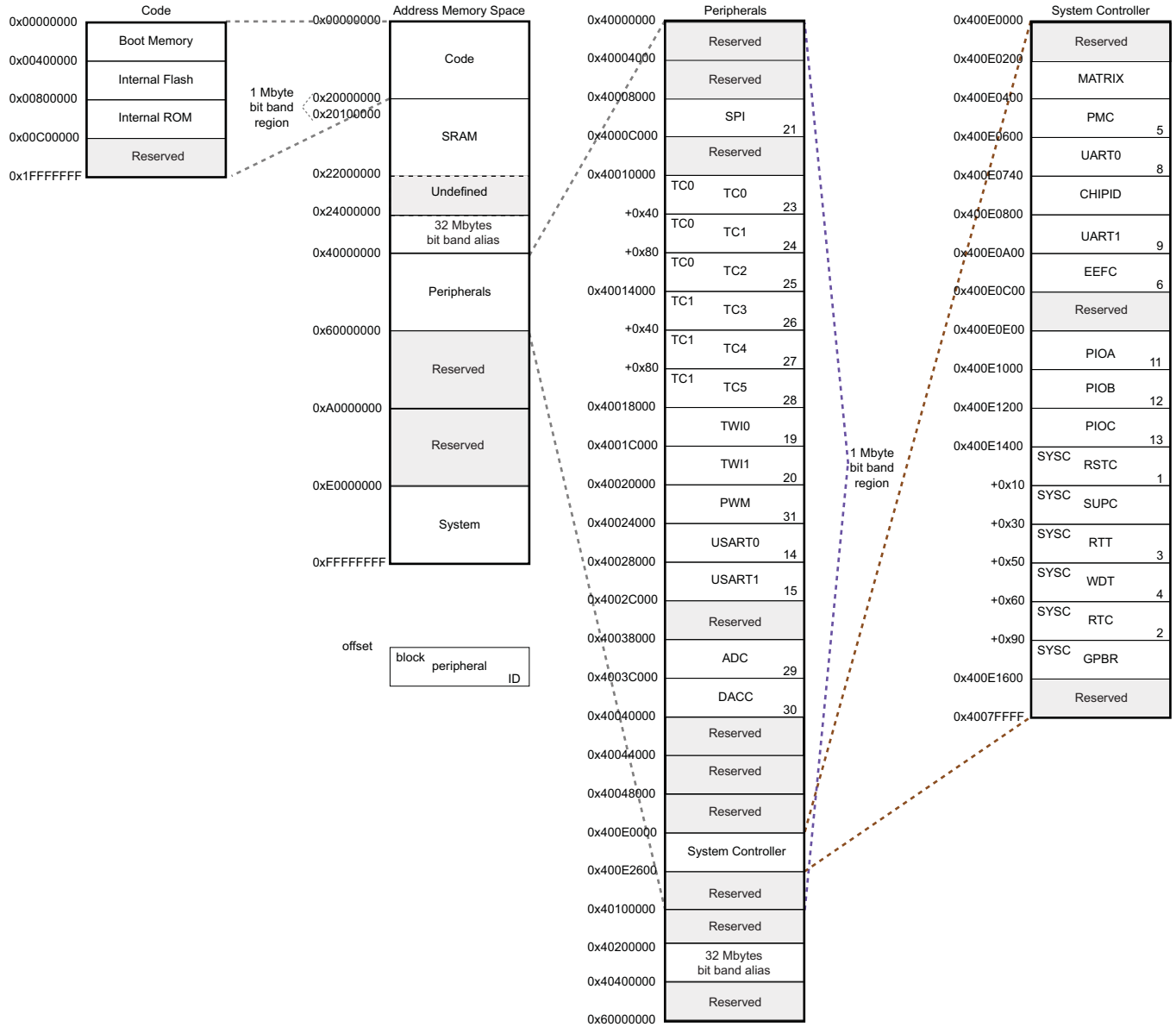
This pin is debounced by SCLK to improve the glitch tolerance. When the ERASE pin is tied high during less than 100 ms, it is not taken into account. The pin must be tied high during more than 220 ms to perform a Flash erase operation.

The ERASE pin is a system I/O pin and can be used as a standard I/O. At startup, the ERASE pin is not configured as a PIO pin. If the ERASE pin is used as a standard I/O, startup level of this pin must be low to prevent unwanted erasing. Please refer to Section 10.3 “Peripheral Signal Multiplexing on I/O Lines” on page 32. Also, if the ERASE pin is used as a standard I/O output, asserting the pin to high does not erase the Flash.

# 7. Memories

## 7.1 Product Mapping

**Figure 7-1. SAM3N4/2/1/0/00 Product Mapping**



## 7.2 Embedded Memories

### 7.2.1 Internal SRAM

Table 7-2 shows the SRAM size for the various devices.

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

| Device | SRAM Size (Kbytes) |
|---|---|
| SAM3N4 | 24 |
| SAM3N2 | 16 |
| SAM3N1 | 8 |
| SAM3N0 | 8 |
| SAM3N00 | 4 |

The SRAM is accessible over System Cortex-M3 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.

RAM size must be configurable by calibration fuses.

### 7.2.2 Internal ROM

The SAM3N product embeds an Internal ROM, which contains the SAM Boot Assistant (SAM-BA), In Application Programming (IAP) routines and Fast Flash Programming Interface (FFPI).

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

### 7.2.3 Embedded Flash

#### 7.2.3.1 Flash Overview

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

**Table 7-2. Embedded Flash Memory Organization per Device**

| Device | Flash Size (Kbytes) | Number of Banks | Number of Pages | Page Size (bytes) | Plane |
|---|---|---|---|---|---|
| SAM3N4 | 256 | 1 | 1024 | 256 | Single |
| SAM3N2 | 128 | 1 | 512 | 256 | Single |
| SAM3N1 | 64 | 1 | 256 | 256 | Single |
| SAM3N0 | 32 | 1 | 128 | 256 | Single |
| SAM3N00 | 16 | 1 | 64 | 256 | Single |

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

#### 7.2.3.2 Flash Power Supply

The Flash is supplied by VDDCORE.

#### 7.2.3.3 Enhanced Embedded Flash Controller

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

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

The user can choose between high performance or lower current consumption by selecting either 128-bit or 64-bit access. It also manages the programming, erasing, locking and unlocking sequences of the Flash using a full set of commands.

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

#### 7.2.3.4 Flash Speed

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

For more details, refer to Section 36.8 “AC Characteristics”.

#### 7.2.3.5 Lock Regions

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

**Table 7-3. Lock bit number**

| Product | Number of Lock Bits | Lock Region Size |
|---|---|---|
| SAM3N4 | 16 | 16 Kbytes (64 pages) |
| SAM3N2 | 8 | 16 Kbytes (64 pages) |
| SAM3N1 | 4 | 16 Kbytes (64 pages) |
| SAM3N0 | 2 | 16 Kbytes (64 pages) |
| SAM3N00 | 1 | 16 Kbytes (64 pages) |

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

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

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

#### 7.2.3.6 Security Bit Feature

The SAM3N features a security bit, based on a specific General Purpose NVM bit (GPNVM bit 0). When the security is enabled, any access to the Flash, either through the ICE interface or through the Fast Flash Programming Interface (FFPI), is forbidden. This ensures the confidentiality of the code programmed in the Flash.

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

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

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

#### 7.2.3.7 Calibration Bits

NVM bits are used to calibrate the brownout detector and the voltage regulator. These bits are factory configured and cannot be changed by the user. The ERASE pin has no effect on the calibration bits.

#### 7.2.3.8 Unique Identifier

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

#### 7.2.3.9 Fast Flash Programming Interface (FFPI)

The FFPI allows programming the device through either a serial JTAG interface or through a multiplexed fully-handshaked parallel port. It allows gang programming with market-standard industrial programmers.

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

The FFPI is enabled and the Fast Programming Mode is entered when TST and PA0 and PA1 are tied low.

#### 7.2.3.10 SAM-BA Boot

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

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

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

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

#### 7.2.3.11 GPNVM Bits

The SAM3N 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 7-4. General-purpose Non volatile Memory Bits**

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

### 7.2.4 Boot Strategies

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

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

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

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

# 8. Real-time Event Management

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

## 8.1 Embedded Characteristics

- IO peripherals generate event triggers which are directly routed to event managers such as ADC or DACC, for example, to start measurement/conversion without processor intervention.
- UART, USART, SPI, TWI, ADC, DACC, PIO also generate event triggers directly connected to Peripheral DMA Controller (PDC) for data transfer without processor intervention.
- Parallel capture logic is directly embedded in PIO and generates trigger event to PDC to capture data without processor intervention.

## 8.2 Real-time Event Mapping

**Table 8-1. Real-time Event Mapping List**

| Function | Application | Description | Event Source | Event Destination |
|---|---|---|---|---|
| Measurement trigger | General-purpose | Trigger source selection in ADC (1) | PIO (ADTRG) | Analog-to-Digital Converter (ADC) |
| | | | TC: TIOA0 | |
| | | | TC: TIOA1 | |
| | | | TC: TIOA2 | |
| Conversion trigger | General-purpose | Trigger source selection in DACC (2) | PIO DATRG | Digital-to-Analog Converter Controller (DACC) |
| | | | TC Output 0 | |
| | | | TC Output 1 | |
| | | | TC Output 2 | |
| Direct Memory Access | General-purpose | Peripheral trigger event generation to transfer data to/from system memory (3) | USART/UART, TWI, ADC | Peripheral DMA Controller (PDC) |

Notes:
1. Refer to “Conversion Triggers” and the “ADC Mode Register” (ADC_MR) in Section 34. “Analog-to-digital Converter (ADC)”.
2. Refer to “DACC Mode Register” (DACC_MR) in Section 35. “Digital to Analog Converter Controller (DACC)”.
3. Refer to Section 23. “Peripheral DMA Controller (PDC)”

# 9. System Controller

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

## 9.1 System Controller and Peripheral Mapping

Please refer to Figure 7-1, "SAM3N4/2/1/0/00 Product Mapping" on page 25.

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

## 9.2 Power-on-Reset, Brownout and Supply Monitor

The SAM3N embeds three features to monitor, warn and/or reset the chip:

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

### 9.2.1 Power-on-Reset

The Power-on-Reset monitors VDDIO. It is always activated and monitors voltage at start up but also during power down. If VDDIO goes below the threshold voltage, the entire chip is reset. For more information, refer to Section 36. “Electrical Characteristics”.

### 9.2.2 Brownout Detector on VDDCORE

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

If VDDCORE goes below the threshold voltage, the reset of the core is asserted. For more information, refer to Section 17. “Supply Controller (SUPC)” and Section 36. “Electrical Characteristics”.

### 9.2.3 Supply Monitor on VDDIO

The Supply Monitor monitors VDDIO. It is inactive by default. It can be activated by software and is fully programmable with 16 steps for the threshold (between 1.9V to 3.4V). It is controlled by the Supply Controller (SUPC). A sample mode is possible. It allows to divide the supply monitor power consumption by a factor of up to 2048. For more information, refer to Section 17. “Supply Controller (SUPC)” and Section 36. “Electrical Characteristics”.

# 10. Peripherals

## 10.1 Peripheral Identifiers

Table 10-1 defines the Peripheral Identifiers of the SAM3N4/2/1/0/00. A peripheral identifier is required for the control of the peripheral interrupt with the Nested Vectored Interrupt Controller and for the control of the peripheral clock with the Power Management Controller.

**Table 10-1. Peripheral Identifiers**

| Instance ID | Instance Name | NVIC Interrupt | PMC Clock Control | Instance Description |
|---|---|---|---|---|
| 0 | SUPC | X | | Supply Controller |
| 1 | RSTC | X | | Reset Controller |
| 2 | RTC | X | | Real-time Clock |
| 3 | RTT | X | | Real-time Timer |
| 4 | WDT | X | | Watchdog Timer |
| 5 | PMC | X | | Power Management Controller |
| 6 | EEFC | X | | Enhanced Embedded Flash Controller |
| 7 | – | – | | Reserved |
| 8 | UART0 | X | X | Universal Asynchronous Receiver Transceiver 0 |
| 9 | UART1 | X | X | Universal Asynchronous Receiver Transceiver 1 |
| 10 | – | – | – | Reserved |
| 11 | PIOA | X | X | Parallel I/O Controller A |
| 12 | PIOB | X | X | Parallel I/O Controller B |
| 13 | PIOC | X | X | Parallel I/O Controller C |
| 14 | USART0 | X | X | Universal Synchronous Asynchronous Receiver Transmitter 0 |
| 15 | USART1 | X | X | Universal Synchronous Asynchronous Receiver Transmitter 1 |
| 16 | – | – | – | Reserved |
| 17 | – | – | – | Reserved |
| 18 | – | – | – | Reserved |
| 19 | TWI0 | X | X | Two-wire Interface 0 |
| 20 | TWI1 | X | X | Two-wire Interface 1 |
| 21 | SPI | X | X | Serial Peripheral Interface |
| 22 | – | – | – | Reserved |
| 23 | TC0 | X | X | Timer Counter Channel 0 |
| 24 | TC1 | X | X | Timer Counter Channel 1 |
| 25 | TC2 | X | X | Timer Counter Channel 2 |
| 26 | TC3 | X | X | Timer Counter Channel 3 |
| 27 | TC4 | X | X | Timer Counter Channel 5 |
| 28 | TC5 | X | X | Timer Counter Channel 5 |
| 29 | ADC | X | X | Analog-to-Digital Converter |
| 30 | DACC | X | X | Digital-to-Analog Converter Controller |
| 31 | PWM | X | X | Pulse Width Modulation |

## 10.2 APB/AHB Bridge

The SAM3N4/2/1/0/00 product embeds one peripheral bridge.

The peripherals of the bridge are clocked by MCK.

## 10.3 Peripheral Signal Multiplexing on I/O Lines

The SAM3N product features up to three PIO controllers (PIOA, PIOB, and PIOC) that multiplex the I/O lines of the peripheral set:

- 2 PIO controllers on 48-pin and 64-pin version devices
- 3 PIO controllers on 100-pin version devices

The SAM3N 64-pin and 100-pin PIO Controller controls up to 32 lines (see Table 10-2, “Multiplexing on PIO Controller A (PIOA),” on page 33). Each line can be assigned to one of three peripheral functions: A, B or C. The multiplexing tables in the following paragraphs define how the I/O lines of the peripherals A, B and C are multiplexed on the PIO Controllers.

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

### 10.3.1 PIO Controller A Multiplexing

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

| I/O Line | Peripheral A | Peripheral B | Peripheral C | Extra Function | System Function | Comments |
|---|---|---|---|---|---|---|
| PA0 | PWM0 | TIOA0 | | WKUP0 (1) | | High drive |
| PA1 | PWM1 | TIOB0 | | WKUP1(1) | | High drive |
| PA2 | PWM2 | SCK0 | DATRG | WKUP2(1) | | High drive |
| PA3 | TWD0 | NPCS3 | | | | High drive |
| PA4 | TWCK0 | TCLK0 | | WKUP3(1) | | |
| PA5 | RXD0 | NPCS3 | | WKUP4(1) | | |
| PA6 | TXD0 | PCK0 | | | | |
| PA7 | RTS0 | PWM3 | | | XIN32 (2) | |
| PA8 | CTS0 | ADTRG | | WKUP5(1) | XOUT32(2) | |
| PA9 | URXD0 | NPCS1 | | WKUP6(1) | | |
| PA10 | UTXD0 | NPCS2 | | | | |
| PA11 | NPCS0 | PWM0 | | WKUP7(1) | | |
| PA12 | MISO | PWM1 | | | | |
| PA13 | MOSI | PWM2 | | | | |
| PA14 | SPCK | PWM3 | | WKUP8(1) | | |
| PA15 | | TIOA1 | | WKUP14(1) | | |
| PA16 | | TIOB1 | | WKUP15(1) | | |
| PA17 | | PCK1 | | AD0 (3) | | |
| PA18 | | PCK2 | | AD1(3) | | |
| PA19 | | | | AD2/WKUP9 (4) | | |
| PA20 | | | | AD3/WKUP10(4) | | |
| PA21 | RXD1 | PCK1 | | AD8(3) | | 64/100-pin versions |
| PA22 | TXD1 | NPCS3 | | AD9(3) | | 64/100-pin versions |
| PA23 | SCK1 | PWM0 | | | | 64/100-pin versions |
| PA24 | RTS1 | PWM1 | | | | 64/100-pin versions |
| PA25 | CTS1 | PWM2 | | | | 64/100-pin versions |
| PA26 | | TIOA2 | | | | 64/100-pin versions |
| PA27 | | TIOB2 | | | | 64/100-pin versions |
| PA28 | | TCLK1 | | | | 64/100-pin versions |
| PA29 | | TCLK2 | | | | 64/100-pin versions |
| PA30 | | NPCS2 | | WKUP11(1) | | 64/100-pin versions |
| PA31 | NPCS1 | PCK2 | | | | 64/100-pin versions |

1. WKUPx can be used if PIO controller defines the I/O line as “input”.
2. Refer to Section 6.2 “System I/O Lines”.
3. To select this extra function, refer to Section 34.5.3 “Analog Inputs”.
4. Analog input has priority over WKUPx pin.

### 10.3.2 PIO Controller B Multiplexing

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

| I/O Line | Peripheral A | Peripheral B | Peripheral C | Extra Function | System Function | Comments |
|---|---|---|---|---|---|---|
| PB0 | PWM0 | | | AD4 (1) | | |
| PB1 | PWM1 | | | AD5(1) | | |
| PB2 | URXD1 | NPCS2 | | AD6/WKUP12 (2) | | |
| PB3 | UTXD1 | PCK2 | | AD7(1) | | |
| PB4 | TWD1 | PWM2 | | | TDI (3) | |
| PB5 | TWCK1 | | | WKUP13 (4) | TDO/TRACESWO(3) | |
| PB6 | | | | | TMS/SWDIO(3) | |
| PB7 | | | | | TCK/SWCLK(3) | |
| PB8 | | | | | XOUT(3) | |
| PB9 | | | | | XIN(3) | |
| PB10 | | | | | | |
| PB11 | | | | | | |
| PB12 | | | | | ERASE(3) | |
| PB13 | | PCK0 | | DAC0 (5) | | 64/100-pin versions |
| PB14 | NPCS1 | PWM3 | | | | 64/100-pin versions |

1. To select this extra function, refer to Section 34.5.3 “Analog Inputs”.
2. Analog input has priority over WKUPx pin.
3. Refer to Section 6.2 “System I/O Lines”.
4. WKUPx can be used if PIO controller defines the I/O line as “input”.
5. DAC0 is enabled when DACC_MR.DACEN is set. See Section 35.7.2 “DACC Mode Register”.

### 10.3.3 PIO Controller C Multiplexing

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

| I/O Line | Peripheral A | Peripheral B | Peripheral C | Extra Function | System Function | Comments |
|---|---|---|---|---|---|---|
| PC0 | | | | | | 100-pin version |
| PC1 | | | | | | 100-pin version |
| PC2 | | | | | | 100-pin version |
| PC3 | | | | | | 100-pin version |
| PC4 | | NPCS1 | | | | 100-pin version |
| PC5 | | | | | | 100-pin version |
| PC6 | | | | | | 100-pin version |
| PC7 | | NPCS2 | | | | 100-pin version |
| PC8 | | PWM0 | | | | 100-pin version |
| PC9 | | PWM1 | | | | 100-pin version |
| PC10 | | PWM2 | | | | 100-pin version |
| PC11 | | PWM3 | | | | 100-pin version |
| PC12 | | | | AD12 (1) | | 100-pin version |
| PC13 | | | | AD10(1) | | 100-pin version |
| PC14 | | PCK2 | | | | 100-pin version |
| PC15 | | | | AD11(1) | | 100-pin version |
| PC16 | | PCK0 | | | | 100-pin version |
| PC17 | | PCK1 | | | | 100-pin version |
| PC18 | | PWM0 | | | | 100-pin version |
| PC19 | | PWM1 | | | | 100-pin version |
| PC20 | | PWM2 | | | | 100-pin version |
| PC21 | | PWM3 | | | | 100-pin version |
| PC22 | | PWM0 | | | | 100-pin version |
| PC23 | | TIOA3 | | | | 100-pin version |
| PC24 | | TIOB3 | | | | 100-pin version |
| PC25 | | TCLK3 | | | | 100-pin version |
| PC26 | | TIOA4 | | | | 100-pin version |
| PC27 | | TIOB4 | | | | 100-pin version |
| PC28 | | TCLK4 | | | | 100-pin version |
| PC29 | | TIOA5 | | AD13(1) | | 100-pin version |
| PC30 | | TIOB5 | | AD14(1) | | 100-pin version |
| PC31 | | TCLK5 | | AD15(1) | | 100-pin version |

1. To select this extra function, refer to Section 34.5.3 “Analog Inputs”.

# 11. ARM Cortex-M3 Processor

## 11.1 About this section

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

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

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

## 11.2 Embedded Characteristics

- Version 2.0
- Thumb-2 (ISA) subset consisting of all base Thumb-2 instructions, 16-bit and 32-bit.
- Harvard processor architecture enabling simultaneous instruction fetch with data load/store.
- Three-stage pipeline.
- Single cycle 32-bit multiply.
- Hardware divide.
- Thumb and Debug states.
- Handler and Thread modes.
- Low latency ISR entry and exit.
- SysTick Timer
  - 24-bit down counter
  - Self-reload capability
  - Flexible System timer
- Nested Vectored Interrupt Controller
  - Thirty-two maskable external interrupts
  - Sixteen priority levels
  - Processor state automatically saved on interrupt entry, and restored on
  - Dynamic reprioritization of interrupts
  - Priority grouping

    selection of pre-empting interrupt levels and non pre-empting interrupt levels
  - Support for tail-chaining and late arrival of interrupts

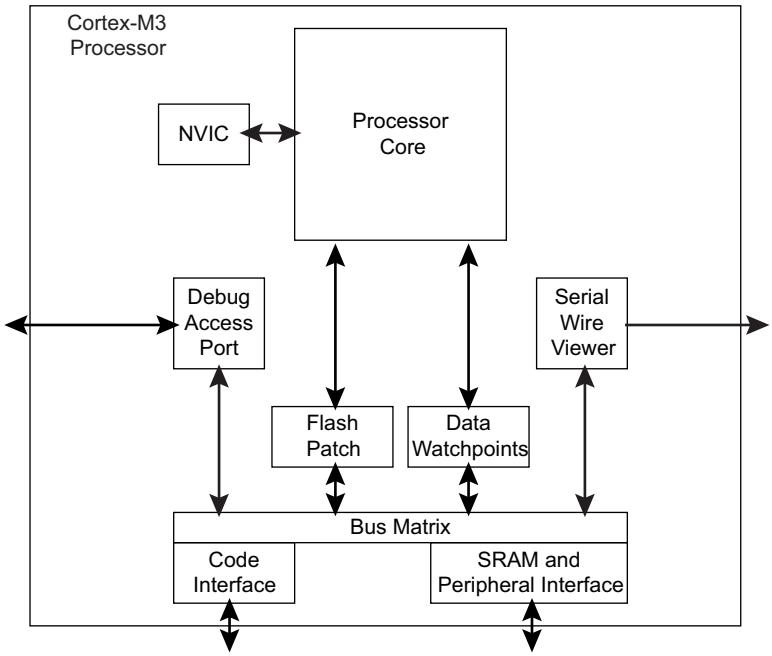
    back-to-back interrupt processing without the overhead of state saving and restoration between interrupts.

Processor state automatically saved on interrupt entry and restored on interrupt exit, with no instruction overhead

## 11.3 About the Cortex-M3 processor and core peripherals

- The Cortex-M3 processor is a high performance 32-bit processor designed for the microcontroller market. It offers significant benefits to developers, including:
- outstanding processing performance combined with fast interrupt handling
- enhanced system debug with extensive breakpoint and trace capabilities
- efficient processor core, system and memories
- ultra-low power consumption with integrated sleep modes

**Figure 11-1. Typical Cortex-M3 Implementation**




The Cortex-M3 processor is built on a high-performance processor core, with a 3-stage pipeline Harvard architecture, making it ideal for demanding embedded applications. The processor delivers exceptional power efficiency through an efficient instruction set and extensively optimized design, providing high-end processing hardware including single-cycle 32x32 multiplication and dedicated hardware division.

To facilitate the design of cost-sensitive devices, the Cortex-M3 processor implements tightly-coupled system components that reduce processor area while significantly improving interrupt handling and system debug capabilities. The Cortex-M3 processor implements a version of the Thumb® instruction set, ensuring high code density and reduced program memory requirements. The Cortex-M3 instruction set provides the exceptional performance expected of a modern 32-bit architecture, with the high code density of 8-bit and 16-bit microcontrollers.

The Cortex-M3 processor closely integrates a configurable *nested interrupt controlle*r (NVIC), to deliver industry-leading interrupt performance. The NVIC provides up to 16 interrupt priority levels. The tight integration of the processor core and NVIC provides fast execution of *interrupt service routines* (ISRs), dramatically reducing the interrupt latency. This is achieved through the hardware stacking of registers, and the ability to suspend load-multiple and store-multiple operations. Interrupt handlers do not require any assembler stubs, removing any code overhead from the ISRs. Tail-chaining optimization also significantly reduces the overhead when switching from one ISR to another.

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

### 11.3.1 System level interface

The Cortex-M3 processor provides multiple interfaces using AMBA® technology to provide high speed, low latency memory accesses. It supports unaligned data accesses and implements atomic bit manipulation that enables faster peripheral controls, system spinlocks and thread-safe Boolean data handling.

### 11.3.2 Integrated configurable debug

The Cortex-M3 processor implements a complete hardware debug solution. This provides high system visibility of the processor and memory through either a traditional JTAG port or a 2-pin *Serial Wire Debug* (SWD) port that is ideal for microcontrollers and other small package devices.

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

### 11.3.3 Cortex-M3 processor features and benefits summary

- tight integration of system peripherals reduces area and development costs
- Thumb instruction set combines high code density with 32-bit performance
- code-patch ability for ROM system updates
- power control optimization of system components
- integrated sleep modes for low power consumption
- fast code execution permits slower processor clock or increases sleep mode time
- hardware division and fast multiplier
- deterministic, high-performance interrupt handling for time-critical applications
- extensive debug and trace capabilities:
  - Serial Wire Debug and Serial Wire Trace reduce the number of pins required for debugging and tracing.

### 11.3.4 Cortex-M3 core peripherals

These are:

#### 11.3.4.1 Nested Vectored Interrupt Controller

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

#### 11.3.4.2 System control block

The *System control block* (SCB) is the programmers model interface to the processor. It provides system implementation information and system control, including configuration, control, and reporting of system exceptions.

#### 11.3.4.3 System timer

The system timer, SysTick, is a 24-bit count-down timer. Use this as a Real Time Operating System (RTOS) tick timer or as a simple counter.

## 11.4 Programmers model

This section describes the Cortex-M3 programmers model. In addition to the individual core register descriptions, it contains information about the processor modes and privilege levels for software execution and stacks.

### 11.4.1 Processor mode and privilege levels for software execution

The processor *modes* are:

#### 11.4.1.1 Thread mode

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

#### 11.4.1.2 Handler mode

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

The *privilege levels* for software execution are:

#### 11.4.1.3 Unprivileged

The software:

- has limited access to the MSR and MRS instructions, and cannot use the CPS instruction
- cannot access the system timer, NVIC, or system control block
- might have restricted access to memory or peripherals.

*Unprivileged software* executes at the unprivileged level.

#### 11.4.1.4 Privileged

The software can use all the instructions and has access to all resources.

*Privileged software* executes at the privileged level.

In Thread mode, the CONTROL register controls whether software execution is privileged or unprivileged, see “CONTROL register” on page 51. In Handler mode, software execution is always privileged.

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

### 11.4.2 Stacks

The processor uses a full descending stack. This means the stack pointer indicates the last stacked item on the stack memory. When the processor pushes a new item onto the stack, it decrements the stack pointer and then writes the item to the new memory location. The processor implements two stacks, the *main stack* and the *process stack*, with independent copies of the stack pointer, see “Stack Pointer” on page 41.

In Thread mode, the CONTROL register controls whether the processor uses the main stack or the process stack, see “CONTROL register” on page 51. In Handler mode, the processor always uses the main stack. The options for processor operations are:

**Table 11-1. Summary of processor mode, execution privilege level, and stack use options**

| Processor mode | Used to execute | Privilege level for software execution | Stack used |
|---|---|---|---|
| Thread | Applications | Privileged or unprivileged (1) | Main stack or process stack(1) |
| Handler | Exception handlers | Always privileged | Main stack |

1. See “CONTROL register” on page 51.

### 11.4.3 Core registers

The processor core registers are:

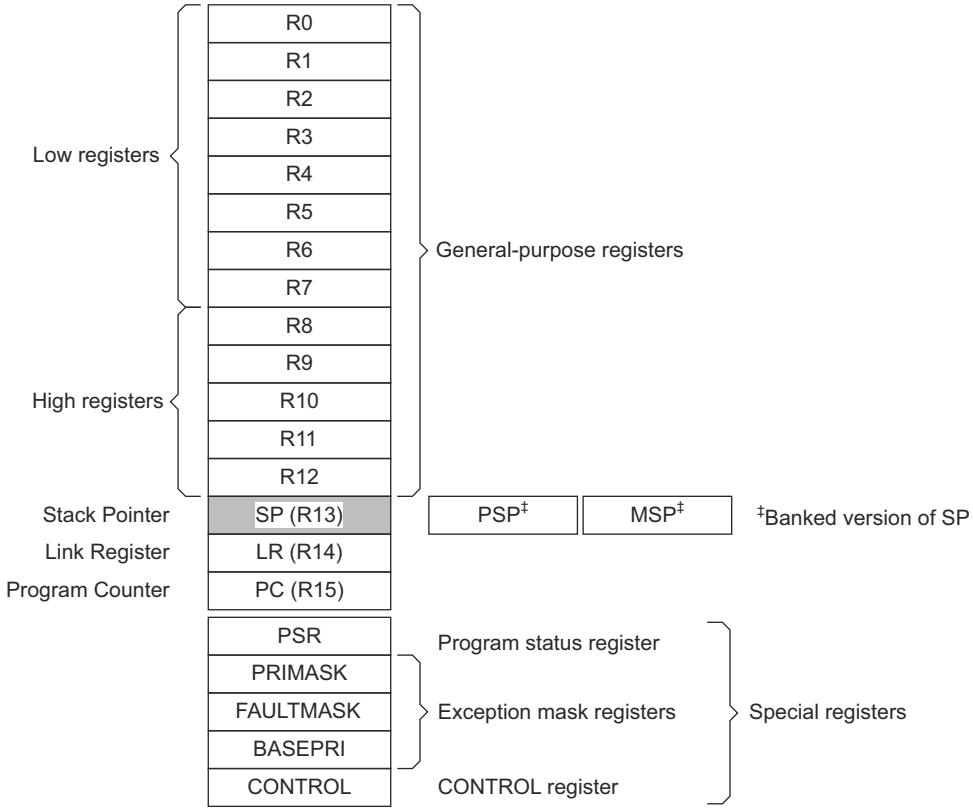



**Table 11-2. Core register set summary**

| Name | Type [1] | Required privilege [2] | Reset value | Description |
|---|---|---|---|---|
| R0-R12 | RW | Either | Unknown | “General-purpose registers” on page 41 |
| MSP | RW | Privileged | See description | “Stack Pointer” on page 41 |
| PSP | RW | Either | Unknown | “Stack Pointer” on page 41 |
| LR | RW | Either | 0xFFFFFFFF | “Link Register” on page 41 |
| PC | RW | Either | See description | “Program Counter” on page 41 |
| PSR | RW | Privileged | 0x01000000 | “Program Status Register” on page 42 |
| ASPR | RW | Either | 0x00000000 | “Application Program Status Register” on page 44 |
| IPSR | RO | Privileged | 0x00000000 | “Interrupt Program Status Register” on page 45 |
| EPSR | RO | Privileged | 0x01000000 | “Execution Program Status Register” on page 46 |
| PRIMASK | RW | Privileged | 0x00000000 | “Priority Mask Register” on page 48 |
| FAULTMASK | RW | Privileged | 0x00000000 | “Fault Mask Register” on page 49 |
| BASEPRI | RW | Privileged | 0x00000000 | “Base Priority Mask Register” on page 50 |
| CONTROL | RW | Privileged | 0x00000000 | “CONTROL register” on page 51 |

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

#### 11.4.3.1 General-purpose registers

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

#### 11.4.3.2 Stack Pointer

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

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

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

#### 11.4.3.3 Link Register

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

#### 11.4.3.4 Program Counter

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

#### 11.4.3.5 Program Status Register

The *Program Status Register* (PSR) combines:

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

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

- **APSR:**

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

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

- **IPSR:**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | ISR_NUMBER |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ISR_NUMBER | | | | | | | |

- **EPSR:**

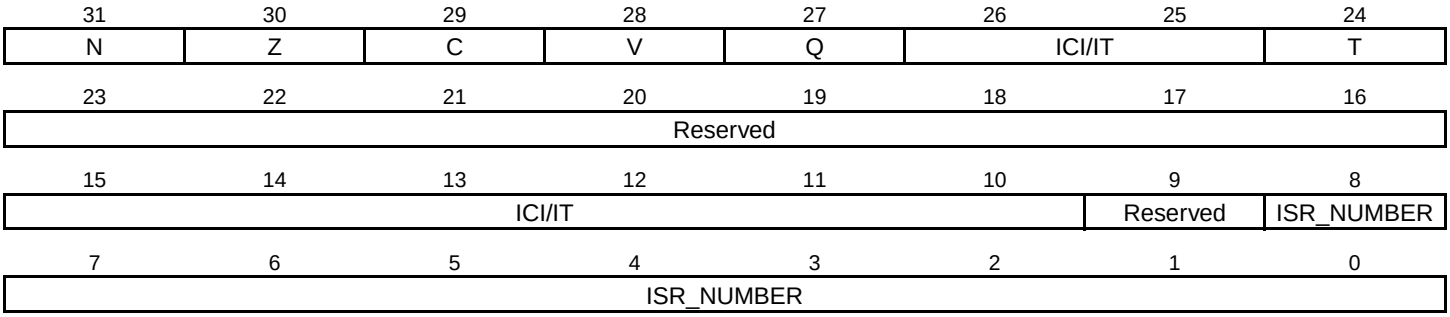
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | ICI/IT | | T |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| ICI/IT | | | | | | Reserved | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

The PSR bit assignments are:




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

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

The PSR combinations and attributes are:

**Table 11-3. PSR register combinations**

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

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

See the instruction descriptions “MRS” on page 136 and “MSR” on page 137 for more information about how to access the program status registers.

#### 11.4.3.6 Application Program Status Register

The APSR contains the current state of the condition flags from previous instruction executions. See the register summary in Table 11-2 on page 40 for its attributes. The bit assignments are:

- **N**

Negative or less than flag:

0 = operation result was positive, zero, greater than, or equal

1 = operation result was negative or less than.

- **Z**

Zero flag:

0 = operation result was not zero

1 = operation result was zero.

- **C**

Carry or borrow flag:

0 = add operation did not result in a carry bit or subtract operation resulted in a borrow bit

1 = add operation resulted in a carry bit or subtract operation did not result in a borrow bit.

- **V**

Overflow flag:

0 = operation did not result in an overflow

1 = operation resulted in an overflow.

- **Q**

Sticky saturation flag:

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

1 = indicates when an SSAT or USAT instruction results in saturation.

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

#### 11.4.3.7 Interrupt Program Status Register

The IPSR contains the exception type number of the current *Interrupt Service Routine* (ISR). See the register summary in Table 11-2 on page 40 for its attributes. The bit assignments are:

- **ISR_NUMBER**

This is the number of the current exception:

0 = Thread mode

1 = Reserved

2 = NMI

3 = Hard fault

4 = Memory management fault

5 = Bus fault

6 = Usage fault

7-10 = Reserved

11 = SVCall

12 = Reserved for Debug

13 = Reserved

14 = PendSV

15 = SysTick

16 = IRQ0

26 = IRQ32

see “Exception types” on page 62 for more information.

#### 11.4.3.8 Execution Program Status Register

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

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

See the register summary in Table 11-2 on page 40 for the EPSR attributes. The bit assignments are:

- **ICI**

Interruptible-continuable instruction bits, see "Interruptible-continuable instructions" on page 47.

- **IT**

Indicates the execution state bits of the IT instruction, see "IT" on page 127.

- **T**

Always set to 1.

Attempts to read the EPSR directly through application software using the MSR instruction always return zero. Attempts to write the EPSR using the MSR instruction in application software are ignored. Fault handlers can examine EPSR value in the stacked PSR to indicate the operation that is at fault. See "Exception entry and return" on page 66.

#### 11.4.3.9 Interruptible-continuable instructions

When an interrupt occurs during the execution of an LDM or STM instruction, the processor:

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

After servicing the interrupt, the processor:

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

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

#### 11.4.3.10 If-Then block

The If-Then block contains up to four instructions following a 16-bit IT instruction. Each instruction in the block is conditional. The conditions for the instructions are either all the same, or some can be the inverse of others. See “IT” on page 127 for more information.

#### 11.4.3.11 Exception mask registers

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

To access the exception mask registers use the MSR and MRS instructions, or the CPS instruction to change the value of PRIMASK or FAULTMASK. See “MRS” on page 136, “MSR” on page 137, and “CPS” on page 132 for more information.

#### 11.4.3.12 Priority Mask Register

The PRIMASK register prevents activation of all exceptions with configurable priority. See the register summary in Table 11-2 on page 40 for its attributes. The bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | PRIMASK |

- **PRIMASK**

0 = no effect

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

#### 11.4.3.13 Fault Mask Register

The FAULTMASK register prevents activation of all exceptions. See the register summary in Table 11-2 on page 40 for its attributes. The bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | FAULTMASK |

- **FAULTMASK**

0 = no effect

1 = prevents the activation of all exceptions.

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

#### 11.4.3.14 Base Priority Mask Register

The BASEPRI register defines the minimum priority for exception processing. When BASEPRI is set to a nonzero value, it prevents the activation of all exceptions with same or lower priority level as the BASEPRI value. See the register summary in Table 11-2 on page 40 for its attributes. The bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| BASEPRI | | | | | | | |

- **BASEPRI**

Priority mask bits:

0x0000 = no effect

Nonzero = defines the base priority for exception processing.

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

This field is similar to the priority fields in the interrupt priority registers. The processor implements only bits[7:4] of this field, bits[3:0] read as zero and ignore writes. See “Interrupt Priority Registers” on page 151 for more information. Remember that higher priority field values correspond to lower exception priorities.

#### 11.4.3.15 CONTROL register

The CONTROL register controls the stack used and the privilege level for software execution when the processor is in Thread mode. See the register summary in Table 11-2 on page 40 for its attributes. The bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

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

- **Active stack pointer**

Defines the current stack:

0 = MSP is the current stack pointer

1 = PSP is the current stack pointer.

In Handler mode this bit reads as zero and ignores writes.

- **Thread mode privilege level**

Defines the Thread mode privilege level:

0 = privileged

1 = unprivileged.

Handler mode always uses the MSP, so the processor ignores explicit writes to the active stack pointer bit of the CONTROL register when in Handler mode. The exception entry and return mechanisms update the CONTROL register.

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

By default, Thread mode uses the MSP. To switch the stack pointer used in Thread mode to the PSP, use the MSR instruction to set the Active stack pointer bit to 1, see “MSR” on page 137.

When changing the stack pointer, software must use an ISB instruction immediately after the MSR instruction. This ensures that instructions after the ISB execute using the new stack pointer. See “ISB” on page 135

### 11.4.4 Exceptions and interrupts

The Cortex-M3 processor supports interrupts and system exceptions. The processor and the *Nested Vectored Interrupt Controller* (NVIC) prioritize and handle all exceptions. An exception changes the normal flow of software control. The processor uses handler mode to handle all exceptions except for reset. See “Exception entry” on page 67 and “Exception return” on page 68 for more information.

The NVIC registers control interrupt handling. See “Nested Vectored Interrupt Controller” on page 144 for more information.

### 11.4.5 Data types

The processor:

- supports the following data types:
  - 32-bit words
  - 16-bit halfwords
  - 8-bit bytes
- supports 64-bit data transfer instructions.
- manages all data memory accesses as little-endian. Instruction memory and *Private Peripheral Bus* (PPB) accesses are always little-endian. See “Memory regions, types and attributes” on page 54 for more information.
-

### 11.4.6 The Cortex Microcontroller Software Interface Standard

For a Cortex-M3 microcontroller system, the *Cortex Microcontroller Software Interface Standard* (CMSIS) defines:

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

The CMSIS includes address definitions and data structures for the core peripherals in the Cortex-M3 processor. It also includes optional interfaces for middleware components comprising a TCP/IP stack and a Flash file system.

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

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

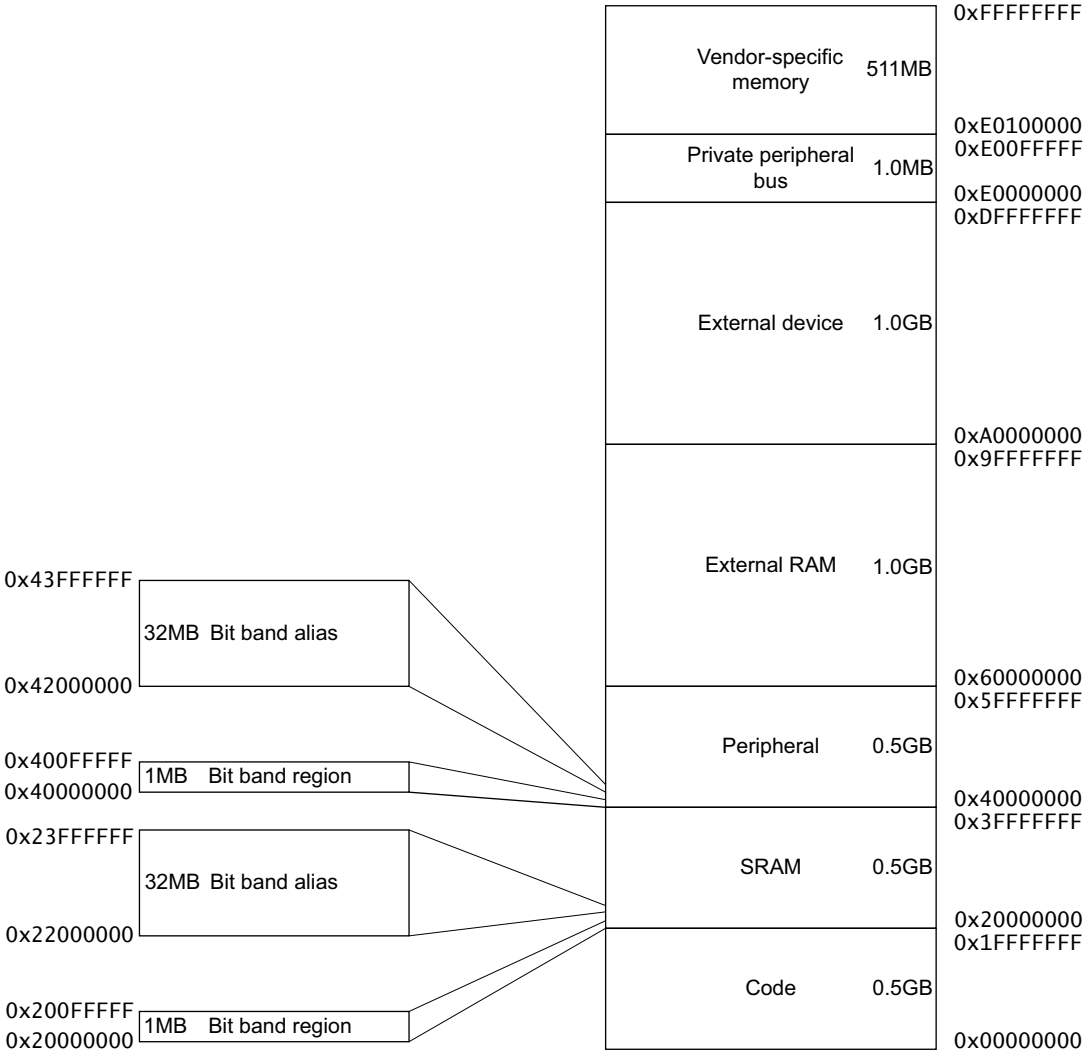
This document uses the register short names defined by the CMSIS. In a few cases these differ from the architectural short names that might be used in other documents.

The following sections give more information about the CMSIS:

- “Power management programming hints” on page 71
- “Intrinsic functions” on page 76
- “The CMSIS mapping of the Cortex-M3 NVIC registers” on page 144
- “NVIC programming hints” on page 156.

## 11.5 Memory model

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




The regions for SRAM and peripherals include bit-band regions. Bit-banding provides atomic operations to bit data, see “Bit-banding” on page 57.

The processor reserves regions of the *Private peripheral bus* (PPB) address range for core peripheral registers, see “About the Cortex-M3 peripherals” on page 143.

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

### 11.5.1 Memory regions, types and attributes

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

The memory types are:

#### 11.5.1.1 Normal

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

#### 11.5.1.2 Device

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

#### 11.5.1.3 Strongly-ordered

The processor preserves transaction order relative to all other transactions.

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

The additional memory attributes include.

#### 11.5.1.4 Shareable

For a shareable memory region, the memory system provides data synchronization between bus masters in a system with multiple bus masters, for example, a processor with a DMA controller.

Strongly-ordered memory is always shareable.

If multiple bus masters can access a non-shareable memory region, software must ensure data coherency between the bus masters.

#### 11.5.1.5 Execute Never (XN)

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

### 11.5.2 Memory system ordering of memory accesses

For most memory accesses caused by explicit memory access instructions, the memory system does not guarantee that the order in which the accesses complete matches the program order of the instructions, providing this does not affect the behavior of the instruction sequence. Normally, if correct program execution depends on two memory accesses completing in program order, software must insert a memory barrier instruction between the memory access instructions, see “Software ordering of memory accesses” on page 56.

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

| A1 \ A2 | Normal access | Device access | | Strongly-ordered access |
|---|---|---|---|---|
| | | Non-shareable | Shareable | |
| Normal access | - | - | - | - |
| Device access, non-shareable | - | < | - | < |
| Device access, shareable | - | - | < | < |
| Strongly-ordered access | - | < | < | < |

Where:

- Means that the memory system does not guarantee the ordering of the accesses.

< Means that accesses are observed in program order, that is, A1 is always observed before A2.

### 11.5.3 Behavior of memory accesses

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

**Table 11-4. Memory access behavior**

| Address range | Memory region | Memory type | XN | Description |
|---|---|---|---|---|
| 0x00000000-0x1FFFFFFF | Code | Normal (1) | - | Executable region for program code. You can also put data here. |
| 0x20000000-0x3FFFFFFF | SRAM | Normal (1) | - | Executable region for data. You can also put code here.<br>This region includes bit band and bit band alias areas, see Table 11-6 on page 58. |
| 0x40000000-0x5FFFFFFF | Peripheral | Device (1) | XN | This region includes bit band and bit band alias areas, see Table 11-6 on page 58. |
| 0x60000000-0x9FFFFFFF | External RAM | Normal (1) | - | Executable region for data. |
| 0xA0000000-0xDFFFFFFF | External device | Device (1) | XN | External Device memory |
| 0xE0000000-0xE00FFFFF | Private Peripheral Bus | Strongly- ordered (1) | XN | This region includes the NVIC, System timer, and system control block. |
| 0xE0100000-0xFFFFFFFF | Reserved | Device (1) | XN | Reserved |

1. See “Memory regions, types and attributes” on page 54 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.

#### 11.5.3.1 Additional memory access constraints for shared memory

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

**Table 11-5. Memory region share ability policies**

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

1. See “Memory regions, types and attributes” on page 54 for more information.
2. The Peripheral and Vendor-specific device regions have no additional access constraints.

### 11.5.4 Software ordering of memory accesses

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

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

"Memory system ordering of memory accesses" on page 55 describes the cases where the memory system guarantees the order of memory accesses. Otherwise, if the order of memory accesses is critical, software must include memory barrier instructions to force that ordering. The processor provides the following memory barrier instructions:

#### 11.5.4.1 DMB

The *Data Memory Barrier* (DMB) instruction ensures that outstanding memory transactions complete before subsequent memory transactions. See "DMB" on page 133.

#### 11.5.4.2 DSB

The *Data Synchronization Barrier* (DSB) instruction ensures that outstanding memory transactions complete before subsequent instructions execute. See "DSB" on page 134.

#### 11.5.4.3 ISB

The *Instruction Synchronization Barrier* (ISB) ensures that the effect of all completed memory transactions is recognizable by subsequent instructions. See "ISB" on page 135.

Use memory barrier instructions in, for example:

- Vector table. If the program changes an entry in the vector table, and then enables the corresponding exception, use a DMB instruction between the operations. This ensures that if the exception is taken immediately after being enabled the processor uses the new exception vector.
- Self-modifying code. If a program contains self-modifying code, use an ISB instruction immediately after the code modification in the program. This ensures subsequent instruction execution uses the updated program.
- Memory map switching. If the system contains a memory map switching mechanism, use a DSB instruction after switching the memory map in the program. This ensures subsequent instruction execution uses the updated memory map.
- Dynamic exception priority change. When an exception priority has to change when the exception is pending or active, use DSB instructions after the change. This ensures the change takes effect on completion of the DSB instruction.
- Using a semaphore in multi-master system. If the system contains more than one bus master, for example, if another processor is present in the system, each processor must use a DMB instruction after any semaphore instructions, to ensure other bus masters see the memory transactions in the order in which they were executed.

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

### 11.5.5 Bit-banding

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

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

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

**Table 11-6. SRAM memory bit-banding regions**

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

**Table 11-7. Peripheral memory bit-banding regions**

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

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

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

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

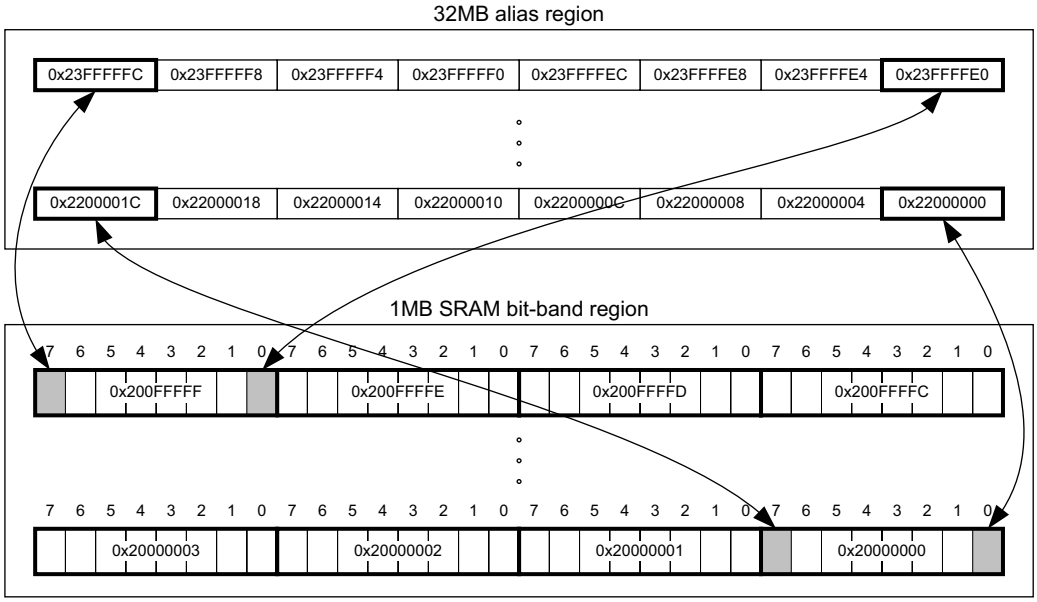
where:

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

Figure 11-2 shows examples of bit-band mapping between the SRAM bit-band alias region and the SRAM bit-band region:

- The alias word at 0x23FFFFE0 maps to bit[0] of the bit-band byte at 0x200FFFFF: 0x23FFFFE0 = 0x22000000 + (0xFFFFF*32) + (0*4).
- The alias word at 0x23FFFFFC maps to bit[7] of the bit-band byte at 0x200FFFFF: 0x23FFFFFC = 0x22000000 + (0xFFFFF*32) + (7*4).
- The alias word at 0x22000000 maps to bit[0] of the bit-band byte at 0x20000000: 0x22000000 = 0x22000000 + (0*32) + (0 *4).
- The alias word at 0x2200001C maps to bit[7] of the bit-band byte at 0x20000000: 0x2200001C = 0x22000000+ (0*32) + (7*4).

**Figure 11-2. Bit-band mapping**




### 11.5.5.1 Directly accessing an alias region

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

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

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

Reading a word in the alias region:

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

### 11.5.5.2 Directly accessing a bit-band region

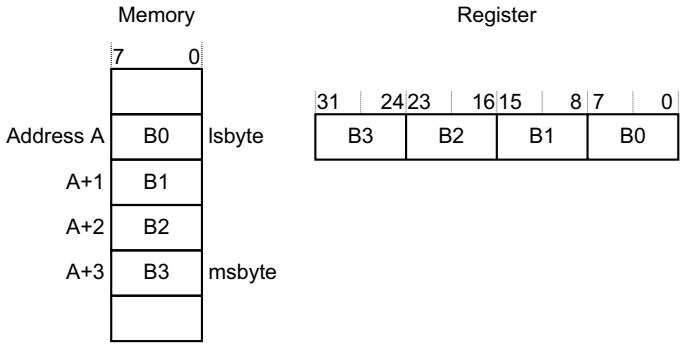
"Behavior of memory accesses" on page 55 describes the behavior of direct byte, halfword, or word accesses to the bit-band regions.

## 11.5.6 Memory endianness

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

#### 11.5.6.1 Little-endian format

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




### 11.5.7 Synchronization primitives

The Cortex-M3 instruction set includes pairs of *synchronization primitives*. These provide a non-blocking mechanism that a thread or process can use to obtain exclusive access to a memory location. Software can use them to perform a guaranteed read-modify-write memory update sequence, or for a semaphore mechanism.

A pair of synchronization primitives comprises:

#### 11.5.7.1 A Load-Exclusive instruction

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

#### 11.5.7.2 A Store-Exclusive instruction

Used to attempt to write to the same memory location, returning a status bit to a register. If this bit is:

0: it indicates that the thread or process gained exclusive access to the memory, and the write succeeds,

1: it indicates that the thread or process did not gain exclusive access to the memory, and no write is performed,

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

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

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

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

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

  0: The read-modify-write completed successfully,

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

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

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

The Cortex-M3 includes an exclusive access monitor, that tags the fact that the processor has executed a Load-Exclusive instruction. If the processor is part of a multiprocessor system, the system also globally tags the memory locations addressed by exclusive accesses by each processor.

The processor removes its exclusive access tag if:

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

In a multiprocessor implementation:

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

For more information about the synchronization primitive instructions, see “LDREX and STREX” on page 97 and “CLREX” on page 99.

### 11.5.8 Programming hints for the synchronization primitives

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

**Table 11-8. C compiler intrinsic functions for exclusive access instructions**

| Instruction | Intrinsic function |
|---|---|
| LDREX, LDREXH, or LDREXB | unsigned int __ldrex(volatile void *ptr) |
| STREX, STREXH, or STREXB | int __strex(unsigned int val, volatile void *ptr) |
| CLREX | void __clrex(void) |

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

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

## 11.6 Exception model

This section describes the exception model.

### 11.6.1 Exception states

Each exception is in one of the following states:

#### 11.6.1.1 Inactive

The exception is not active and not pending.

#### 11.6.1.2 Pending

The exception is waiting to be serviced by the processor.

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

#### 11.6.1.3 Active

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

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

#### 11.6.1.4 Active and pending

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

### 11.6.2 Exception types

The exception types are:

#### 11.6.2.1 Reset

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

#### 11.6.2.2 Non Maskable Interrupt (NMI)

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

NMIs cannot be:

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

#### 11.6.2.3 Hard fault

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

#### 11.6.2.4 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.

#### 11.6.2.5 Usage fault

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

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

The following can cause a usage fault when the core is configured to report them:

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

#### 11.6.2.6 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.

#### 11.6.2.7 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.

#### 11.6.2.8 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.

#### 11.6.2.9 Interrupt (IRQ)

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

**Table 11-9. Properties of the different exception types**

| Exception number [1] | IRQ number [1] | Exception type | Priority | Vector address or offset [2] | Activation |
|---|---|---|---|---|---|
| 1 | - | Reset | -3, the highest | 0x00000004 | Asynchronous |
| 2 | -14 | NMI | -2 | 0x00000008 | Asynchronous |
| 3 | -13 | Hard fault | -1 | 0x0000000C | - |
| 4 | -12 | Memory management fault | Configurable [3] | 0x00000010 | Synchronous |
| 5 | -11 | Bus fault | Configurable [3] | 0x00000014 | Synchronous when precise, asynchronous when imprecise |
| 6 | -10 | Usage fault | Configurable [3] | 0x00000018 | Synchronous |
| 7-10 | - | - | - | Reserved | - |
| 11 | -5 | SVCall | Configurable [3] | 0x0000002C | Synchronous |
| 12-13 | - | - | - | Reserved | - |
| 14 | -2 | PendSV | Configurable [3] | 0x00000038 | Asynchronous |
| 15 | -1 | SysTick | Configurable [3] | 0x0000003C | Asynchronous |
| 16 and above | 0 and above [4] | Interrupt (IRQ) | Configurable [5] | 0x00000040 and above [6] | Asynchronous |

1. To simplify the software layer, the CMSIS only uses IRQ numbers and therefore uses negative values for exceptions other than interrupts. The IPSR returns the Exception number, see "Interrupt Program Status Register" on page 45.
2. See "Vector table" on page 65 for more information.
3. See "System Handler Priority Registers" on page 170.
4. See the "Peripheral Identifiers" section of the datasheet.
5. See "Interrupt Priority Registers" on page 151.
6. Increasing in steps of 4.

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

Privileged software can disable the exceptions that Table 11-9 on page 64 shows as having configurable priority, see:

- "System Handler Control and State Register" on page 174
- "Interrupt Clear-enable Registers" on page 147.

For more information about hard faults, memory management faults, bus faults, and usage faults, see "Fault handling" on page 68.

### 11.6.3 Exception handlers

The processor handles exceptions using:

#### 11.6.3.1 Interrupt Service Routines (ISRs)

Interrupts IRQ0 to IRQ32 are the exceptions handled by ISRs.

#### 11.6.3.2 Fault handlers

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

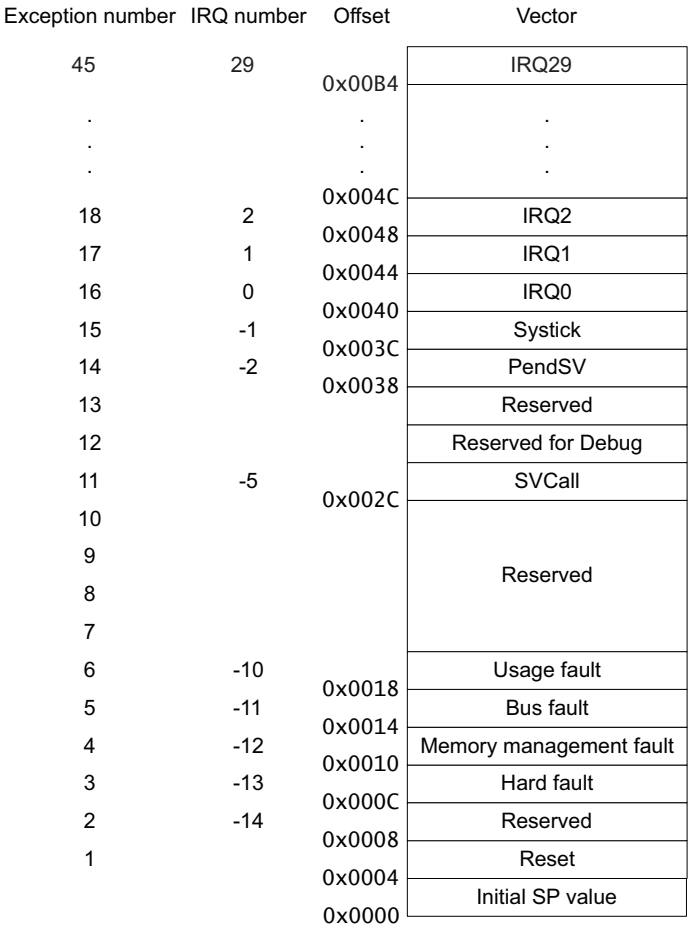
### 11.6.3.3 System handlers

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

## 11.6.4 Vector table

The vector table contains the reset value of the stack pointer, and the start addresses, also called exception vectors, for all exception handlers. Figure 11-3 on page 65 shows the order of the exception vectors in the vector table. The least-significant bit of each vector must be 1, indicating that the exception handler is Thumb code.

**Figure 11-3. Vector table**




On system reset, the vector table is fixed at address 0x00000000. Privileged software can write to the VTOR to relocate the vector table start address to a different memory location, in the range 0x00000080 to 0x3FFFFF80, see "Vector Table Offset Register" on page 163.

### 11.6.5 Exception priorities

As Table 11-9 on page 64 shows, all exceptions have an associated priority, with:

- a lower priority value indicating a higher priority
- configurable priorities for all exceptions except Reset, Hard fault.

If software does not configure any priorities, then all exceptions with a configurable priority have a priority of 0. For information about configuring exception priorities see

- "System Handler Priority Registers" on page 170
- "Interrupt Priority Registers" on page 151.

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

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

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

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

### 11.6.6 Interrupt priority grouping

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

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

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

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

For information about splitting the interrupt priority fields into group priority and subpriority, see "Application Interrupt and Reset Control Register" on page 164.

### 11.6.7 Exception entry and return

Descriptions of exception handling use the following terms:

#### 11.6.7.1 Preemption

When the processor is executing an exception handler, an exception can preempt the exception handler if its priority is higher than the priority of the exception being handled. See "Interrupt priority grouping" on page 66 for more information about preemption by an interrupt.

When one exception preempts another, the exceptions are called nested exceptions. See "Exception entry" on page 67 more information.

#### 11.6.7.2 Return

This occurs when the exception handler is completed, and:

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

The processor pops the stack and restores the processor state to the state it had before the interrupt occurred. See “Exception return” on page 68 for more information.

#### 11.6.7.3 Tail-chaining

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

#### 11.6.7.4 Late-arriving

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

#### 11.6.7.5 Exception entry

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

- the processor is in Thread mode
- the new exception is of higher priority than the exception being handled, in which case the new exception preempts the original exception.

When one exception preempts another, the exceptions are nested.

Sufficient priority means the exception has more priority than any limits set by the mask registers, see “Exception mask registers” on page 47. An exception with less priority than this is pending but is not handled by the processor.

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

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

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

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

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

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

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

#### 11.6.7.6 Exception return

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

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

EXC_RETURN is the value loaded into the LR on exception entry. The exception mechanism relies on this value to detect when the processor has completed an exception handler. The lowest four bits of this value provide information on the return stack and processor mode. Table 11-10 shows the EXC_RETURN[3:0] values with a description of the exception return behavior.

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

**Table 11-10. Exception return behavior**

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

## 11.7 Fault handling

Faults are a subset of the exceptions, see “Exception model” on page 62. The following generate a fault:

- a bus error on:
  - an instruction fetch or vector table load
  - a data access
- an internally-detected error such as an undefined instruction or an attempt to change state with a BX instruction
- attempting to execute an instruction from a memory region marked as *Non-Executable* (XN).

### 11.7.1 Fault types

Table 11-11 shows the types of fault, the handler used for the fault, the corresponding fault status register, and the register bit that indicates that the fault has occurred. See “Configurable Fault Status Register” on page 176 for more information about the fault status registers.

**Table 11-11. Faults**

| Fault | Handler | Bit name | Fault status register |
|---|---|---|---|
| Bus error on a vector read | Hard fault | VECTTBL | “Hard Fault Status Register” on page 182 |
| Fault escalated to a hard fault | | FORCED | |
| Bus error: | Bus fault | - | - |
| during exception stacking | | STKERR | “Bus Fault Status Register” on page 178 |
| during exception unstacking | | UNSTKERR | |
| during instruction prefetch | | IBUSERR | |
| Precise data bus error | | PRECISERR | |
| Imprecise data bus error | | IMPRECISERR | |
| Attempt to access a coprocessor | Usage fault | NOCP | “Usage Fault Status Register” on page 180 |
| Undefined instruction | | UNDEFINSTR | |
| Attempt to enter an invalid instruction set state (1) | | INVSTATE | |
| Invalid EXC_RETURN value | | INVPC | |
| Illegal unaligned load or store | | UNALIGNED | |
| Divide By 0 | | DIVBYZERO | |

1. Attempting to use an instruction set other than the Thumb instruction set.

### 11.7.2 Fault escalation and hard faults

All faults exceptions except for hard fault have configurable exception priority, see “System Handler Priority Registers” on page 170. Software can disable execution of the handlers for these faults, see “System Handler Control and State Register” on page 174.

Usually, the exception priority, together with the values of the exception mask registers, determines whether the processor enters the fault handler, and whether a fault handler can preempt another fault handler. as described in “Exception model” on page 62.

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

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

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

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

### 11.7.3 Fault status registers and fault address registers

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

**Table 11-12. Fault status and fault address registers**

| Handler | Status register name | Address register name | Register description |
|---|---|---|---|
| Hard fault | HFSR | - | "Hard Fault Status Register" on page 182 |
| Memory management fault | MMFSR | MMFAR | "Memory Management Fault Status Register" on page 177<br>"Memory Management Fault Address Register" on page 183 |
| Bus fault | BFSR | BFAR | "Bus Fault Status Register" on page 178<br>"Bus Fault Address Register" on page 184 |
| Usage fault | UFSR | - | "Usage Fault Status Register" on page 180 |

### 11.7.4 Lockup

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

- it is reset

## 11.8 Power management

The Cortex-M3 processor sleep modes reduce power consumption:

- Backup Mode
- Wait Mode
- Sleep Mode

The SLEEPDEEP bit of the SCR selects which sleep mode is used, see "System Control Register" on page 167. For more information about the behavior of the sleep modes see "Low Power Modes" in the PMC section of the datasheet.

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

### 11.8.1 Entering sleep mode

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

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

#### 11.8.1.1 Wait for interrupt

The *wait for interrupt* instruction, WFI, causes immediate entry to sleep mode. When the processor executes a WFI instruction it stops executing instructions and enters sleep mode. See “WFI” on page 142 for more information.

#### 11.8.1.2 Wait for event

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

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

See “WFE” on page 141 for more information.

#### 11.8.1.3 Sleep-on-exit

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

### 11.8.2 Wakeup from sleep mode

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

#### 11.8.2.1 Wakeup from WFI or sleep-on-exit

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

Some embedded systems might have to execute system restore tasks after the processor wakes up, and before it executes an interrupt handler. To achieve this set the PRIMASK bit to 1 and the FAULTMASK bit to 0. If an interrupt arrives that is enabled and has a higher priority than current exception priority, the processor wakes up but does not execute the interrupt handler until the processor sets PRIMASK to zero. For more information about PRIMASK and FAULTMASK see “Exception mask registers” on page 47.

#### 11.8.2.2 Wakeup from WFE

The processor wakes up if:

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

In addition, if the SEVONPEND bit in the SCR is set to 1, any new pending interrupt triggers an event and wakes up the processor, even if the interrupt is disabled or has insufficient priority to cause exception entry. For more information about the SCR see “System Control Register” on page 167.

### 11.8.3 Power management programming hints

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

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

## 11.9 Instruction set summary

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

In Table 11-13:

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

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

**Table 11-13. Cortex-M3 instructions**

| Mnemonic | Operands | Brief description | Flags | Page |
|---|---|---|---|---|
| ADC, ADCS | {Rd,} Rn, Op2 | Add with Carry | N,Z,C,V | page 101 |
| ADD, ADDS | {Rd,} Rn, Op2 | Add | N,Z,C,V | page 101 |
| ADD, ADDW | {Rd,} Rn, #imm12 | Add | N,Z,C,V | page 101 |
| ADR | Rd, label | Load PC-relative address | - | page 86 |
| AND, ANDS | {Rd,} Rn, Op2 | Logical AND | N,Z,C | page 103 |
| ASR, ASRS | Rd, Rm, <Rs\|#n> | Arithmetic Shift Right | N,Z,C | page 104 |
| B | label | Branch | - | page 124 |
| BFC | Rd, #lsb, #width | Bit Field Clear | - | page 120 |
| BFI | Rd, Rn, #lsb, #width | Bit Field Insert | - | page 120 |
| BIC, BICS | {Rd,} Rn, Op2 | Bit Clear | N,Z,C | page 103 |
| BKPT | #imm | Breakpoint | - | page 131 |
| BL | label | Branch with Link | - | page 124 |
| BLX | Rm | Branch indirect with Link | - | page 124 |
| BX | Rm | Branch indirect | - | page 124 |
| CBNZ | Rn, label | Compare and Branch if Non Zero | - | page 126 |
| CBZ | Rn, label | Compare and Branch if Zero | - | page 126 |
| CLREX | - | Clear Exclusive | - | page 99 |
| CLZ | Rd, Rm | Count leading zeros | - | page 106 |
| CMN, CMNS | Rn, Op2 | Compare Negative | N,Z,C,V | page 107 |
| CMP, CMPS | Rn, Op2 | Compare | N,Z,C,V | page 107 |
| CPSID | iflags | Change Processor State, Disable Interrupts | - | page 132 |
| CPSIE | iflags | Change Processor State, Enable Interrupts | - | page 132 |
| DMB | - | Data Memory Barrier | - | page 133 |
| DSB | - | Data Synchronization Barrier | - | page 134 |
| EOR, EORS | {Rd,} Rn, Op2 | Exclusive OR | N,Z,C | page 103 |
| ISB | - | Instruction Synchronization Barrier | - | page 135 |
| IT | - | If-Then condition block | - | page 127 |
| LDM | Rn{!}, reglist | Load Multiple registers, increment after | - | page 94 |

**Table 11-13. Cortex-M3 instructions (Continued)**

| Mnemonic | Operands | Brief description | Flags | Page |
|---|---|---|---|---|
| LDMDB, LDMEA | Rn{!}, reglist | Load Multiple registers, decrement before | - | page 94 |
| LDMFD, LDMIA | Rn{!}, reglist | Load Multiple registers, increment after | - | page 94 |
| LDR | Rt, [Rn, #offset] | Load Register with word | - | page 89 |
| LDRB, LDRBT | Rt, [Rn, #offset] | Load Register with byte | - | page 89 |
| LDRD | Rt, Rt2, [Rn, #offset] | Load Register with two bytes | - | page 89 |
| LDREX | Rt, [Rn, #offset] | Load Register Exclusive | - | page 89 |
| LDREXB | Rt, [Rn] | Load Register Exclusive with byte | - | page 89 |
| LDREXH | Rt, [Rn] | Load Register Exclusive with halfword | - | page 89 |
| LDRH, LDRHT | Rt, [Rn, #offset] | Load Register with halfword | - | page 89 |
| LDRSB, LDRSBT | Rt, [Rn, #offset] | Load Register with signed byte | - | page 89 |
| LDRSH, LDRSHT | Rt, [Rn, #offset] | Load Register with signed halfword | - | page 89 |
| LDRT | Rt, [Rn, #offset] | Load Register with word | - | page 89 |
| LSL, LSLS | Rd, Rm, <Rs\|#n> | Logical Shift Left | N,Z,C | page 104 |
| LSR, LSRS | Rd, Rm, <Rs\|#n> | Logical Shift Right | N,Z,C | page 104 |
| MLA | Rd, Rn, Rm, Ra | Multiply with Accumulate, 32-bit result | - | page 114 |
| MLS | Rd, Rn, Rm, Ra | Multiply and Subtract, 32-bit result | - | page 114 |
| MOV, MOVS | Rd, Op2 | Move | N,Z,C | page 108 |
| MOVT | Rd, #imm16 | Move Top | - | page 110 |
| MOVW, MOV | Rd, #imm16 | Move 16-bit constant | N,Z,C | page 108 |
| MRS | Rd, spec_reg | Move from special register to general register | - | page 136 |
| MSR | spec_reg, Rm | Move from general register to special register | N,Z,C,V | page 137 |
| MUL, MULS | {Rd,} Rn, Rm | Multiply, 32-bit result | N,Z | page 114 |
| MVN, MVNS | Rd, Op2 | Move NOT | N,Z,C | page 108 |
| NOP | - | No Operation | - | page 138 |
| ORN, ORNS | {Rd,} Rn, Op2 | Logical OR NOT | N,Z,C | page 103 |
| ORR, ORRS | {Rd,} Rn, Op2 | Logical OR | N,Z,C | page 103 |
| POP | reglist | Pop registers from stack | - | page 96 |
| PUSH | reglist | Push registers onto stack | - | page 96 |
| RBIT | Rd, Rn | Reverse Bits | - | page 111 |
| REV | Rd, Rn | Reverse byte order in a word | - | page 111 |
| REV16 | Rd, Rn | Reverse byte order in each halfword | - | page 111 |
| REVSH | Rd, Rn | Reverse byte order in bottom halfword and sign extend | - | page 111 |
| ROR, RORS | Rd, Rm, <Rs\|#n> | Rotate Right | N,Z,C | page 104 |
| RRX, RRXS | Rd, Rm | Rotate Right with Extend | N,Z,C | page 104 |

**Table 11-13. Cortex-M3 instructions (Continued)**

| Mnemonic | Operands | Brief description | Flags | Page |
|---|---|---|---|---|
| RSB, RSBS | {Rd,} Rn, Op2 | Reverse Subtract | N,Z,C,V | page 101 |
| SBC, SBCS | {Rd,} Rn, Op2 | Subtract with Carry | N,Z,C,V | page 101 |
| SBFX | Rd, Rn, #lsb, #width | Signed Bit Field Extract | - | page 121 |
| SDIV | {Rd,} Rn, Rm | Signed Divide | - | page 116 |
| SEV | - | Send Event | - | page 139 |
| SMLAL | RdLo, RdHi, Rn, Rm | Signed Multiply with Accumulate (32 x 32 + 64), 64-bit result | - | page 115 |
| SMULL | RdLo, RdHi, Rn, Rm | Signed Multiply (32 x 32), 64-bit result | - | page 115 |
| SSAT | Rd, #n, Rm {,shift #s} | Signed Saturate | Q | page 117 |
| STM | Rn{!}, reglist | Store Multiple registers, increment after | - | page 94 |
| STMDB, STMEA | Rn{!}, reglist | Store Multiple registers, decrement before | - | page 94 |
| STMFD, STMIA | Rn{!}, reglist | Store Multiple registers, increment after | - | page 94 |
| STR | Rt, [Rn, #offset] | Store Register word | - | page 89 |
| STRB, STRBT | Rt, [Rn, #offset] | Store Register byte | - | page 89 |
| STRD | Rt, Rt2, [Rn, #offset] | Store Register two words | - | page 89 |
| STREX | Rd, Rt, [Rn, #offset] | Store Register Exclusive | - | page 97 |
| STREXB | Rd, Rt, [Rn] | Store Register Exclusive byte | - | page 97 |
| STREXH | Rd, Rt, [Rn] | Store Register Exclusive halfword | - | page 97 |
| STRH, STRHT | Rt, [Rn, #offset] | Store Register halfword | - | page 89 |
| STRT | Rt, [Rn, #offset] | Store Register word | - | page 89 |
| SUB, SUBS | {Rd,} Rn, Op2 | Subtract | N,Z,C,V | page 101 |
| SUB, SUBW | {Rd,} Rn, #imm12 | Subtract | N,Z,C,V | page 101 |
| SVC | #imm | Supervisor Call | - | page 140 |
| SXTB | {Rd,} Rm {,ROR #n} | Sign extend a byte | - | page 122 |
| SXTH | {Rd,} Rm {,ROR #n} | Sign extend a halfword | - | page 122 |
| TBB | [Rn, Rm] | Table Branch Byte | - | page 129 |
| TBH | [Rn, Rm, LSL #1] | Table Branch Halfword | - | page 129 |
| TEQ | Rn, Op2 | Test Equivalence | N,Z,C | page 112 |
| TST | Rn, Op2 | Test | N,Z,C | page 112 |
| UBFX | Rd, Rn, #lsb, #width | Unsigned Bit Field Extract | - | page 121 |
| UDIV | {Rd,} Rn, Rm | Unsigned Divide | - | page 116 |
| UMLAL | RdLo, RdHi, Rn, Rm | Unsigned Multiply with Accumulate (32 x 32 + 64), 64-bit result | - | page 115 |
| UMULL | RdLo, RdHi, Rn, Rm | Unsigned Multiply (32 x 32), 64-bit result | - | page 115 |
| USAT | Rd, #n, Rm {,shift #s} | Unsigned Saturate | Q | page 117 |
| UXTB | {Rd,} Rm {,ROR #n} | Zero extend a byte | - | page 122 |
| UXTH | {Rd,} Rm {,ROR #n} | Zero extend a halfword | - | page 122 |

**Table 11-13. Cortex-M3 instructions (Continued)**

| Mnemonic | Operands | Brief description | Flags | Page |
|---|---|---|---|---|
| WFE | - | Wait For Event | - | page 141 |
| WFI | - | Wait For Interrupt | - | page 142 |

## 11.10 Intrinsic functions

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

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

**Table 11-14. CMSIS intrinsic functions to generate some Cortex-M3 instructions**

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

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

**Table 11-15. CMSIS intrinsic functions to access the special registers**

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

## 11.11 About the instruction descriptions

The following sections give more information about using the instructions:

- “Operands” on page 77
- “Restrictions when using PC or SP” on page 77
- “Flexible second operand” on page 77
- “Shift Operations” on page 78
- “Address alignment” on page 81
- “PC-relative expressions” on page 81
- “Conditional execution” on page 81
- “Instruction width selection” on page 83.

### 11.11.1 Operands

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

Operands in some instructions are flexible in that they can either be a register or a constant. See “Flexible second operand” .

### 11.11.2 Restrictions when using PC or SP

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

Bit[0] of any address you write to the PC with a BX, BLX, LDM, LDR, or POP instruction must be 1 for correct execution, because this bit indicates the required instruction set, and the Cortex-M3 processor only supports Thumb instructions.

### 11.11.3 Flexible second operand

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

*Operand2* can be a:

- “Constant”
- “Register with optional shift” on page 78

#### 11.11.3.1 Constant

You specify an Operand2 constant in the form:

```
#constant
```

where `constant` can be:

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

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

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

When an Operand2 constant is used with the instructions MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ or TST, the carry flag is updated to bit[31] of the constant, if the constant is greater than 255 and can be

produced by shifting an 8-bit value. These instructions do not affect the carry flag if Operand2 is any other constant.

#### 11.11.3.2 Instruction substitution

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

#### 11.11.3.3 Register with optional shift

You specify an Operand2 register in the form:

*Rm {, shift}*

where:

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

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

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

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

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

### 11.11.4 Shift Operations

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

- directly by the instructions ASR, LSR, LSL, ROR, and RRX, and the result is written to a destination register
- during the calculation of *Operand2* by the instructions that specify the second operand as a register with shift, see “Flexible second operand” on page 77. The result is used by the instruction.

The permitted shift lengths depend on the shift type and the instruction, see the individual instruction description or “Flexible second operand” on page 77. If the shift length is 0, no shift occurs. Register shift operations update the carry flag except when the specified shift length is 0. The following sub-sections describe the various shift operations and how they affect the carry flag. In these descriptions, *Rm* is the register containing the value to be shifted, and *n* is the shift length.

### 11.11.4.1 ASR

Arithmetic shift right by *n* bits moves the left-hand 32-n bits of the register *Rm*, to the right by *n* places, into the right-hand 32-n bits of the result. And it copies the original bit[31] of the register into the left-hand *n* bits of the result. See Figure 11-4 on page 79.

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

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

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

**Figure 11-4. ASR #3**


### 11.11.4.2 LSR

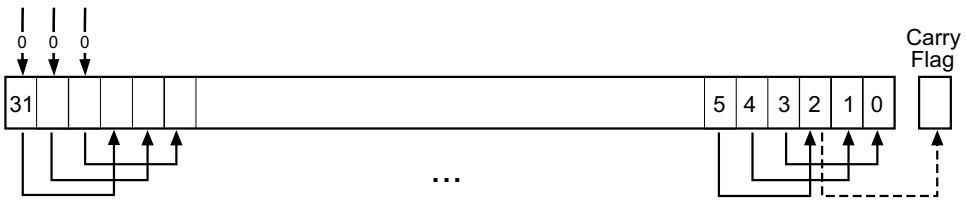
Logical shift right by n bits moves the left-hand 32-n bits of the register Rm, to the right by *n* places, into the right-hand 32-n bits of the result. And it sets the left-hand n bits of the result to 0. See Figure 11-5.

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

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

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

**Figure 11-5. LSR #3**



### 11.11.4.3 LSL

Logical shift left by *n* bits moves the right-hand 32-n bits of the register Rm, to the left by *n* places, into the left-hand 32-n bits of the result. And it sets the right-hand *n* bits of the result to 0. See Figure 11-6 on page 80.

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

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

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

**Figure 11-6. LSL #3**

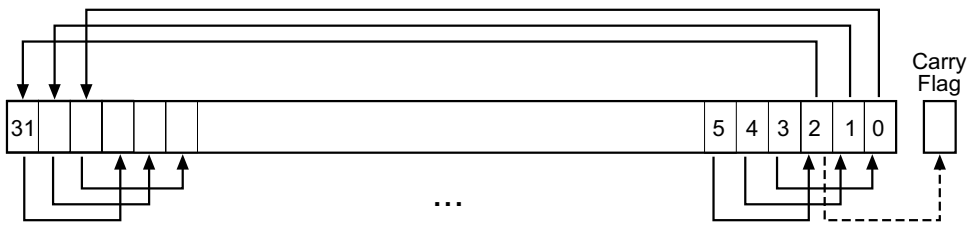

### 11.11.4.4 ROR

Rotate right by *n* bits moves the left-hand 32-*n* bits of the register *Rm*, to the right by *n* places, into the right-hand 32-*n* bits of the result. And it moves the right-hand *n* bits of the register into the left-hand *n* bits of the result. See Figure 11-7.

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

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

**Figure 11-7. ROR #3**

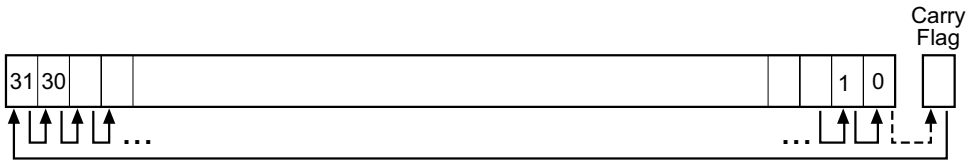


#### 11.11.4.5 RRX

Rotate right with extend moves the bits of the register *Rm* to the right by one bit. And it copies the carry flag into bit[31] of the result. See Figure 11-8 on page 81.

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

**Figure 11-8. RRX**




### 11.11.5 Address alignment

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

The Cortex-M3 processor supports unaligned access only for the following instructions:

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

All other load and store instructions generate a usage fault exception if they perform an unaligned access, and therefore their accesses must be address aligned. For more information about usage faults see “Fault handling” on page 68.

Unaligned accesses are usually slower than aligned accesses. In addition, some memory regions might not support unaligned accesses. Therefore, ARM recommends that programmers ensure that accesses are aligned. To avoid accidental generation of unaligned accesses, use the UNALIGN_TRP bit in the Configuration and Control Register to trap all unaligned accesses, see “Configuration and Control Register” on page 168.

### 11.11.6 PC-relative expressions

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

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

### 11.11.7 Conditional execution

Most data processing instructions can optionally update the condition flags in the *Application Program Status Register* (APSR) according to the result of the operation, see “Application Program Status Register” on page 44.

Some instructions update all flags, and some only update a subset. If a flag is not updated, the original value is preserved. See the instruction descriptions for the flags they affect.

You can execute an instruction conditionally, based on the condition flags set in another instruction, either:

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

Conditional execution is available by using conditional branches or by adding condition code suffixes to instructions. See Table 11-16 on page 83 for a list of the suffixes to add to instructions to make them conditional instructions. The condition code suffix enables the processor to test a condition based on the flags. If the condition test of a conditional instruction fails, the instruction:

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

Conditional instructions, except for conditional branches, must be inside an If-Then instruction block. See "IT" on page 127 for more information and restrictions when using the IT instruction. Depending on the vendor, the assembler might automatically insert an IT instruction if you have conditional instructions outside the IT block.

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

This section describes:

- "The condition flags"
- "Condition code suffixes" .

#### 11.11.7.1 The condition flags

The APSR contains the following condition flags:

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

For more information about the APSR see "Program Status Register" on page 42.

A carry occurs:

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

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

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

#### 11.11.7.2 Condition code suffixes

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

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

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

**Table 11-16. Condition code suffixes**

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

#### 11.11.7.3 Absolute value

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

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

#### 11.11.7.4 Compare and update value

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

```
CMP     R0, R1        ; Compare R0 and R1, setting flags
ITT     GT            ; IT instruction for the two GT conditions
CMPGT   R2, R3        ; If 'greater than', compare R2 and R3, setting flags
MOVGT   R4, R5        ; If still 'greater than', do R4 = R5
```

### 11.11.8 Instruction width selection

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

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

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

#### 11.11.8.1 Instruction width selection

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

```
        BCS.W  label      ; creates a 32-bit instruction even for a short branch

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

## 11.12 Memory access instructions

Table 11-17 shows the memory access instructions:

**Table 11-17. Memory access instructions**

| Mnemonic | Brief description | See |
|---|---|---|
| ADR | Load PC-relative address | "ADR" on page 86 |
| CLREX | Clear Exclusive | "CLREX" on page 99 |
| LDM{mode} | Load Multiple registers | "LDM and STM" on page 94 |
| LDR{type} | Load Register using immediate offset | "LDR and STR, immediate offset" on page 87 |
| LDR{type} | Load Register using register offset | "LDR and STR, register offset" on page 89 |
| LDR{type}T | Load Register with unprivileged access | "LDR and STR, unprivileged" on page 91 |
| LDR | Load Register using PC-relative address | "LDR, PC-relative" on page 92 |
| LDREX{type} | Load Register Exclusive | "LDREX and STREX" on page 97 |
| POP | Pop registers from stack | "PUSH and POP" on page 96 |
| PUSH | Push registers onto stack | "PUSH and POP" on page 96 |
| STM{mode} | Store Multiple registers | "LDM and STM" on page 94 |
| STR{type} | Store Register using immediate offset | "LDR and STR, immediate offset" on page 87 |
| STR{type} | Store Register using register offset | "LDR and STR, register offset" on page 89 |
| STR{type}T | Store Register with unprivileged access | "LDR and STR, unprivileged" on page 91 |
| STREX{type} | Store Register Exclusive | "LDREX and STREX" on page 97 |

### 11.12.1 ADR

Load PC-relative address.

#### 11.12.1.1 Syntax

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

where:

cond — is an optional condition code, see “Conditional execution” on page 81.

Rd — is the destination register.

label — is a PC-relative expression. See “PC-relative expressions” on page 81.

#### 11.12.1.2 Operation

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

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

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

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

You might have to use the .W suffix to get the maximum offset range or to generate addresses that are not word-aligned. See “Instruction width selection” on page 83.

#### 11.12.1.3 Restrictions

*Rd* must not be SP and must not be PC.

#### 11.12.1.4 Condition flags

This instruction does not change the flags.

#### 11.12.1.5 Examples

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

### 11.12.2 LDR and STR, immediate offset

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

#### 11.12.2.1 Syntax

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

where:

op is one of:

- LDR Load Register.
- STR Store Register.

type is one of:

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

cond is an optional condition code, see “Conditional execution” on page 81.

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.

#### 11.12.2.2 Operation

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

STR instructions store one or two register values to memory.

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

#### 11.12.2.3 Offset addressing

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

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

#### 11.12.2.4 Pre-indexed addressing

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

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

#### 11.12.2.5 Post-indexed addressing

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

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

The value to load or store can be a byte, halfword, word, or two words. Bytes and halfwords can either be signed or unsigned. See “Address alignment” on page 81.

Table 11-18 shows the ranges of offset for immediate, pre-indexed and post-indexed forms.

**Table 11-18. Offset ranges**

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

#### 11.12.2.6 Restrictions

For load instructions:

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

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

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

For store instructions:

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

#### 11.12.2.7 Condition flags

These instructions do not change the flags.

#### 11.12.2.8 Examples

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

### 11.12.3 LDR and STR, register offset

Load and Store with register offset.

#### 11.12.3.1 Syntax

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

where:

op — is one of:

- LDR — Load Register.
- STR — Store Register.

type — is one of:

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

cond — is an optional condition code, see “Conditional execution” on page 81.

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.

#### 11.12.3.2 Operation

LDR instructions load a register with a value from memory.

STR instructions store a register value into memory.

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

The value to load or store can be a byte, halfword, or word. For load instructions, bytes and halfwords can either be signed or unsigned. See “Address alignment” on page 81.

#### 11.12.3.3 Restrictions

In these instructions:

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

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

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

#### 11.12.3.4 Condition flags

These instructions do not change the flags.

#### 11.12.3.5 Examples

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

### 11.12.4 LDR and STR, unprivileged

Load and Store with unprivileged access.

#### 11.12.4.1 Syntax

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

where:

op is one of:

- LDR Load Register.
- STR Store Register.

type is one of:

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

cond is an optional condition code, see “Conditional execution” on page 81.

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*.

#### 11.12.4.2 Operation

These load and store instructions perform the same function as the memory access instructions with immediate offset, see “LDR and STR, immediate offset” on page 87. 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.

#### 11.12.4.3 Restrictions

In these instructions:

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

#### 11.12.4.4 Condition flags

These instructions do not change the flags.

#### 11.12.4.5 Examples

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

### 11.12.5 LDR, PC-relative

Load register from memory.

#### 11.12.5.1 Syntax

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

where:

type is one of:

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

cond is an optional condition code, see “Conditional execution” on page 81.

Rt is the register to load or store.

Rt2 is the second register to load or store.

label is a PC-relative expression. See “PC-relative expressions” on page 81.

#### 11.12.5.2 Operation

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

The value to load or store can be a byte, halfword, or word. For load instructions, bytes and halfwords can either be signed or unsigned. See “Address alignment” on page 81.

*label* must be within a limited range of the current instruction. Table 11-19 shows the possible offsets between *label* and the PC.

**Table 11-19. Offset ranges**

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

You might have to use the .W suffix to get the maximum offset range. See “Instruction width selection” on page 83.

#### 11.12.5.3 Restrictions

In these instructions:

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

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

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

#### 11.12.5.4 Condition flags

These instructions do not change the flags.

#### 11.12.5.5 Examples

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

### 11.12.6 LDM and STM

Load and Store Multiple registers.

#### 11.12.6.1 Syntax

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

where:

op is one of:

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

addr_mode is any one of the following:

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

cond is an optional condition code, see “Conditional execution” on page 81.

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

! is an optional writeback suffix.

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

reglist is a list of one or more registers to be loaded or stored, enclosed in braces. It can contain register ranges. It must be comma separated if it contains more than one register or register range, see “Examples” on page 95.

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

#### 11.12.6.2 Operation

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

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

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

For LDMDB, LDMEA, STMDB, and STMFD the memory addresses used for the accesses are at 4-byte intervals ranging from *Rn* to *Rn* - 4 * (*n*-1), where *n* is the number of registers in *reglist*. The accesses happen in order of decreasing register numbers, with the highest numbered register using the highest memory address and the lowest number register using the lowest memory address. If the writeback suffix is specified, the value of *Rn* - 4 * (*n*-1) is written back to *Rn*.

The PUSH and POP instructions can be expressed in this form. See “PUSH and POP” on page 96 for details.

#### 11.12.6.3 Restrictions

In these instructions:

- *Rn* must not be PC
- *reglist* must not contain SP
- in any STM instruction, *reglist* must not contain PC
- in any LDM instruction, *reglist* must not contain PC if it contains LR
- *reglist* must not contain *Rn* if you specify the writeback suffix.

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

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

#### 11.12.6.4 Condition flags

These instructions do not change the flags.

#### 11.12.6.5 Examples

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

#### 11.12.6.6 Incorrect examples

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

### 11.12.7 PUSH and POP

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

#### 11.12.7.1 Syntax

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

where:

cond is an optional condition code, see “Conditional execution” on page 81.

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.

#### 11.12.7.2 Operation

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

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

See “LDM and STM” on page 94 for more information.

#### 11.12.7.3 Restrictions

In these instructions:

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

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

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

#### 11.12.7.4 Condition flags

These instructions do not change the flags.

#### 11.12.7.5 Examples

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

### 11.12.8 LDREX and STREX

Load and Store Register Exclusive.

#### 11.12.8.1 Syntax

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

where:

cond — is an optional condition code, see “Conditional execution” on page 81.

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*.

#### 11.12.8.2 Operation

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

STREX, STREXB, and STREXH attempt to store a word, byte, and halfword respectively to a memory address. The address used in any Store-Exclusive instruction must be the same as the address in the most recently executed Load-exclusive instruction. The value stored by the Store-Exclusive instruction must also have the same data size as the value loaded by the preceding Load-exclusive instruction. This means software must always use a Load-exclusive instruction and a matching Store-Exclusive instruction to perform a synchronization operation, see “Synchronization primitives” on page 60

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.

#### 11.12.8.3 Restrictions

In these instructions:

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

#### 11.12.8.4 Condition flags

These instructions do not change the flags.

#### 11.12.8.5 Examples

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

### 11.12.9 CLREX

Clear Exclusive.

#### 11.12.9.1 Syntax

`CLREX{`*`cond`*`}`

where:

cond is an optional condition code, see “Conditional execution” on page 81.

#### 11.12.9.2 Operation

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

See “Synchronization primitives” on page 60 for more information.

#### 11.12.9.3 Condition flags

These instructions do not change the flags.

#### 11.12.9.4 Examples

```
CLREX
```

## 11.13 General data processing instructions

Table 11-20 shows the data processing instructions:

**Table 11-20. Data processing instructions**

| Mnemonic | Brief description | See |
|---|---|---|
| ADC | Add with Carry | “ADD, ADC, SUB, SBC, and RSB” on page 101 |
| ADD | Add | “ADD, ADC, SUB, SBC, and RSB” on page 101 |
| ADDW | Add | “ADD, ADC, SUB, SBC, and RSB” on page 101 |
| AND | Logical AND | “AND, ORR, EOR, BIC, and ORN” on page 103 |
| ASR | Arithmetic Shift Right | “ASR, LSL, LSR, ROR, and RRX” on page 104 |
| BIC | Bit Clear | “AND, ORR, EOR, BIC, and ORN” on page 103 |
| CLZ | Count leading zeros | “CLZ” on page 106 |
| CMN | Compare Negative | “CMP and CMN” on page 107 |
| CMP | Compare | “CMP and CMN” on page 107 |
| EOR | Exclusive OR | “AND, ORR, EOR, BIC, and ORN” on page 103 |
| LSL | Logical Shift Left | “ASR, LSL, LSR, ROR, and RRX” on page 104 |
| LSR | Logical Shift Right | “ASR, LSL, LSR, ROR, and RRX” on page 104 |
| MOV | Move | “MOV and MVN” on page 108 |
| MOVT | Move Top | “MOVT” on page 110 |
| MOVW | Move 16-bit constant | “MOV and MVN” on page 108 |
| MVN | Move NOT | “MOV and MVN” on page 108 |
| ORN | Logical OR NOT | “AND, ORR, EOR, BIC, and ORN” on page 103 |
| ORR | Logical OR | “AND, ORR, EOR, BIC, and ORN” on page 103 |
| RBIT | Reverse Bits | “REV, REV16, REVSH, and RBIT” on page 111 |
| REV | Reverse byte order in a word | “REV, REV16, REVSH, and RBIT” on page 111 |
| REV16 | Reverse byte order in each halfword | “REV, REV16, REVSH, and RBIT” on page 111 |
| REVSH | Reverse byte order in bottom halfword and sign extend | “REV, REV16, REVSH, and RBIT” on page 111 |
| ROR | Rotate Right | “ASR, LSL, LSR, ROR, and RRX” on page 104 |
| RRX | Rotate Right with Extend | “ASR, LSL, LSR, ROR, and RRX” on page 104 |
| RSB | Reverse Subtract | “ADD, ADC, SUB, SBC, and RSB” on page 101 |
| SBC | Subtract with Carry | “ADD, ADC, SUB, SBC, and RSB” on page 101 |
| SUB | Subtract | “ADD, ADC, SUB, SBC, and RSB” on page 101 |
| SUBW | Subtract | “ADD, ADC, SUB, SBC, and RSB” on page 101 |
| TEQ | Test Equivalence | “TST and TEQ” on page 112 |
| TST | Test | “TST and TEQ” on page 112 |

### 11.13.1 ADD, ADC, SUB, SBC, and RSB

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

#### 11.13.1.1 Syntax

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

where:

op is one of:

- ADD Add.
- ADC Add with Carry.
- SUB Subtract.
- SBC Subtract with Carry.
- RSB Reverse Subtract.

S is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see “Conditional execution” on page 81.

cond is an optional condition code, see “Conditional execution” on page 81.

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

Rn is the register holding the first operand.

Operand2 is a flexible second operand.

See “Flexible second operand” on page 77 for details of the options.

imm12 is any value in the range 0-4095.

#### 11.13.1.2 Operation

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

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

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

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

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

Use ADC and SBC to synthesize multiword arithmetic, see “Multiword arithmetic examples” on page 102.

See also “ADR” on page 86.

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

#### 11.13.1.3 Restrictions

In these instructions:

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

- *Rm* must not be PC and must not be SP
- if the instruction is conditional, it must be the last instruction in the IT block

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

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

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

#### 11.13.1.4 Condition flags

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

#### 11.13.1.5 Examples

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

#### 11.13.1.6 Multiword arithmetic examples

#### 11.13.1.7 64-bit addition

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

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

#### 11.13.1.8 96-bit subtraction

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

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

### 11.13.2 AND, ORR, EOR, BIC, and ORN

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

#### 11.13.2.1 Syntax

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

where:

op is one of:

- AND logical AND.
- ORR logical OR, or bit set.
- EOR logical Exclusive OR.
- BIC logical AND NOT, or bit clear.
- ORN logical OR NOT.

S is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see “Conditional execution” on page 81.

cond is an optional condition code, see See “Conditional execution” on page 81..

Rd is the destination register.

Rn is the register holding the first operand.

Operand2 is a flexible second operand. See “Flexible second operand” on page 77 for details of the options.

#### 11.13.2.2 Operation

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

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

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

#### 11.13.2.3 Restrictions

Do not use SP and do not use PC.

#### 11.13.2.4 Condition flags

If S is specified, these instructions:

- update the N and Z flags according to the result
- can update the C flag during the calculation of *Operand2*, see “Flexible second operand” on page 77
- do not affect the V flag.

#### 11.13.2.5 Examples

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

### 11.13.3 ASR, LSL, LSR, ROR, and RRX

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

#### 11.13.3.1 Syntax

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

where:

op is one of:

- ASR Arithmetic Shift Right.
- LSL Logical Shift Left.
- LSR Logical Shift Right.
- ROR Rotate Right.

S is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see “Conditional execution” on page 81.

Rd is the destination register.

Rm is the register holding the value to be shifted.

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

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

- ASR shift length from 1 to 32
- LSL shift length from 0 to 31
- LSR shift length from 1 to 32
- ROR shift length from 1 to 31.

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

#### 11.13.3.2 Operation

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

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

In all these instructions, the result is written to *Rd*, but the value in register *Rm* remains unchanged. For details on what result is generated by the different instructions, see “Shift Operations” on page 78.

#### 11.13.3.3 Restrictions

Do not use SP and do not use PC.

#### 11.13.3.4 Condition flags

If S is specified:

- these instructions update the N and Z flags according to the result
- the C flag is updated to the last bit shifted out, except when the shift length is 0, see “Shift Operations” on page 78.

#### 11.13.3.5 Examples

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

### 11.13.4 CLZ

Count Leading Zeros.

#### 11.13.4.1 Syntax

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

where:

cond is an optional condition code, see “Conditional execution” on page 81.

Rd is the destination register.

Rm is the operand register.

#### 11.13.4.2 Operation

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

#### 11.13.4.3 Restrictions

Do not use SP and do not use PC.

#### 11.13.4.4 Condition flags

This instruction does not change the flags.

#### 11.13.4.5 Examples

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

### 11.13.5 CMP and CMN

Compare and Compare Negative.

#### 11.13.5.1 Syntax

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

where:

cond — is an optional condition code, see “Conditional execution” on page 81.

Rn — is the register holding the first operand.

Operand2 — is a flexible second operand. See “Flexible second operand” on page 77 for details of the options.

#### 11.13.5.2 Operation

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

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

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

#### 11.13.5.3 Restrictions

In these instructions:

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

#### 11.13.5.4 Condition flags

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

#### 11.13.5.5 Examples

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

### 11.13.6 MOV and MVN

Move and Move NOT.

#### 11.13.6.1 Syntax

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

where:

S is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see “Conditional execution” on page 81.

cond is an optional condition code, see “Conditional execution” on page 81.

Rd is the destination register.

Operand2 is a flexible second operand. See “Flexible second operand” on page 77 for details of the options.

imm16 is any value in the range 0-65535.

#### 11.13.6.2 Operation

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

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

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

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

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

See “ASR, LSL, LSR, ROR, and RRX” on page 104.

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.

#### 11.13.6.3 Restrictions

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

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

When *Rd* is PC in a MOV instruction:

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

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

#### 11.13.6.4 Condition flags

If S is specified, these instructions:

- update the N and Z flags according to the result
- can update the C flag during the calculation of *Operand2*, see “Flexible second operand” on page 77
- do not affect the V flag.

#### 11.13.6.5 Example

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

### 11.13.7 MOVT

Move Top.

#### 11.13.7.1 Syntax

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

where:

cond is an optional condition code, see “Conditional execution” on page 81.

Rd is the destination register.

imm16 is a 16-bit immediate constant.

#### 11.13.7.2 Operation

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

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

#### 11.13.7.3 Restrictions

*Rd* must not be SP and must not be PC.

#### 11.13.7.4 Condition flags

This instruction does not change the flags.

#### 11.13.7.5 Examples

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

### 11.13.8 REV, REV16, REVSH, and RBIT

Reverse bytes and Reverse bits.

#### 11.13.8.1 Syntax

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

where:

op is any of:

- REV Reverse byte order in a word.
- REV16 Reverse byte order in each halfword independently.
- REVSH Reverse byte order in the bottom halfword, and sign extend to 32 bits.
- RBIT Reverse the bit order in a 32-bit word.

cond is an optional condition code, see “Conditional execution” on page 81.

Rd is the destination register.

Rn is the register holding the operand.

#### 11.13.8.2 Operation

Use these instructions to change endianness of data:

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

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

REVSH converts either:

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

#### 11.13.8.3 Restrictions

Do not use SP and do not use PC.

#### 11.13.8.4 Condition flags

These instructions do not change the flags.

#### 11.13.8.5 Examples

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

### 11.13.9 TST and TEQ

Test bits and Test Equivalence.

#### 11.13.9.1 Syntax

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

where:

cond — is an optional condition code, see “Conditional execution” on page 81.

Rn — is the register holding the first operand.

Operand2 — is a flexible second operand. See “Flexible second operand” on page 77 for details of the options.

#### 11.13.9.2 Operation

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

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

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

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

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

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

#### 11.13.9.3 Restrictions

Do not use SP and do not use PC.

#### 11.13.9.4 Condition flags

These instructions:

- update the N and Z flags according to the result
- can update the C flag during the calculation of *Operand2*, see “Flexible second operand” on page 77
- do not affect the V flag.

#### 11.13.9.5 Examples

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

## 11.14 Multiply and divide instructions

Table 11-21 shows the multiply and divide instructions:

**Table 11-21. Multiply and divide instructions**

| Mnemonic | Brief description | See |
|---|---|---|
| MLA | Multiply with Accumulate, 32-bit result | "MUL, MLA, and MLS" on page 114 |
| MLS | Multiply and Subtract, 32-bit result | "MUL, MLA, and MLS" on page 114 |
| MUL | Multiply, 32-bit result | "MUL, MLA, and MLS" on page 114 |
| SDIV | Signed Divide | "SDIV and UDIV" on page 116 |
| SMLAL | Signed Multiply with Accumulate (32x32+64), 64-bit result | "UMULL, UMLAL, SMULL, and SMLAL" on page 115 |
| SMULL | Signed Multiply (32x32), 64-bit result | "UMULL, UMLAL, SMULL, and SMLAL" on page 115 |
| UDIV | Unsigned Divide | "SDIV and UDIV" on page 116 |
| UMLAL | Unsigned Multiply with Accumulate (32x32+64), 64-bit result | "UMULL, UMLAL, SMULL, and SMLAL" on page 115 |
| UMULL | Unsigned Multiply (32x32), 64-bit result | "UMULL, UMLAL, SMULL, and SMLAL" on page 115 |

### 11.14.1 MUL, MLA, and MLS

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

#### 11.14.1.1 Syntax

```
MUL{S}{cond} {Rd,} Rn, Rm ; Multiply
MLA{cond} Rd, Rn, Rm, Ra  ; Multiply with accumulate
MLS{cond} Rd, Rn, Rm, Ra  ; Multiply with subtract
```

where:

cond is an optional condition code, see “Conditional execution” on page 81.

S is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see “Conditional execution” on page 81.

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.

#### 11.14.1.2 Operation

The MUL instruction multiplies the values from *Rn* and *Rm*, and places the least significant 32 bits of the result in *Rd*.

The MLA instruction multiplies the values from *Rn* and *Rm*, adds the value from *Ra*, and places the least significant 32 bits of the result in *Rd*.

The MLS instruction multiplies the values from *Rn* and *Rm*, subtracts the product from the value from *Ra*, and places the least significant 32 bits of the result in *Rd*.

The results of these instructions do not depend on whether the operands are signed or unsigned.

#### 11.14.1.3 Restrictions

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

If you use the S suffix with the MUL instruction:

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

#### 11.14.1.4 Condition flags

If S is specified, the MUL instruction:

- updates the N and Z flags according to the result
- does not affect the C and V flags.

#### 11.14.1.5 Examples

```
MUL     R10, R2, R5     ; Multiply, R10 = R2 x R5
MLA     R10, R2, R1, R5 ; Multiply with accumulate, R10 = (R2 x R1) + R5
MULS    R0, R2, R2      ; Multiply with flag update, R0 = R2 x R2
MULLT   R2, R3, R2      ; Conditionally multiply, R2 = R3 x R2
MLS     R4, R5, R6, R7  ; Multiply with subtract, R4 = R7 - (R5 x R6)
```

### 11.14.2 UMULL, UMLAL, SMULL, and SMLAL

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

#### 11.14.2.1 Syntax

```
op{cond} RdLo, RdHi, Rn, Rm
```

where:

op is one of:

- UMULL Unsigned Long Multiply.
- UMLAL Unsigned Long Multiply, with Accumulate.
- SMULL Signed Long Multiply.
- SMLAL Signed Long Multiply, with Accumulate.

cond is an optional condition code, see “Conditional execution” on page 81.

*RdHi*, *RdLo* are the destination registers.

For UMLAL and SMLAL they also hold the accumulating value.

Rn, Rm are registers holding the operands.

#### 11.14.2.2 Operation

The UMULL instruction interprets the values from *Rn* and *Rm* as unsigned integers. It multiplies these integers and places the least significant 32 bits of the result in *RdLo*, and the most significant 32 bits of the result in *RdHi*.

The UMLAL instruction interprets the values from *Rn* and *Rm* as unsigned integers. It multiplies these integers, adds the 64-bit result to the 64-bit unsigned integer contained in *RdHi* and *RdLo*, and writes the result back to *RdHi* and *RdLo*.

The SMULL instruction interprets the values from *Rn* and *Rm* as two’s complement signed integers. It multiplies these integers and places the least significant 32 bits of the result in *RdLo*, and the most significant 32 bits of the result in *RdHi*.

The SMLAL instruction interprets the values from *Rn* and *Rm* as two’s complement signed integers. It multiplies these integers, adds the 64-bit result to the 64-bit signed integer contained in *RdHi* and *RdLo*, and writes the result back to *RdHi* and *RdLo*.

#### 11.14.2.3 Restrictions

In these instructions:

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

#### 11.14.2.4 Condition flags

These instructions do not affect the condition code flags.

#### 11.14.2.5 Examples

```
UMULL       R0, R4, R5, R6   ; Unsigned (R4,R0) = R5 x R6
SMLAL       R4, R5, R3, R8   ; Signed (R5,R4) = (R5,R4) + R3 x R8
```

### 11.14.3 SDIV and UDIV

Signed Divide and Unsigned Divide.

#### 11.14.3.1 Syntax

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

where:

cond is an optional condition code, see “Conditional execution” on page 81.

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.

#### 11.14.3.2 Operation

SDIV performs a signed integer division of the value in *Rn* by the value in *Rm*.

UDIV performs an unsigned integer division of the value in *Rn* by the value in *Rm*.

For both instructions, if the value in *Rn* is not divisible by the value in *Rm*, the result is rounded towards zero.

#### 11.14.3.3 Restrictions

Do not use SP and do not use PC.

#### 11.14.3.4 Condition flags

These instructions do not change the flags.

#### 11.14.3.5 Examples

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

## 11.15 Saturating instructions

This section describes the saturating instructions, SSAT and USAT.

### 11.15.1 SSAT and USAT

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

#### 11.15.1.1 Syntax

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

where:

op is one of:

- SSAT Saturates a signed value to a signed range.
- USAT Saturates a signed value to an unsigned range.

cond is an optional condition code, see “Conditional execution” on page 81.

Rd is the destination register.

n specifies the bit position to saturate to:

- *n* ranges from 1 to 32 for SSAT
- *n* ranges from 0 to 31 for USAT.

Rm is the register containing the value to saturate.

*shift* #s is an optional shift applied to *Rm* before saturating. It must be one of the following:

- ASR #*s* where *s* is in the range 1 to 31
- LSL #*s* where *s* is in the range 0 to 31.

#### 11.15.1.2 Operation

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

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

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

For signed *n*-bit saturation using SSAT, this means that:

- if the value to be saturated is less than $-2^{n-1}$, the result returned is $-2^{n-1}$
- if the value to be saturated is greater than $2^{n-1}-1$, the result returned is $2^{n-1}-1$
- otherwise, the result returned is the same as the value to be saturated.

For unsigned *n*-bit saturation using USAT, this means that:

- if the value to be saturated is less than 0, the result returned is 0
- if the value to be saturated is greater than $2^{n}-1$, the result returned is $2^{n}-1$
- otherwise, the result returned is the same as the value to be saturated.

If the returned result is different from the value to be saturated, it is called *saturation*. If saturation occurs, the instruction sets the Q flag to 1 in the APSR. Otherwise, it leaves the Q flag unchanged. To clear the Q flag to 0, you must use the MSR instruction, see “MSR” on page 137.

To read the state of the Q flag, use the MRS instruction, see “MRS” on page 136.

#### 11.15.1.3 Restrictions

Do not use SP and do not use PC.

#### 11.15.1.4 Condition flags

These instructions do not affect the condition code flags.

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

#### 11.15.1.5 Examples

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

## 11.16 Bitfield instructions

Table 11-22 shows the instructions that operate on adjacent sets of bits in registers or bitfields:

**Table 11-22. Packing and unpacking instructions**

| Mnemonic | Brief description | See |
|---|---|---|
| BFC | Bit Field Clear | "BFC and BFI" on page 120 |
| BFI | Bit Field Insert | "BFC and BFI" on page 120 |
| SBFX | Signed Bit Field Extract | "SBFX and UBFX" on page 121 |
| SXTB | Sign extend a byte | "SXT and UXT" on page 122 |
| SXTH | Sign extend a halfword | "SXT and UXT" on page 122 |
| UBFX | Unsigned Bit Field Extract | "SBFX and UBFX" on page 121 |
| UXTB | Zero extend a byte | "SXT and UXT" on page 122 |
| UXTH | Zero extend a halfword | "SXT and UXT" on page 122 |

### 11.16.1 BFC and BFI

Bit Field Clear and Bit Field Insert.

#### 11.16.1.1 Syntax

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

where:

cond — is an optional condition code, see “Conditional execution” on page 81.

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*.

#### 11.16.1.2 Operation

BFC clears a bitfield in a register. It clears *width* bits in *Rd*, starting at the low bit position *lsb*. Other bits in *Rd* are unchanged.

BFI copies a bitfield into one register from another register. It replaces *width* bits in *Rd* starting at the low bit position *lsb*, with *width* bits from *Rn* starting at bit[0]. Other bits in *Rd* are unchanged.

#### 11.16.1.3 Restrictions

Do not use SP and do not use PC.

#### 11.16.1.4 Condition flags

These instructions do not affect the flags.

#### 11.16.1.5 Examples

```
BFC   R4, #8, #12       ; Clear bit 8 to bit 19 (12 bits) of R4 to 0
BFI   R9, R2, #8, #12   ; Replace bit 8 to bit 19 (12 bits) of R9 with
                        ; bit 0 to bit 11 from R2
```

### 11.16.2 SBFX and UBFX

Signed Bit Field Extract and Unsigned Bit Field Extract.

#### 11.16.2.1 Syntax

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

where:

cond — is an optional condition code, see “Conditional execution” on page 81.

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*.

#### 11.16.2.2 Operation

SBFX extracts a bitfield from one register, sign extends it to 32 bits, and writes the result to the destination register.

UBFX extracts a bitfield from one register, zero extends it to 32 bits, and writes the result to the destination register.

#### 11.16.2.3 Restrictions

Do not use SP and do not use PC.

#### 11.16.2.4 Condition flags

These instructions do not affect the flags.

#### 11.16.2.5 Examples

```
SBFX  R0, R1, #20, #4  ; Extract bit 20 to bit 23 (4 bits) from R1 and sign
                       ; extend to 32 bits and then write the result to R0.
UBFX  R8, R11, #9, #10 ; Extract bit 9 to bit 18 (10 bits) from R11 and zero
                       ; extend to 32 bits and then write the result to R8
```

### 11.16.3 SXT and UXT

Sign extend and Zero extend.

#### 11.16.3.1 Syntax

```
SXTextend{cond} {Rd,} Rm {, ROR #n}
UXTextend{cond} {Rd}, Rm {, ROR #n}
```

where:

extend is one of:

- B Extends an 8-bit value to a 32-bit value.
- H Extends a 16-bit value to a 32-bit value.

cond is an optional condition code, see “Conditional execution” on page 81.

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.

#### 11.16.3.2 Operation

These instructions do the following:

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

  SXTB extracts bits[7:0] and sign extends to 32 bits.

  UXTB extracts bits[7:0] and zero extends to 32 bits.

  SXTH extracts bits[15:0] and sign extends to 32 bits.

  UXTH extracts bits[15:0] and zero extends to 32 bits.

#### 11.16.3.3 Restrictions

Do not use SP and do not use PC.

#### 11.16.3.4 Condition flags

These instructions do not affect the flags.

#### 11.16.3.5 Examples

```
SXTH  R4, R6, ROR #16  ; Rotate R6 right by 16 bits, then obtain the lower
                       ; halfword of the result and then sign extend to
                       ; 32 bits and write the result to R4.
UXTB  R3, R10          ; Extract lowest byte of the value in R10 and zero
                       ; extend it, and write the result to R3
```

## 11.17 Branch and control instructions

Table 11-23 shows the branch and control instructions:

**Table 11-23. Branch and control instructions**

| Mnemonic | Brief description | See |
|---|---|---|
| B | Branch | "B, BL, BX, and BLX" on page 124 |
| BL | Branch with Link | "B, BL, BX, and BLX" on page 124 |
| BLX | Branch indirect with Link | "B, BL, BX, and BLX" on page 124 |
| BX | Branch indirect | "B, BL, BX, and BLX" on page 124 |
| CBNZ | Compare and Branch if Non Zero | "CBZ and CBNZ" on page 126 |
| CBZ | Compare and Branch if Non Zero | "CBZ and CBNZ" on page 126 |
| IT | If-Then | "IT" on page 127 |
| TBB | Table Branch Byte | "TBB and TBH" on page 129 |
| TBH | Table Branch Halfword | "TBB and TBH" on page 129 |

### 11.17.1 B, BL, BX, and BLX

Branch instructions.

#### 11.17.1.1 Syntax

```
B{cond} label
BL{cond} label
BX{cond} Rm
BLX{cond} Rm
```

where:

B is branch (immediate).

BL is branch with link (immediate).

BX is branch indirect (register).

BLX is branch indirect with link (register).

cond is an optional condition code, see “Conditional execution” on page 81.

label is a PC-relative expression. See “PC-relative expressions” on page 81.

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.

#### 11.17.1.2 Operation

All these instructions cause a branch to *label*, or to the address indicated in *Rm*. In addition:

- The BL and BLX instructions write the address of the next instruction to LR (the link register, R14).
- The BX and BLX instructions cause a UsageFault exception if bit[0] of *Rm* is 0.

B*cond* label is the only conditional instruction that can be either inside or outside an IT block. All other branch instructions must be conditional inside an IT block, and must be unconditional outside the IT block, see “IT” on page 127.

Table 11-24 shows the ranges for the various branch instructions.

**Table 11-24. Branch ranges**

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

You might have to use the .W suffix to get the maximum branch range. See “Instruction width selection” on page 83.

#### 11.17.1.3 Restrictions

The restrictions are:

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

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

#### 11.17.1.4 Condition flags

These instructions do not change the flags.

#### 11.17.1.5 Examples

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

### 11.17.2 CBZ and CBNZ

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

#### 11.17.2.1 Syntax

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

where:

Rn is the register holding the operand.

label is the branch destination.

#### 11.17.2.2 Operation

Use the CBZ or CBNZ instructions to avoid changing the condition code flags and to reduce the number of instructions.

CBZ Rn, label does not change condition flags but is otherwise equivalent to:

```
CMP     Rn, #0
BEQ     label
```

CBNZ Rn, label does not change condition flags but is otherwise equivalent to:

```
CMP     Rn, #0
BNE     label
```

#### 11.17.2.3 Restrictions

The restrictions are:

- *Rn* must be in the range of R0 to R7
- the branch destination must be within 4 to 130 bytes after the instruction
- these instructions must not be used inside an IT block.

#### 11.17.2.4 Condition flags

These instructions do not change the flags.

#### 11.17.2.5 Examples

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

### 11.17.3 IT

If-Then condition instruction.

#### 11.17.3.1 Syntax

```
IT{x{y{z}}} cond
```

where:

x specifies the condition switch for the second instruction in the IT block.

y specifies the condition switch for the third instruction in the IT block.

z specifies the condition switch for the fourth instruction in the IT block.

cond specifies the condition for the first instruction in the IT block.

The condition switch for the second, third and fourth instruction in the IT block can be either:

T Then. Applies the condition *cond* to the instruction.

E Else. Applies the inverse condition of *cond* to the instruction.

It is possible to use AL (the *always* condition) for *cond* in an IT instruction. If this is done, all of the instructions in the IT block must be unconditional, and each of *x*, *y*, and *z* must be T or omitted but not E.

#### 11.17.3.2 Operation

The IT instruction makes up to four following instructions conditional. The conditions can be all the same, or some of them can be the logical inverse of the others. The conditional instructions following the IT instruction form the *IT block*.

The instructions in the IT block, including any branches, must specify the condition in the {*cond*} part of their syntax.

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

A BKPT instruction in an IT block is always executed, even if its condition fails.

Exceptions can be taken between an IT instruction and the corresponding IT block, or within an IT block. Such an exception results in entry to the appropriate exception handler, with suitable return information in LR and stacked PSR.

Instructions designed for use for exception returns can be used as normal to return from the exception, and execution of the IT block resumes correctly. This is the only way that a PC-modifying instruction is permitted to branch to an instruction in an IT block.

#### 11.17.3.3 Restrictions

The following instructions are not permitted in an IT block:

- IT
- CBZ and CBNZ
- CPSID and CPSIE.

Other restrictions when using an IT block are:

- a branch or any instruction that modifies the PC must either be outside an IT block or must be the last instruction inside the IT block. These are:
  - ADD PC, PC, Rm
  - MOV PC, Rm
  - B, BL, BX, BLX
  - any LDM, LDR, or POP instruction that writes to the PC
  - TBB and TBH

- do not branch to any instruction inside an IT block, except when returning from an exception handler
- all conditional instructions except B*cond* must be inside an IT block. B*cond* can be either outside or inside an IT block but has a larger branch range if it is inside one
- each instruction inside the IT block must specify a condition code suffix that is either the same or logical inverse as for the other instructions in the block.

Your assembler might place extra restrictions on the use of IT blocks, such as prohibiting the use of assembler directives within them.

#### 11.17.3.4 Condition flags

This instruction does not change the flags.

#### 11.17.3.5 Example

```
ITTE   NE           ; Next 3 instructions are conditional
ANDNE  R0, R0, R1   ; ANDNE does not update condition flags
ADDSNE R2, R2, #1   ; ADDSNE updates condition flags
MOVEQ  R2, R3       ; Conditional move


CMP    R0, #9       ; Convert R0 hex value (0 to 15) into ASCII
                    ; ('0'-'9', 'A'-'F')
ITE    GT           ; Next 2 instructions are conditional
ADDGT  R1, R0, #55  ; Convert 0xA -> 'A'
ADDLE  R1, R0, #48  ; Convert 0x0 -> '0'


IT     GT           ; IT block with only one conditional instruction
ADDGT  R1, R1, #1   ; Increment R1 conditionally


ITTEE  EQ           ; Next 4 instructions are conditional
MOVEQ  R0, R1       ; Conditional move
ADDEQ  R2, R2, #10  ; Conditional add
ANDNE  R3, R3, #1   ; Conditional AND
BNE.W  dloop        ; Branch instruction can only be used in the last
                    ; instruction of an IT block


IT     NE           ; Next instruction is conditional
ADD    R0, R0, R1   ; Syntax error: no condition code used in IT block
```

### 11.17.4 TBB and TBH

Table Branch Byte and Table Branch Halfword.

#### 11.17.4.1 Syntax

```
TBB [Rn, Rm]
TBH [Rn, Rm, LSL #1]
```

where:

Rn is the register containing the address of the table of branch lengths. If *Rn* is PC, then the address of the table is the address of the byte immediately following the TBB or TBH instruction.

Rm is the index register. This contains an index into the table. For halfword tables, LSL #1 doubles the value in *Rm* to form the right offset into the table.

#### 11.17.4.2 Operation

These instructions cause a PC-relative forward branch using a table of single byte offsets for TBB, or halfword offsets for TBH. *Rn* provides a pointer to the table, and *Rm* supplies an index into the table. For TBB the branch offset is twice the unsigned value of the byte returned from the table. and for TBH the branch offset is twice the unsigned value of the halfword returned from the table. The branch occurs to the address at that offset from the address of the byte immediately after the TBB or TBH instruction.

#### 11.17.4.3 Restrictions

The restrictions are:

- *Rn* must not be SP
- *Rm* must not be SP and must not be PC
- when any of these instructions is used inside an IT block, it must be the last instruction of the IT block.

#### 11.17.4.4 Condition flags

These instructions do not change the flags.

#### 11.17.4.5 Examples

```
       ADR.W  R0, BranchTable_Byte
       TBB   [R0, R1]          ; R1 is the index, R0 is the base address of the
                               ; branch table
   Case1
   ; an instruction sequence follows
   Case2
   ; an instruction sequence follows
   Case3
   ; an instruction sequence follows
   BranchTable_Byte
       DCB    0                    ; Case1 offset calculation
       DCB    ((Case2-Case1)/2)    ; Case2 offset calculation
       DCB    ((Case3-Case1)/2)    ; Case3 offset calculation
       TBH    [PC, R1, LSL #1]     ; R1 is the index, PC is used as base of the
                                   ; branch table
   BranchTable_H
       DCI    ((CaseA - BranchTable_H)/2)  ; CaseA offset calculation
       DCI    ((CaseB - BranchTable_H)/2)  ; CaseB offset calculation
       DCI    ((CaseC - BranchTable_H)/2)  ; CaseC offset calculation

   CaseA
   ; an instruction sequence follows
   CaseB
   ; an instruction sequence follows
   CaseC
   ; an instruction sequence follows
```

## 11.18 Miscellaneous instructions

Table 11-25 shows the remaining Cortex-M3 instructions:

**Table 11-25. Miscellaneous instructions**

| Mnemonic | Brief description | See |
|---|---|---|
| BKPT | Breakpoint | "BKPT" on page 131 |
| CPSID | Change Processor State, Disable Interrupts | "CPS" on page 132 |
| CPSIE | Change Processor State, Enable Interrupts | "CPS" on page 132 |
| DMB | Data Memory Barrier | "DMB" on page 133 |
| DSB | Data Synchronization Barrier | "DSB" on page 134 |
| ISB | Instruction Synchronization Barrier | "ISB" on page 135 |
| MRS | Move from special register to register | "MRS" on page 136 |
| MSR | Move from register to special register | "MSR" on page 137 |
| NOP | No Operation | "NOP" on page 138 |
| SEV | Send Event | "SEV" on page 139 |
| SVC | Supervisor Call | "SVC" on page 140 |
| WFE | Wait For Event | "WFE" on page 141 |
| WFI | Wait For Interrupt | "WFI" on page 142 |

### 11.18.1 BKPT

Breakpoint.

#### 11.18.1.1 Syntax

```
BKPT #imm
```

where:

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

#### 11.18.1.2 Operation

The BKPT instruction causes the processor to enter Debug state. Debug tools can use this to investigate system state when the instruction at a particular address is reached.

*imm* is ignored by the processor. If required, a debugger can use it to store additional information about the breakpoint.

The BKPT instruction can be placed inside an IT block, but it executes unconditionally, unaffected by the condition specified by the IT instruction.

#### 11.18.1.3 Condition flags

This instruction does not change the flags.

#### 11.18.1.4 Examples

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

### 11.18.2 CPS

Change Processor State.

#### 11.18.2.1 Syntax

```
CPSeffect iflags
```

where:

effect is one of:

- IE Clears the special purpose register.
- ID Sets the special purpose register.

iflags is a sequence of one or more flags:

- i Set or clear PRIMASK.
- f Set or clear FAULTMASK.

#### 11.18.2.2 Operation

CPS changes the PRIMASK and FAULTMASK special register values. See “Exception mask registers” on page 47 for more information about these registers.

#### 11.18.2.3 Restrictions

The restrictions are:

- use CPS only from privileged software, it has no effect if used in unprivileged software
- CPS cannot be conditional and so must not be used inside an IT block.

#### 11.18.2.4 Condition flags

This instruction does not change the condition flags.

#### 11.18.2.5 Examples

```
CPSID i  ; Disable interrupts and configurable fault handlers (set PRIMASK)
CPSID f  ; Disable interrupts and all fault handlers (set FAULTMASK)
CPSIE i  ; Enable interrupts and configurable fault handlers (clear PRIMASK)
CPSIE f  ; Enable interrupts and fault handlers (clear FAULTMASK)
```

### 11.18.3 DMB

Data Memory Barrier.

#### 11.18.3.1 Syntax

```
DMB{cond}
```

where:

cond is an optional condition code, see “Conditional execution” on page 81.

#### 11.18.3.2 Operation

DMB acts as a data memory barrier. It ensures that all explicit memory accesses that appear, in program order, before the DMB instruction are completed before any explicit memory accesses that appear, in program order, after the DMB instruction. DMB does not affect the ordering or execution of instructions that do not access memory.

#### 11.18.3.3 Condition flags

This instruction does not change the flags.

#### 11.18.3.4 Examples

```
DMB  ; Data Memory Barrier
```

### 11.18.4 DSB

Data Synchronization Barrier.

#### 11.18.4.1 Syntax

```
DSB{cond}
```

where:

cond is an optional condition code, see “Conditional execution” on page 81.

#### 11.18.4.2 Operation

DSB acts as a special data synchronization memory barrier. Instructions that come after the DSB, in program order, do not execute until the DSB instruction completes. The DSB instruction completes when all explicit memory accesses before it complete.

#### 11.18.4.3 Condition flags

This instruction does not change the flags.

#### 11.18.4.4 Examples

```
DSB ; Data Synchronisation Barrier
```

### 11.18.5 ISB

Instruction Synchronization Barrier.

#### 11.18.5.1 Syntax

```
ISB{cond}
```

where:

cond is an optional condition code, see “Conditional execution” on page 81.

#### 11.18.5.2 Operation

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

#### 11.18.5.3 Condition flags

This instruction does not change the flags.

#### 11.18.5.4 Examples

```
ISB  ; Instruction Synchronisation Barrier
```

### 11.18.6 MRS

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

#### 11.18.6.1 Syntax

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

where:

cond is an optional condition code, see “Conditional execution” on page 81.

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.

#### 11.18.6.2 Operation

Use MRS in combination with MSR as part of a read-modify-write sequence for updating a PSR, for example to clear the Q flag.

In process swap code, the programmers model state of the process being swapped out must be saved, including relevant PSR contents. Similarly, the state of the process being swapped in must also be restored. These operations use MRS in the state-saving instruction sequence and MSR in the state-restoring instruction sequence.

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

See “MSR” on page 137.

#### 11.18.6.3 Restrictions

*Rd* must not be SP and must not be PC.

#### 11.18.6.4 Condition flags

This instruction does not change the flags.

#### 11.18.6.5 Examples

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

### 11.18.7 MSR

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

#### 11.18.7.1 Syntax

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

where:

cond is an optional condition code, see “Conditional execution” on page 81.

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.

#### 11.18.7.2 Operation

The register access operation in MSR depends on the privilege level. Unprivileged software can only access the APSR, see “Application Program Status Register” on page 44. Privileged software can access all special registers.

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

When you write to BASEPRI_MAX, the instruction writes to BASEPRI only if either:

- *Rn* is non-zero and the current BASEPRI value is 0
- *Rn* is non-zero and less than the current BASEPRI value.

See “MRS” on page 136.

#### 11.18.7.3 Restrictions

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

#### 11.18.7.4 Condition flags

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

#### 11.18.7.5 Examples

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

### 11.18.8 NOP

No Operation.

#### 11.18.8.1 Syntax

```
NOP{cond}
```

where:

cond is an optional condition code, see “Conditional execution” on page 81.

#### 11.18.8.2 Operation

NOP does nothing. NOP is not necessarily a time-consuming NOP. The processor might remove it from the pipeline before it reaches the execution stage.

Use NOP for padding, for example to place the following instruction on a 64-bit boundary.

#### 11.18.8.3 Condition flags

This instruction does not change the flags.

#### 11.18.8.4 Examples

```
NOP  ; No operation
```

### 11.18.9 SEV

Send Event.

#### 11.18.9.1 Syntax

SEV{*cond*}

where:

cond is an optional condition code, see “Conditional execution” on page 81.

#### 11.18.9.2 Operation

SEV is a hint instruction that causes an event to be signaled to all processors within a multiprocessor system. It also sets the local event register to 1, see “Power management” on page 70.

#### 11.18.9.3 Condition flags

This instruction does not change the flags.

#### 11.18.9.4 Examples

```
SEV ; Send Event
```

### 11.18.10SVC

Supervisor Call.

#### 11.18.10.1 Syntax

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

where:

cond is an optional condition code, see “Conditional execution” on page 81.

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

#### 11.18.10.2 Operation

The SVC instruction causes the SVC exception.

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

#### 11.18.10.3 Condition flags

This instruction does not change the flags.

#### 11.18.10.4 Examples

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

### 11.18.11WFE

Wait For Event.

#### 11.18.11.1 Syntax

`WFE{`*`cond`*`}`

where:

cond is an optional condition code, see “Conditional execution” on page 81.

#### 11.18.11.2 Operation

WFE is a hint instruction.

If the event register is 0, WFE suspends execution until one of the following events occurs:

- an exception, unless masked by the exception mask registers or the current priority level
- an exception enters the Pending state, if SEVONPEND in the System Control Register is set
- a Debug Entry request, if Debug is enabled
- an event signaled by a peripheral or another processor in a multiprocessor system using the SEV instruction.

If the event register is 1, WFE clears it to 0 and returns immediately.

For more information see “Power management” on page 70.

#### 11.18.11.3 Condition flags

This instruction does not change the flags.

#### 11.18.11.4 Examples

```
WFE  ; Wait for event
```

### 11.18.12WFI

Wait for Interrupt.

#### 11.18.12.1 Syntax

```
WFI{cond}
```

where:

cond is an optional condition code, see “Conditional execution” on page 81.

#### 11.18.12.2 Operation

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

- an exception
- a Debug Entry request, regardless of whether Debug is enabled.

#### 11.18.12.3 Condition flags

This instruction does not change the flags.

#### 11.18.12.4 Examples

```
WFI ; Wait for interrupt
```

## 11.19 About the Cortex-M3 peripherals

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

**Table 11-26. Core peripheral register regions**

| Address | Core peripheral | Description |
|---|---|---|
| 0xE000E008-<br>0xE000E00F | System control block | Table 11-30 on page 157 |
| 0xE000E010-<br>0xE000E01F | System timer | Table 11-33 on page 186 |
| 0xE000E100-<br>0xE000E4EF | Nested Vectored Interrupt Controller | Table 11-27 on page 144 |
| 0xE000ED00-<br>0xE000ED3F | System control block | Table 11-30 on page 157 |
| 0xE000ED90-<br>0xE000ED93 | MPU Type Register | Reads as zero, indicating no MPU is implemented (1) |
| 0xE000EF00-<br>0xE000EF03 | Nested Vectored Interrupt Controller | Table 11-27 on page 144 |

1. Software can read the MPU Type Register at 0xE000ED90 to test for the presence of a *memory protection unit* (MPU).

In register descriptions:

- the register *type* is described as follows:
  - RW Read and write.
  - RO Read-only.
  - WO Write-only.
- the *required privilege* gives the privilege level required to access the register, as follows:
  - Privileged Only privileged software can access the register.
  - Unprivileged Both unprivileged and privileged software can access the register.

## 11.20 Nested Vectored Interrupt Controller

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

- 1 to 33 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 and pulse detection of interrupt signals.
- Dynamic reprioritization of interrupts.
- Grouping of priority values into group priority and subpriority fields.
- Interrupt tail-chaining.

The processor automatically stacks its state on exception entry and unstacks this state on exception exit, with no instruction overhead. This provides low latency exception handling. The hardware implementation of the NVIC registers is:

**Table 11-27. NVIC register summary**

| Address | Name | Type | Required privilege | Reset value | Description |
|---|---|---|---|---|---|
| 0xE000E100 | ISER0 | RW | Privileged | 0x00000000 | “Interrupt Set-enable Registers” on page 146 |
| 0xE000E180 | ICER0 | RW | Privileged | 0x00000000 | “Interrupt Clear-enable Registers” on page 147 |
| 0xE000E200 | ISPR0 | RW | Privileged | 0x00000000 | “Interrupt Set-pending Registers” on page 148 |
| 0xE000E280 | ICPR0 | RW | Privileged | 0x00000000 | “Interrupt Clear-pending Registers” on page 149 |
| 0xE000E300 | IABR0 | RO | Privileged | 0x00000000 | “Interrupt Active Bit Registers” on page 150 |
| 0xE000E400-0xE000E41C | IPR0-IPR8 | RW | Privileged | 0x00000000 | “Interrupt Priority Registers” on page 151 |
| 0xE000EF00 | STIR | WO | Configurable (1) | 0x00000000 | “Software Trigger Interrupt Register” on page 154 |

1. See the register description for more information.

### 11.20.1 The CMSIS mapping of the Cortex-M3 NVIC registers

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

- the Set-enable, Clear-enable, Set-pending, Clear-pending and Active Bit registers map to arrays of 32-bit integers, so that:
  - the array ISER[0] corresponds to the registers ISER0
  - the array ICER[0] corresponds to the registers ICER0
  - the array ISPR[0] corresponds to the registers ISPR0
  - the array ICPR[0] corresponds to the registers ICPR0
  - the array IABR[0] corresponds to the registers IABR0
- the 4-bit fields of the Interrupt Priority Registers map to an array of 4-bit integers, so that the array IP[0] to IP[32] corresponds to the registers IPR0-IPR8, and the array entry IP[*n*] holds the interrupt priority for interrupt *n*.

The CMSIS provides thread-safe code that gives atomic access to the Interrupt Priority Registers. For more information see the description of the NVIC_SetPriority function in “NVIC programming hints” on page 156. Table 11-28 shows how the interrupts, or IRQ numbers, map onto the interrupt registers and corresponding CMSIS variables that have one bit per interrupt.

**Table 11-28. Mapping of interrupts to the interrupt variables**

| Interrupts | CMSIS array elements (1) | | | | |
|---|---|---|---|---|---|
| | **Set-enable** | **Clear-enable** | **Set-pending** | **Clear-pending** | **Active Bit** |
| 0-32 | ISER[0] | ICER[0] | ISPR[0] | ICPR[0] | IABR[0] |

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

### 11.20.2 Interrupt Set-enable Registers

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

- the register summary in Table 11-27 on page 144 for the register attributes
- Table 11-28 on page 145 for which interrupts are controlled by each register.

The bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| SETENA bits | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| SETENA bits | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| SETENA bits | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SETENA bits | | | | | | | |

- **SETENA**

Interrupt set-enable bits.

Write:

0 = no effect

1 = enable interrupt.

Read:

0 = interrupt disabled

1 = interrupt enabled.

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

### 11.20.3 Interrupt Clear-enable Registers

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

- the register summary in Table 11-27 on page 144 for the register attributes
- Table 11-28 on page 145 for which interrupts are controlled by each register

The bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| CLRENA | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| CLRENA | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CLRENA | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CLRENA | | | | | | | |

- **CLRENA**

Interrupt clear-enable bits.

Write:

0 = no effect

1 = disable interrupt.

Read:

0 = interrupt disabled

1 = interrupt enabled.

### 11.20.4 Interrupt Set-pending Registers

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

- the register summary in Table 11-27 on page 144 for the register attributes
- Table 11-28 on page 145 for which interrupts are controlled by each register.

The bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| SETPEND | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| SETPEND | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| SETPEND | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SETPEND | | | | | | | |

- **SETPEND**

Interrupt set-pending bits.

Write:

0 = no effect.

1 = changes interrupt state to pending.

Read:

0 = interrupt is not pending.

1 = interrupt is pending.

Writing 1 to the ISPR bit corresponding to:

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

### 11.20.5 Interrupt Clear-pending Registers

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

- the register summary in Table 11-27 on page 144 for the register attributes
- Table 11-28 on page 145 for which interrupts are controlled by each register.

The bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| CLRPEND | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| CLRPEND | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CLRPEND | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CLRPEND | | | | | | | |

- **CLRPEND**

Interrupt clear-pending bits.

Write:

0 = no effect.

1 = removes pending state an interrupt.

Read:

0 = interrupt is not pending.

1 = interrupt is pending.

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

### 11.20.6 Interrupt Active Bit Registers

The IABR0 register indicates which interrupts are active. See:

- the register summary in Table 11-27 on page 144 for the register attributes
- Table 11-28 on page 145 for which interrupts are controlled by each register.

The bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| ACTIVE | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| ACTIVE | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| ACTIVE | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ACTIVE | | | | | | | |

- **ACTIVE**

Interrupt active flags:

0 = interrupt not active

1 = interrupt active.

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

### 11.20.7 Interrupt Priority Registers

The IPR0-IPR8 registers provide a 4-bit priority field for each interrupt (See the “Peripheral Identifiers” section of the datasheet for more details). These registers are byte-accessible. See the register summary in Table 11-27 on page 144 for their attributes. Each register holds four priority fields, that map up to four elements in the CMSIS interrupt priority array IP[0] to IP[32], as shown:

#### 11.20.7.1 IPRm

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

#### 11.20.7.2 IPR4

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

#### 11.20.7.3 IPR3

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

### 11.20.7.4 IPR2

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

### 11.20.7.5 IPR1

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

### 11.20.7.6 IPR0

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

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

Each priority field holds a priority value, 0-15. The lower the value, the greater the priority of the corresponding interrupt. The processor implements only bits[7:4] of each field, bits[3:0] read as zero and ignore writes.

See “The CMSIS mapping of the Cortex-M3 NVIC registers” on page 144 for more information about the IP[0] to IP[32] interrupt priority array, that provides the software view of the interrupt priorities.

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

- the corresponding IPR number, *M*, is given by *M* = *N* DIV 4
- the byte offset of the required Priority field in this register is *N* MOD 4, where:
  - byte offset 0 refers to register bits[7:0]

– byte offset 1 refers to register bits[15:8]
– byte offset 2 refers to register bits[23:16]
– byte offset 3 refers to register bits[31:24].

### 11.20.8 Software Trigger Interrupt Register

Write to the STIR to generate a *Software Generated Interrupt* (SGI). See the register summary in Table 11-27 on page 144 for the STIR attributes.

When the USERSETMPEND bit in the SCR is set to 1, unprivileged software can access the STIR, see "System Control Register" on page 167.

Only privileged software can enable unprivileged access to the STIR.

The bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | INTID |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| INTID | | | | | | | |

- **INTID**

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

### 11.20.9 Level-sensitive interrupts

The processor supports level-sensitive interrupts.

A level-sensitive interrupt is held asserted until the peripheral deasserts the interrupt signal. Typically this happens because the ISR accesses the peripheral, causing it to clear the interrupt request.

When the processor enters the ISR, it automatically removes the pending state from the interrupt, see “Hardware and software control of interrupts” . For a level-sensitive interrupt, if the signal is not deasserted before the processor returns from the ISR, the interrupt becomes pending again, and the processor must execute its ISR again. This means that the peripheral can hold the interrupt signal asserted until it no longer needs servicing.

#### 11.20.9.1 Hardware and software control of interrupts

The Cortex-M3 latches all interrupts. A peripheral interrupt becomes pending for one of the following reasons:

- the NVIC detects that the interrupt signal is HIGH and the interrupt is not active
- the NVIC detects a rising edge on the interrupt signal
- software writes to the corresponding interrupt set-pending register bit, see “Interrupt Set-pending Registers” on page 148, or to the STIR to make an SGI pending, see “Software Trigger Interrupt Register” on page 154.

A pending interrupt remains pending until one of the following:

The processor enters the ISR for the interrupt. This changes the state of the interrupt from pending to active. Then:

  - For a level-sensitive interrupt, when the processor returns from the ISR, the NVIC samples the interrupt signal. If the signal is asserted, the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR. Otherwise, the state of the interrupt changes to inactive.
  - If the interrupt signal is not pulsed while the processor is in the ISR, when the processor returns from the ISR the state of the interrupt changes to inactive.
- Software writes to the corresponding interrupt clear-pending register bit.

For a level-sensitive interrupt, if the interrupt signal is still asserted, the state of the interrupt does not change. Otherwise, the state of the interrupt changes to inactive.

### 11.20.10NVIC design hints and tips

Ensure software uses correctly aligned register accesses. The processor does not support unaligned accesses to NVIC registers. See the individual register descriptions for the supported access sizes.

A interrupt can enter pending state even it is disabled.

Before programming VTOR to relocate the vector table, ensure the vector table entries of the new vector table are setup for fault handlers and all enabled exception like interrupts. For more information see “Vector Table Offset Register” on page 163.

#### 11.20.10.1 NVIC programming hints

Software uses the CPSIE I and CPSID I instructions to enable and disable interrupts. The CMSIS provides the following intrinsic functions for these instructions:

```
void __disable_irq(void) // Disable Interrupts
void __enable_irq(void) // Enable Interrupts
```

In addition, the CMSIS provides a number of functions for NVIC control, including:

**Table 11-29. CMSIS functions for NVIC control**

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

For more information about these functions see the CMSIS documentation.

## 11.21 System control block

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

**Table 11-30. Summary of the system control block registers**

| Address | Name | Type | Required privilege | Reset value | Description |
|---|---|---|---|---|---|
| 0xE000E008 | ACTLR | RW | Privileged | 0x00000000 | "Auxiliary Control Register" on page 158 |
| 0xE000ED00 | CPUID | RO | Privileged | 0x412FC230 | "CPUID Base Register" on page 159 |
| 0xE000ED04 | ICSR | RW(1) | Privileged | 0x00000000 | "Interrupt Control and State Register" on page 160 |
| 0xE000ED08 | VTOR | RW | Privileged | 0x00000000 | "Vector Table Offset Register" on page 163 |
| 0xE000ED0C | AIRCR | RW(1) | Privileged | 0xFA050000 | "Application Interrupt and Reset Control Register" on page 164 |
| 0xE000ED10 | SCR | RW | Privileged | 0x00000000 | "System Control Register" on page 167 |
| 0xE000ED14 | CCR | RW | Privileged | 0x00000200 | "Configuration and Control Register" on page 168 |
| 0xE000ED18 | SHPR1 | RW | Privileged | 0x00000000 | "System Handler Priority Register 1" on page 171 |
| 0xE000ED1C | SHPR2 | RW | Privileged | 0x00000000 | "System Handler Priority Register 2" on page 172 |
| 0xE000ED20 | SHPR3 | RW | Privileged | 0x00000000 | "System Handler Priority Register 3" on page 173 |
| 0xE000ED24 | SHCRS | RW | Privileged | 0x00000000 | "System Handler Control and State Register" on page 174 |
| 0xE000ED28 | CFSR | RW | Privileged | 0x00000000 | "Configurable Fault Status Register" on page 176 |
| 0xE000ED28 | MMSR(2) | RW | Privileged | 0x00 | "Memory Management Fault Address Register" on page 183 |
| 0xE000ED29 | BFSR(2) | RW | Privileged | 0x00 | "Bus Fault Status Register" on page 178 |
| 0xE000ED2A | UFSR(2) | RW | Privileged | 0x0000 | "Usage Fault Status Register" on page 180 |
| 0xE000ED2C | HFSR | RW | Privileged | 0x00000000 | "Hard Fault Status Register" on page 182 |
| 0xE000ED34 | MMAR | RW | Privileged | Unknown | "Memory Management Fault Address Register" on page 183 |
| 0xE000ED38 | BFAR | RW | Privileged | Unknown | "Bus Fault Address Register" on page 184 |

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

### 11.21.1 The CMSIS mapping of the Cortex-M3 SCB registers

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

### 11.21.2 Auxiliary Control Register

The ACTLR provides disable bits for the following processor functions:

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

See the register summary in Table 11-30 on page 157 for the ACTLR attributes. The bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

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

- **DISFOLD**

When set to 1, disables IT folding. see "About IT folding" on page 158 for more information.

- **DISDEFWBUF**

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

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

- **DISMCYCINT**

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

#### 11.21.2.1 About IT folding

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

### 11.21.3 CPUID Base Register

The CPUID register contains the processor part number, version, and implementation information. See the register summary in Table 11-30 on page 157 for its attributes. The bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| Implementer | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| Variant | | | | Constant | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| PartNo | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| PartNo | | | | Revision | | | |

- **Implementer**

Implementer code:

0x41 = ARM

- **Variant**

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

0x2 = r2p0

- **Constant**

Reads as 0xF

- **PartNo**

Part number of the processor:

0xC23 = Cortex-M3

- **Revision**

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

0x0 = r2p0

### 11.21.4 Interrupt Control and State Register

The ICSR:

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

See the register summary in Table 11-30 on page 157, and the Type descriptions in Table 11-33 on page 186, for the ICSR attributes. The bit assignments are:

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

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

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

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| VECTACTIVE | | | | | | | |

- **PENDSVSET**

RW

PendSV set-pending bit.

Write:

0 = no effect

1 = changes PendSV exception state to pending.

Read:

0 = PendSV exception is not pending

1 = PendSV exception is pending.

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

- **PENDSVCLR**

WO

PendSV clear-pending bit.

Write:

0 = no effect

1 = removes the pending state from the PendSV exception.

- **PENDSTSET**

RW

SysTick exception set-pending bit.

Write:

0 = no effect

1 = changes SysTick exception state to pending.

Read:

0 = SysTick exception is not pending

1 = SysTick exception is pending.

- **PENDSTCLR**

WO

SysTick exception clear-pending bit.

Write:

0 = no effect

1 = removes the pending state from the SysTick exception.

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

- **Reserved for Debug use**

RO

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

- **ISRPENDING**

RO

Interrupt pending flag, excluding Faults:

0 = interrupt not pending

1 = interrupt pending.

- **VECTPENDING**

RO

Indicates the exception number of the highest priority pending enabled exception:

0 = no pending exceptions

Nonzero = the exception number of the highest priority pending enabled exception.

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

- **RETTOBASE**

RO

Indicates whether there are preempted active exceptions:

0 = there are preempted active exceptions to execute

1 = there are no active exceptions, or the currently-executing exception is the only active exception.

- **VECTACTIVE**

RO

Contains the active exception number:

0 = Thread mode

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

Subtract 16 from this value to obtain the IRQ number required to index into the Interrupt Clear-Enable, Set-Enable, Clear-Pending, Set-Pending, or Priority Registers, see “Interrupt Program Status Register” on page 45.

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

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

Note: 1. This is the same value as IPSR bits [8:0] see “Interrupt Program Status Register” on page 45.

### 11.21.5 Vector Table Offset Register

The VTOR indicates the offset of the vector table base address from memory address 0x00000000. See the register summary in Table 11-30 on page 157 for its attributes.

The bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| Reserved | | TBLOFF | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| TBLOFF | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| TBLOFF | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| TBLOFF | Reserved | | | | | | |

- **TBLOFF**

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

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

0 = code

1 = SRAM.

Bit[29] is sometimes called the TBLBASE bit.

When setting TBLOFF, you must align the offset to the number of exception entries in the vector table. The minimum alignment is 32 words, enough for up to 16 interrupts. For more interrupts, adjust the alignment by rounding up to the next power of two. For example, if you require 21 interrupts, the alignment must be on a 64-word boundary because the required table size is 37 words, and the next power of two is 64.

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

### 11.21.6 Application Interrupt and Reset Control Register

The AIRCR provides priority grouping control for the exception model, endian status for data accesses, and reset control of the system. See the register summary in Table 11-30 on page 157 and Table 11-33 on page 186 for its attributes.

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

The bit assignments are:

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

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| ENDIANESS | Reserved | | | | PRIGROUP | | |

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

- **VECTKEYSTAT**

Register Key:

Reads as 0xFA05

- **VECTKEY**

Register key:

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

- **ENDIANESS**

RO

Data endianness bit:

0 = Little-endian

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

- **PRIGROUP**

R/W

Interrupt priority grouping field. This field determines the split of group priority from subpriority, see “Binary point” on page 166.

- **SYSRESETREQ**

WO

System reset request:

0 = no effect

1 = asserts a proc_reset_signal.

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

This bit reads as 0.

- **VECTCLRACTIVE**

WO

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

- **VECTRESET**

WO

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

#### 11.21.6.1 Binary point

The PRIGROUP field indicates the position of the binary point that splits the PRI_*n* fields in the Interrupt Priority Registers into separate *group priority* and *subpriority* fields. Table 11-31 shows how the PRIGROUP value controls this split.

**Table 11-31. Priority grouping**

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

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

Determining preemption of an exception uses only the group priority field, see “Interrupt priority grouping” on page 66.

### 11.21.7 System Control Register

The SCR controls features of entry to and exit from low power state. See the register summary in Table 11-30 on page 157 for its attributes. The bit assignments are:

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

- **SEVONPEND**

Send Event on Pending bit:

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

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

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

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

- **SLEEPDEEP**

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

0 = sleep

1 = deep sleep.

- **SLEEPONEXIT**

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

0 = do not sleep when returning to Thread mode.

1 = enter sleep, or deep sleep, on return from an ISR.

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

### 11.21.8 Configuration and Control Register

The CCR controls entry to Thread mode and enables:

- the handlers for hard fault and faults escalated by FAULTMASK to ignore bus faults
- trapping of divide by zero and unaligned accesses
- access to the STIR by unprivileged software, see “Software Trigger Interrupt Register” on page 154.

See the register summary in Table 11-30 on page 157 for the CCR attributes.

The bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | STKALIGN | BFHFNMIGN |

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

- **STKALIGN**

Indicates stack alignment on exception entry:

0 = 4-byte aligned

1 = 8-byte aligned.

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

- **BFHFNMIGN**

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

0 = data bus faults caused by load and store instructions cause a lock-up

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

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

- **DIV_0_TRP**

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

0 = do not trap divide by 0

1 = trap divide by 0.

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

- **UNALIGN_TRP**

Enables unaligned access traps:

0 = do not trap unaligned halfword and word accesses

1 = trap unaligned halfword and word accesses.

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

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

- **USERSETMPEND**

Enables unprivileged software access to the STIR, see “Software Trigger Interrupt Register” on page 154:

0 = disable

1 = enable.

- **NONEBASETHRDENA**

Indicates how the processor enters Thread mode:

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

1 = processor can enter Thread mode from any level under the control of an EXC_RETURN value, see “Exception return” on page 68.

### 11.21.9 System Handler Priority Registers

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

SHPR1-SHPR3 are byte accessible. See the register summary in Table 11-30 on page 157 for their attributes.

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

**Table 11-32. System fault handler priority fields**

| Handler | Field | Register description |
|---|---|---|
| Memory management fault | PRI_4 | "System Handler Priority Register 1" on page 171 |
| Bus fault | PRI_5 | |
| Usage fault | PRI_6 | |
| SVCall | PRI_11 | "System Handler Priority Register 2" on page 172 |
| PendSV | PRI_14 | "System Handler Priority Register 3" on page 173 |
| SysTick | PRI_15 | |

Each PRI_N field is 8 bits wide, but the processor implements only bits[7:4] of each field, and bits[3:0] read as zero and ignore writes.

#### 11.21.9.1 System Handler Priority Register 1

The bit assignments are:

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

- **PRI_7**

Reserved

- **PRI_6**

Priority of system handler 6, usage fault

- **PRI_5**

Priority of system handler 5, bus fault

- **PRI_4**

Priority of system handler 4, memory management fault

### 11.21.9.2 System Handler Priority Register 2

The bit assignments are:

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

- **PRI_11**

Priority of system handler 11, SVCall

### 11.21.9.3 System Handler Priority Register 3

The bit assignments are:

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

- **PRI_15**

Priority of system handler 15, SysTick exception

- **PRI_14**

Priority of system handler 14, PendSV

### 11.21.10System Handler Control and State Register

The SHCSR enables the system handlers, and indicates:

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

See the register summary in Table 11-30 on page 157 for the SHCSR attributes. The bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

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

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

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

- **USGFAULTENA**

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

- **BUSFAULTENA**

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

- **MEMFAULTENA**

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

- **SVCALLPENDED**

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

- **BUSFAULTPENDED**

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

- **MEMFAULTPENDED**

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

- **USGFAULTPENDED**

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

- **SYSTICKACT**

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

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

- **PENDSVACT**

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

- **MONITORACT**

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

- **SVCALLACT**

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

- **USGFAULTACT**

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

- **BUSFAULTACT**

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

- **MEMFAULTACT**

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

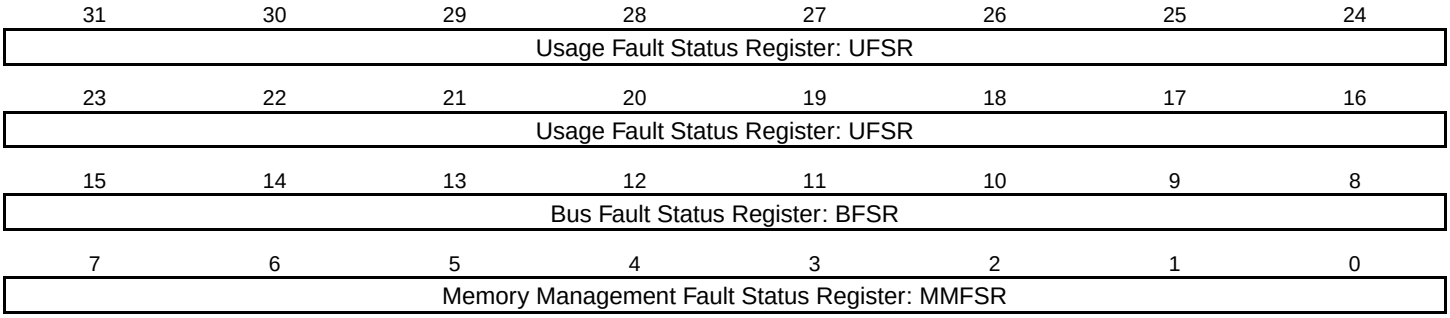
If you disable a system handler and the corresponding fault occurs, the processor treats the fault as a hard fault.

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

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

### 11.21.11 Configurable Fault Status Register

The CFSR indicates the cause of a memory management fault, bus fault, or usage fault. See the register summary in Table 11-30 on page 157 for its attributes. The bit assignments are:




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

- "Memory Management Fault Status Register" on page 177
- "Bus Fault Status Register" on page 178
- "Usage Fault Status Register" on page 180.

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

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

#### 11.21.11.1 Memory Management Fault Status Register

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

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

- **MMARVALID**

*Memory Management Fault Address Register* (MMAR) valid flag:

0 = value in MMAR is not a valid fault address

1 = MMAR holds a valid fault address.

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

- **MSTKERR**

Memory manager fault on stacking for exception entry:

0 = no stacking fault

1 = stacking for an exception entry has caused one or more access violations.

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

- **MUNSTKERR**

Memory manager fault on unstacking for a return from exception:

0 = no unstacking fault

1 = unstack for an exception return has caused one or more access violations.

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

- **DACCVIOL**

Data access violation flag:

0 = no data access violation fault

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

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

- **IACCVIOL**

Instruction access violation flag:

0 = no instruction access violation fault

1 = the processor attempted an instruction fetch from a location that does not permit execution.

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

#### 11.21.11.2 Bus Fault Status Register

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

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

- **BFARVALID**

*Bus Fault Address Register* (BFAR) valid flag:

0 = value in BFAR is not a valid fault address

1 = BFAR holds a valid fault address.

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

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

- **STKERR**

Bus fault on stacking for exception entry:

0 = no stacking fault

1 = stacking for an exception entry has caused one or more bus faults.

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

- **UNSTKERR**

Bus fault on unstacking for a return from exception:

0 = no unstacking fault

1 = unstack for an exception return has caused one or more bus faults.

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

- **IMPRECISERR**

Imprecise data bus error:

0 = no imprecise data bus error

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

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

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

- **PRECISERR**

Precise data bus error:

0 = no precise data bus error

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

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

- **IBUSERR**

Instruction bus error:

0 = no instruction bus error

1 = instruction bus error.

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

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

#### 11.21.11.3 Usage Fault Status Register

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | DIVBYZERO | UNALIGNED |

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

- **DIVBYZERO**

Divide by zero usage fault:

0 = no divide by zero fault, or divide by zero trapping not enabled

1 = the processor has executed an SDIV or UDIV instruction with a divisor of 0.

When the processor sets this bit to 1, the PC value stacked for the exception return points to the instruction that performed the divide by zero.

Enable trapping of divide by zero by setting the DIV_0_TRP bit in the CCR to 1, see “Configuration and Control Register” on page 168.

- **UNALIGNED**

Unaligned access usage fault:

0 = no unaligned access fault, or unaligned access trapping not enabled

1 = the processor has made an unaligned memory access.

Enable trapping of unaligned accesses by setting the UNALIGN_TRP bit in the CCR to 1, see “Configuration and Control Register” on page 168.

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

- **NOCP**

No coprocessor usage fault. The processor does not support coprocessor instructions:

0 = no usage fault caused by attempting to access a coprocessor

1 = the processor has attempted to access a coprocessor.

- **INVPC**

Invalid PC load usage fault, caused by an invalid PC load by EXC_RETURN:

0 = no invalid PC load usage fault

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

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

- **INVSTATE**

Invalid state usage fault:

0 = no invalid state usage fault

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

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

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

- **UNDEFINSTR**

Undefined instruction usage fault:

0 = no undefined instruction usage fault

1 = the processor has attempted to execute an undefined instruction.

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

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

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

### 11.21.12Hard Fault Status Register

The HFSR gives information about events that activate the hard fault handler. See the register summary in Table 11-30 on page 157 for its attributes.

This register is read, write to clear. This means that bits in the register read normally, but writing 1 to any bit clears that bit to 0. The bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| DEBUGEVT | FORCED | Reserved | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | VECTTBL | Reserved |

- **DEBUGEVT**

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

- **FORCED**

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

0 = no forced hard fault

1 = forced hard fault.

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

- **VECTTBL**

Indicates a bus fault on a vector table read during exception processing:

0 = no bus fault on vector table read

1 = bus fault on vector table read.

This error is always handled by the hard fault handler.

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

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

### 11.21.13Memory Management Fault Address Register

The MMFAR contains the address of the location that generated a memory management fault. See the register summary in Table 11-30 on page 157 for its attributes. The bit assignments are:

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

- **ADDRESS**

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

When an unaligned access faults, the address is the actual address that faulted. Because a single read or write instruction can be split into multiple aligned accesses, the fault address can be any address in the range of the requested access size.

Flags in the MMFSR indicate the cause of the fault, and whether the value in the MMFAR is valid. See “Memory Management Fault Status Register” on page 177.

### 11.21.14Bus Fault Address Register

The BFAR contains the address of the location that generated a bus fault. See the register summary in Table 11-30 on page 157 for its attributes. The bit assignments are:

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

- **ADDRESS**

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

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

Flags in the BFSR indicate the cause of the fault, and whether the value in the BFAR is valid. See “Bus Fault Status Register” on page 178.

### 11.21.15System control block design hints and tips

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

- except for the CFSR and SHPR1-SHPR3, it must use aligned word accesses
- for the CFSR and SHPR1-SHPR3 it can use byte or aligned halfword or word accesses.

The processor does not support unaligned accesses to system control block registers.

In a fault handler. to determine the true faulting address:

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

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

## 11.22 System timer, SysTick

The processor has a 24-bit system timer, SysTick, that counts down from the reload value to zero, reloads (wraps to) the value in the LOAD register on the next clock edge, then counts down on subsequent clocks.

When the processor is halted for debugging the counter does not decrement.

The system timer registers are:

**Table 11-33. System timer registers summary**

| Address | Name | Type | Required privilege | Reset value | Description |
|---|---|---|---|---|---|
| 0xE000E010 | CTRL | RW | Privileged | 0x00000004 | “SysTick Control and Status Register” on page 187 |
| 0xE000E014 | LOAD | RW | Privileged | 0x00000000 | “SysTick Reload Value Register” on page 188 |
| 0xE000E018 | VAL | RW | Privileged | 0x00000000 | “SysTick Current Value Register” on page 190 |
| 0xE000E01C | CALIB | RO | Privileged | 0x0002904 (1) | “SysTick Calibration Value Register” on page 191 |

1. SysTick calibration value.

### 11.22.1 SysTick Control and Status Register

The SysTick CTRL register enables the SysTick features. See the register summary in Table 11-33 on page 186 for its attributes. The bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | COUNTFLAG |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |

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

- **COUNTFLAG**

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

- **CLKSOURCE**

Indicates the clock source:

0 = MCK/8

1 = MCK

- **TICKINT**

Enables SysTick exception request:

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

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

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

- **ENABLE**

Enables the counter:

0 = counter disabled

1 = counter enabled.

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

### 11.22.2 SysTick Reload Value Register

The LOAD register specifies the start value to load into the VAL register. See the register summary in Table 11-33 on page 186 for its attributes. The bit assignments are:

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

- **RELOAD**

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

#### 11.22.2.1 Calculating the RELOAD value

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

The RELOAD value is calculated according to its use:

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

### 11.22.3 SysTick Current Value Register

The VAL register contains the current value of the SysTick counter. See the register summary in Table 11-33 on page 186 for its attributes. The bit assignments are:

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

- **CURRENT**

Reads return the current value of the SysTick counter.

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

### 11.22.4 SysTick Calibration Value Register

The CALIB register indicates the SysTick calibration properties. See the register summary in Table 11-33 on page 186 for its attributes. The bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| NOREF | SKEW | Reserved | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| TENMS | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| TENMS | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| TENMS | | | | | | | |

- **NOREF**

Reads as zero.

- **SKEW**

Reads as zero

- **TENMS**

Read as 0x0002904. The SysTick calibration value is fixed at 0x0002904 (10500), which allows the generation of a time base of 1 ms with SysTick clock at 6 MHz (48/8 = 6 MHz)

### 11.22.5 SysTick design hints and tips

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

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

## 11.23 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"

"Little-endian (LE)" *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.

.

Condition field

A four-bit field in an instruction that specifies a condition under which the instruction can execute.

Conditional execution

If the condition code flags indicate that the corresponding condition is true when the instruction starts executing, it executes normally. Otherwise, the instruction does nothing.

Context

The environment that each process operates in for a multitasking operating system. In ARM processors, this is limited to mean the physical address range that it can access in memory and the associated memory access permissions.

Coprocessor

A processor that supplements the main processor. Cortex-M3 does not support any coprocessors.

Debugger

A debugging system that includes a program, used to detect, locate, and correct software faults, together with custom hardware that supports software debugging.

Direct Memory Access (DMA)

An operation that accesses main memory directly, without the processor performing any accesses to the data concerned.

Doubleword

A 64-bit data item. The contents are taken as being an unsigned integer unless otherwise stated.

Doubleword-aligned

A data item having a memory address that is divisible by eight.

Endianness

Byte ordering. The scheme that determines the order that successive bytes of a data word are stored in memory. An aspect of the system's memory mapping.

*See also* "Little-endian (LE)"

Exception

An event that interrupts program execution. When an exception occurs, the processor suspends the normal program flow and starts execution at the address indicated by the corresponding exception vector. The indicated address contains the first instruction of the handler for the exception.

An exception can be an interrupt request, a fault, or a software-generated system exception. Faults include attempting an invalid memory access, attempting to execute an instruction in an invalid processor state, and attempting to execute an undefined instruction.

Exception service routine

*See* "Interrupt handler" .

Exception vector

*See* "Interrupt vector" .

Flat address mapping

A system of organizing memory in which each physical address in the memory space is the same as the corresponding virtual address.

Halfword

A 16-bit data item.

Illegal instruction

An instruction that is architecturally Undefined.

Implementation-defined

The behavior is not architecturally defined, but is defined and documented by individual implementations.

Implementation-specific

The behavior is not architecturally defined, and does not have to be documented by individual implementations. Used when there are a number of implementation options available and the option chosen does not affect software compatibility.

Index register

In some load and store instruction descriptions, the value of this register is used as an offset to be added to or subtracted from the base register value to form the address that is sent to memory. Some addressing modes optionally enable the index register value to be shifted prior to the addition or subtraction.

*See also* "Base register"

Instruction cycle count

The number of cycles that an instruction occupies the Execute stage of the pipeline.

Interrupt handler

A program that control of the processor is passed to when an interrupt occurs.

Interrupt vector

One of a number of fixed addresses in low memory, or in high memory if high vectors are configured, that contains the first instruction of the corresponding interrupt handler.

Little-endian (LE)

Byte ordering scheme in which bytes of increasing significance in a data word are stored at increasing addresses in memory.

*See also* ""Little-endian (LE)" See also "Little-endian memory" .Breakpoint" , "." , "Endianness" .

Little-endian memory

Memory in which:

a byte or halfword at a word-aligned address is the least significant byte or halfword within the word at that address

a byte at a halfword-aligned address is the least significant byte within the halfword at that address.

.

Load/store architecture

A processor architecture where data-processing operations only operate on register contents, not directly on memory contents.

Prefetching

In pipelined processors, the process of fetching instructions from memory to fill up the pipeline before the preceding instructions have finished executing. Prefetching an instruction does not mean that the instruction has to be executed.

Read

Reads are defined as memory operations that have the semantics of a load. Reads include the Thumb instructions LDM, LDR, LDRSH, LDRH, LDRSB, LDRB, and POP.

Region

A partition of memory space.

Reserved

A field in a control register or instruction format is reserved if the field is to be defined by the implementation, or produces Unpredictable results if the contents of the field are not zero. These fields are reserved for use in future extensions of the architecture or are implementation-specific. All reserved bits not used by the implementation must be written as 0 and read as 0.

Should Be One (SBO)

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

Should Be Zero (SBZ)

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

Should Be Zero or Preserved (SBZP)

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

Thread-safe

In a multi-tasking environment, thread-safe functions use safeguard mechanisms when accessing shared resources, to ensure correct operation without the risk of shared access conflicts.

Thumb instruction

One or two halfwords that specify an operation for a processor to perform. Thumb instructions must be halfword-aligned.

Unaligned

A data item stored at an address that is not divisible by the number of bytes that defines the data size is said to be unaligned. For example, a word stored at an address that is not divisible by four.

Undefined

Indicates an instruction that generates an Undefined instruction exception.

Unpredictable (UNP)

You cannot rely on the behavior. Unpredictable behavior must not represent security holes. Unpredictable behavior must not halt or hang the processor, or any parts of the system.

Warm reset

Also known as a core reset. Initializes the majority of the processor excluding the debug controller and debug logic. This type of reset is useful if you are using the debugging features of a processor.

Word

A 32-bit data item.

Write

Writes are defined as operations that have the semantics of a store. Writes include the Thumb instructions STM, STR, STRH, STRB, and PUSH.

# 12. Debug and Test Features

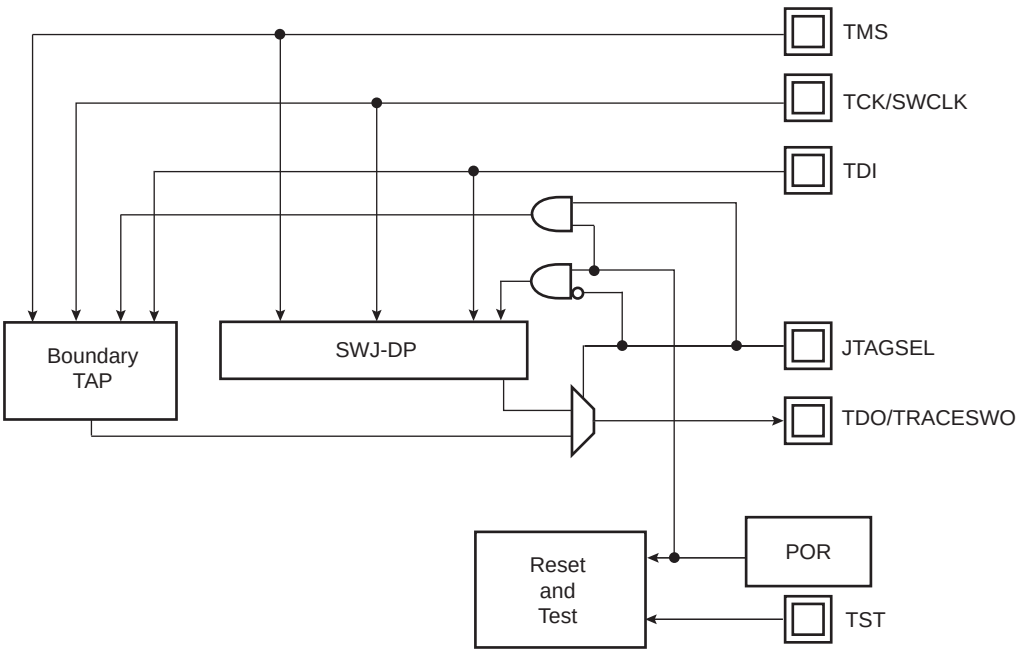
## 12.1 Description

The SAM3 Series Microcontrollers feature a number of complementary debug and test capabilities. The Serial Wire/JTAG Debug Port (SWJ-DP) combining a Serial Wire Debug Port (SW-DP) and JTAG Debug (JTAG-DP) port is used for standard debugging functions, such as downloading code and single-stepping through programs. It also embeds a serial wire trace.

## 12.2 Embedded Characteristics

- Debug access to all memory and registers in the system, including Cortex-M3 register bank when the core is running, halted, or held in reset.
- Serial Wire Debug Port (SW-DP) and Serial Wire JTAG Debug Port (SWJ-DP) debug access
- Flash Patch and Breakpoint (FPB) unit for implementing breakpoints and code patches
- Data Watchpoint and Trace (DWT) unit for implementing watchpoints, data tracing, and system profiling
- Instrumentation Trace Macrocell (ITM) for support of printf style debugging
- IEEE1149.1 JTAG Boundary-scan on All Digital Pins

**Figure 12-1. Debug and Test Block Diagram**

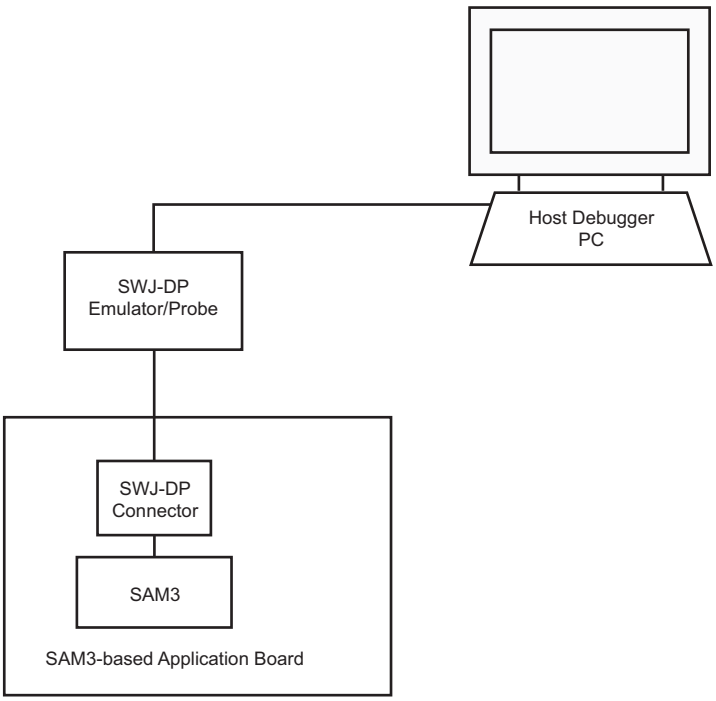


## 12.3 Application Examples

### 12.3.1 Debug Environment

Figure 12-2 shows a complete debug environment example. The SWJ-DP interface is used for standard debugging functions, such as downloading code and single-stepping through the program and viewing core and peripheral registers.

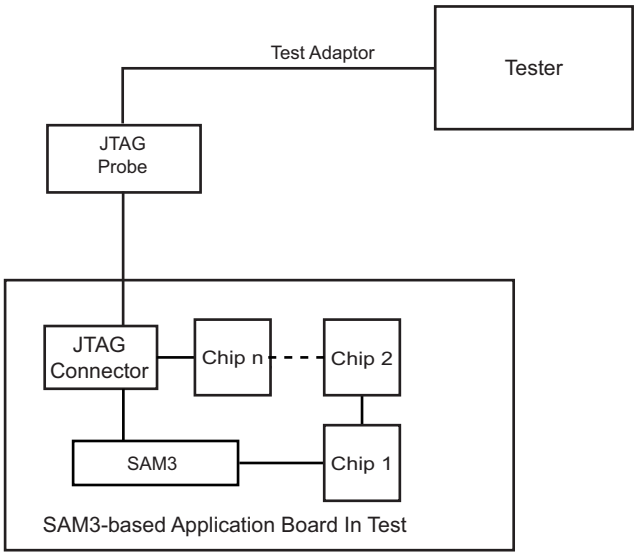
**Figure 12-2. Application Debug Environment Example**




### 12.3.2 Test Environment

Figure 12-3 shows a test environment example (JTAG Boundary scan). Test vectors are sent and interpreted by the tester. In this example, the “board in test” is designed using a number of JTAG-compliant devices. These devices can be connected to form a single scan chain.

**Figure 12-3. Application Test Environment Example**




## 12.4 Debug and Test Pin Description

**Table 12-1. Debug and Test Signal List**

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

1. TDO pin is set in input mode when the Cortex-M3 Core is not in debug mode. Thus the internal pull-up corresponding to this PIO line must be enabled to avoid current consumption due to floating input.

## 12.5 Functional Description

### 12.5.1 Test Pin

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

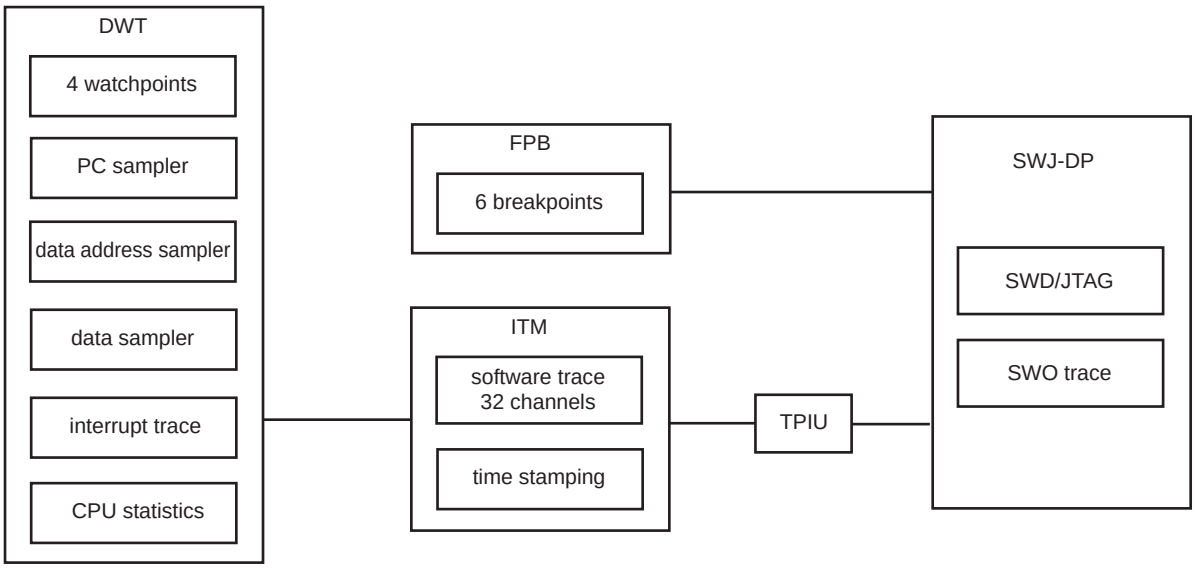
### 12.5.2 Debug Architecture

Figure 12-4 shows the Debug Architecture used in the SAM3. The Cortex-M3 embeds five functional units for debug:

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

The debug architecture information that follows is mainly dedicated to developers of SWJ-DP Emulators/Probes and debugging tool vendors for Cortex M3-based microcontrollers. For further details on SWJ-DP see the Cortex M3 technical reference manual.

**Figure 12-4. Debug Architecture**




### 12.5.3 Serial Wire/JTAG Debug Port (SWJ-DP)

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

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

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

**Table 12-2. SWJ-DP Pin List**

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

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

#### 12.5.3.1 SW-DP and JTAG-DP Selection Mechanism

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

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

### 12.5.4 FPB (Flash Patch Breakpoint)

The FPB:

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

The FPB unit contains:

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

### 12.5.5 DWT (Data Watchpoint and Trace)

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

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

The DWT contains counters for the following items:

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

### 12.5.6 ITM (Instrumentation Trace Macrocell)

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

- **Software trace**: Software can write directly to ITM stimulus registers. This can be done thanks to the “printf” function. For more information, refer to Section 12.5.6.1 “How to Configure the ITM”.
- **Hardware trace**: The ITM emits packets generated by the DWT.
- **Time stamping**: Timestamps are emitted relative to packets. The ITM contains a 21-bit counter to generate the timestamp.

#### 12.5.6.1 How to Configure the ITM

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

- Configure the TPIU for asynchronous trace mode (refer to Section 12.5.6.3 “5.4.3. How to Configure the TPIU”)
- Enable the write accesses into the ITM registers by writing “0xC5ACCE55” into the Lock Access Register (Address: 0xE0000FB0)
- Write 0x00010015 into the Trace Control Register:
  - Enable ITM
  - Enable Synchronization packets
  - Enable SWO behavior
  - Fix the ATB ID to 1
- Write 0x1 into the Trace Enable Register:
  - Enable the Stimulus port 0
- Write 0x1 into the Trace Privilege Register:
  - Stimulus port 0 only accessed in privileged mode (Clearing a bit in this register will result in the corresponding stimulus port being accessible in user mode.)
- Write into the Stimulus port 0 register: TPIU (Trace Port Interface Unit)

  The TPIU acts as a bridge between the on-chip trace data and the Instruction Trace Macrocell (ITM).

  The TPIU formats and transmits trace data off-chip at frequencies asynchronous to the core.

#### 12.5.6.2 Asynchronous Mode

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

Two encoding formats are available for the single pin output:

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

#### 12.5.6.3 5.4.3. How to Configure the TPIU

This example only concerns the asynchronous trace mode.

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

### 12.5.7 IEEE® 1149.1 JTAG Boundary Scan

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

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

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

#### 12.5.7.1 JTAG Boundary-scan Register

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

Each SAM3  input/output pin corresponds to a 3-bit register in the BSR. The OUTPUT bit contains data that can be forced on the pad. The INPUT bit facilitates the observability of data applied to the pad. The CONTROL bit selects the direction of the pad.

For more information, please refer to BDSL files available for the SAM3 Series.

### 12.5.8 ID Code Register

Access: Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| VERSION | | | | PART NUMBER | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| PART NUMBER | | | | | | | |

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

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| MANUFACTURER IDENTITY | | | | | | | 1 |

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

Set to 0x0.

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

| Chip Name | Chip ID |
|---|---|
| SAM3N | 0x05B2E |

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

Set to 0x01F.

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

Set to 0x1.

| Chip Name | JTAG ID Code |
|---|---|
| SAM3N | 0x05B2E03F |

# 13. Reset Controller (RSTC)

## 13.1 Description

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

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

## 13.2 Embedded Characteristics

The Reset Controller is based on a Power-on-Reset cell, and a Supply Monitor on VDDCORE.

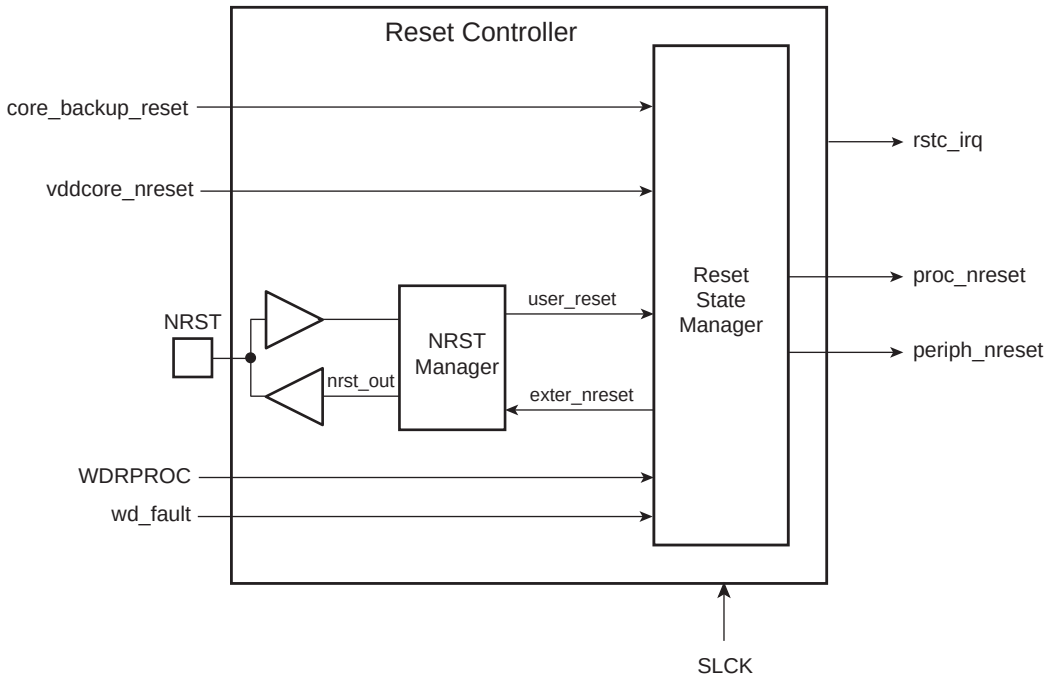
The Reset Controller is capable to return to the software the source of the last reset, either a general reset, a wake-up reset, a software reset, a user reset or a watchdog reset.

The Reset Controller controls the internal resets of the system and the NRST pin input/output. It is capable to shape a reset signal for the external devices, simplifying to a minimum connection of a push-button on the NRST pin to implement a manual reset.

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

## 13.3 Block Diagram

**Figure 13-1. Reset Controller Block Diagram**



## 13.4 Functional Description

### 13.4.1 Reset Controller Overview

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

- proc_nreset: Processor reset line. It also resets the Watchdog Timer.
- periph_nreset: Affects the whole set of embedded peripherals.
- nrst_out: Drives the NRST pin.

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

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

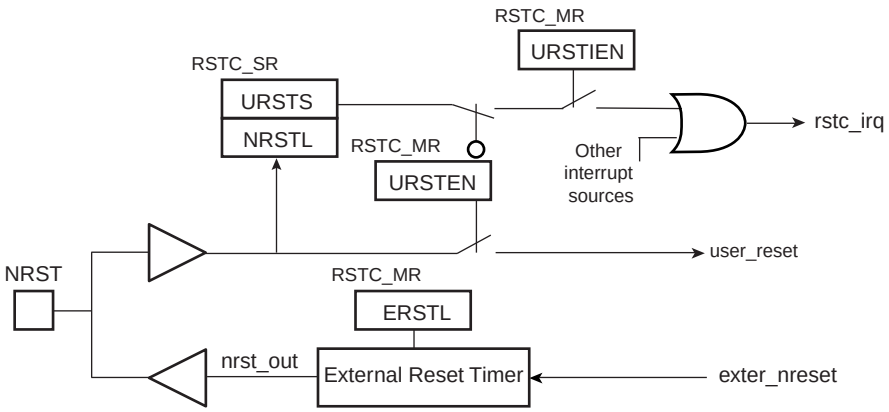
The Reset Controller Mode Register (RSTC_MR), allowing the configuration of the Reset Controller, is powered with VDDIO, so that its configuration is saved as long as VDDIO is on.

### 13.4.2 NRST Manager

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

The NRST Manager samples the NRST input pin and drives this pin low when required by the Reset State Manager. Figure 13-2 shows the block diagram of the NRST Manager.

**Figure 13-2. NRST Manager**




#### 13.4.2.1 NRST Signal or Interrupt

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

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

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

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

#### 13.4.2.2 NRST External Reset Control

The Reset State Manager asserts the signal ext_nreset to assert the NRST pin. When this occurs, the “nrst_out” signal is driven low by the NRST Manager for a time programmed by the field ERSTL in RSTC_MR. This assertion duration, named EXTERNAL_RESET_LENGTH, lasts $2^{(ERSTL+1)}$ Slow Clock cycles. This gives the approximate duration of an assertion between 60 µs and 2 seconds. Note that ERSTL at 0 defines a two-cycle duration for the NRST pulse.

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

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

### 13.4.3 Brownout Manager

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

### 13.4.4 Reset States

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

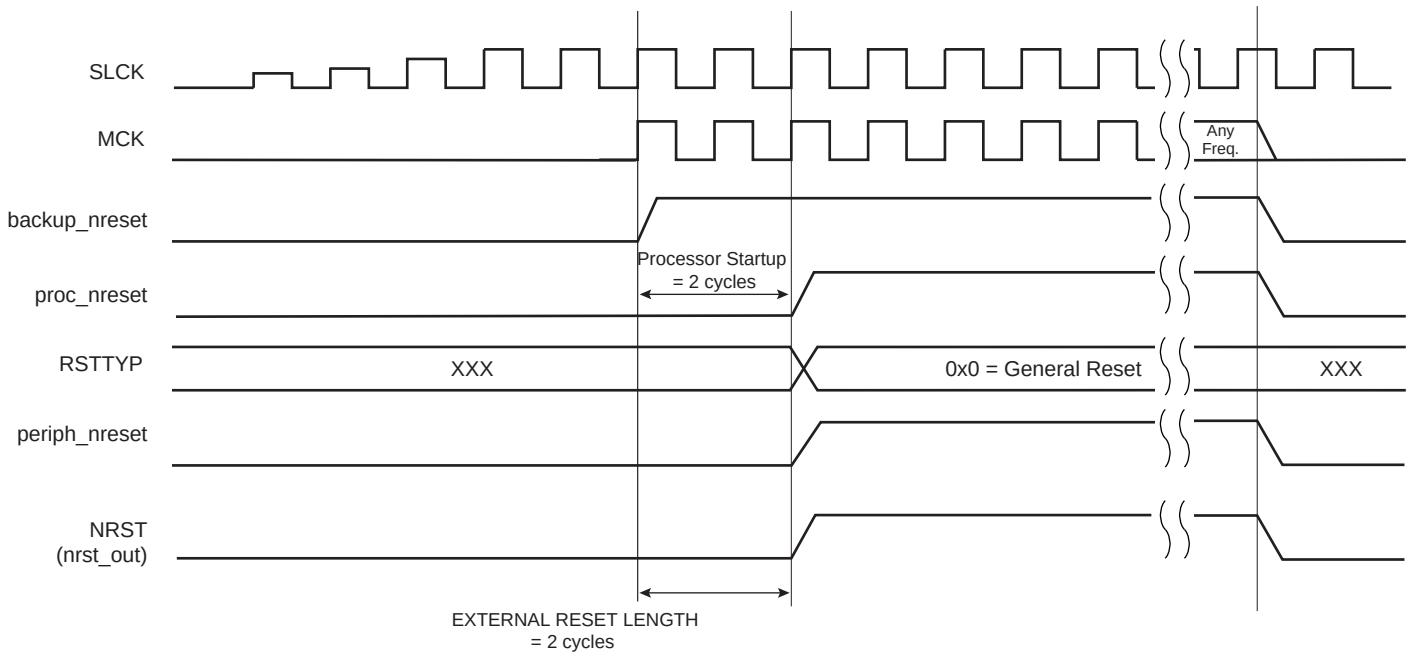
#### 13.4.4.1 General Reset

A general reset occurs when a 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 the field RSTTYP in RSTC_SR reports a General Reset. As the RSTC_MR is reset, the NRST line rises 2 cycles after the vddcore_nreset, as ERSTL defaults at value 0x0.

Figure 13-3 shows how the General Reset affects the reset signals.

**Figure 13-3. General Reset State**



#### 13.4.4.2 Backup Reset

A Backup reset occurs when the chip returns from Backup mode. The core_backup_reset signal is asserted by the Supply Controller when a Backup reset occurs.

The field RSTTYP in RSTC_SR is updated to report a Backup Reset.

#### 13.4.4.3 User Reset

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

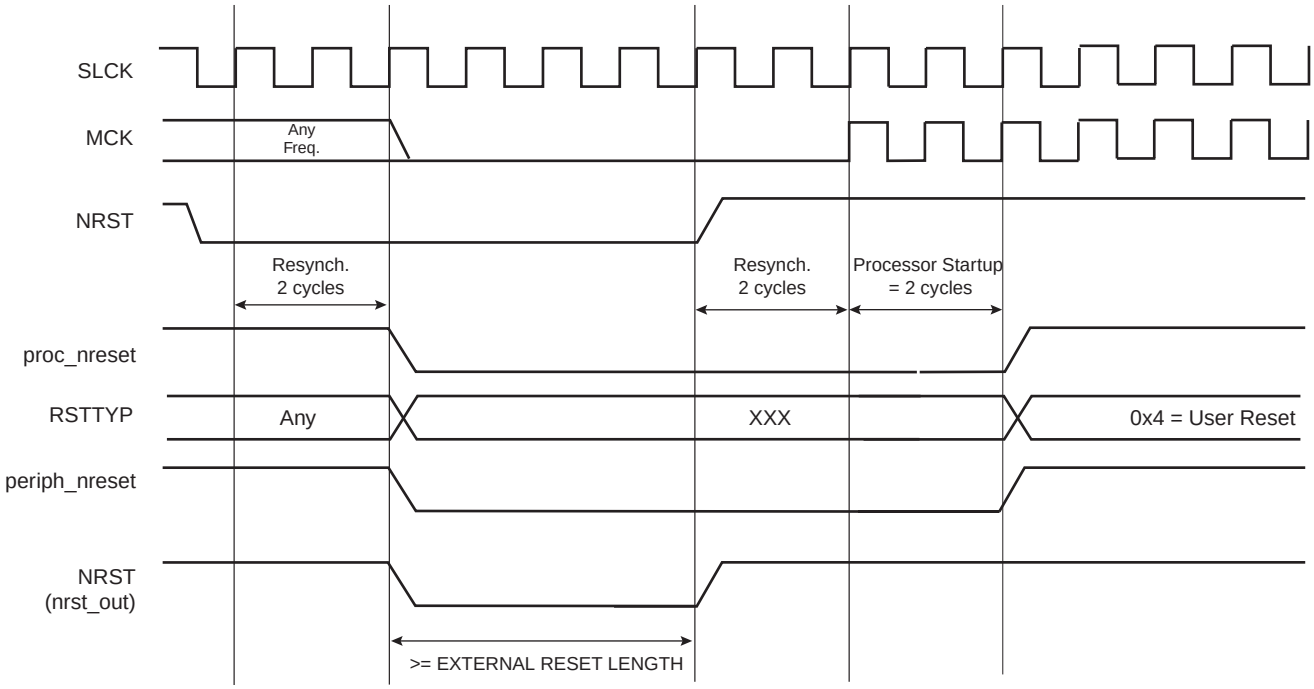
The User Reset is entered as soon as a low level is detected on NRST. The Processor Reset and the Peripheral Reset are asserted.

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

When the processor reset signal is released, the RSTTYP field of the Status Register (RSTC_SR) is loaded with the value 0x4, indicating a User Reset.

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

**Figure 13-4. User Reset State**




#### 13.4.4.4 Software Reset

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

- PROCRST: Writing PROCRST at 1 resets the processor and the watchdog timer.
- PERRST: Writing PERRST at 1 resets all the embedded peripherals, including the memory system, and, in particular, the Remap Command. The Peripheral Reset is generally used for debug purposes.
  Except for debug purposes , PERRST must always be used in conjunction with PROCRST (PERRST and PROCRST set both at 1 simultaneously).

- EXTRST: Writing EXTRST at 1 asserts low the NRST pin during a time defined by the field ERSTL in the Mode Register (RSTC_MR).

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

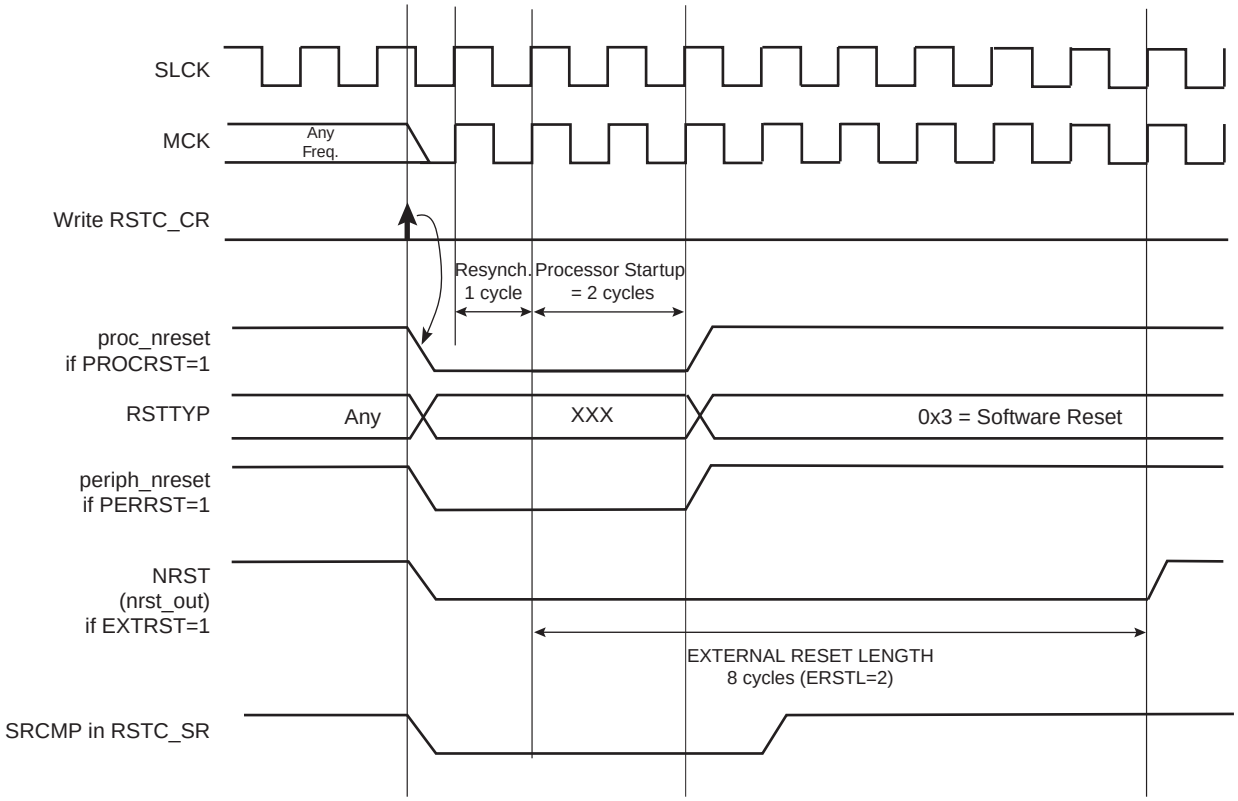
The internal reset signals are asserted as soon as the register write is performed. This is detected on the Master Clock (MCK). They are released when the software reset is left, i.e.; synchronously to SLCK.

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

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

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

**Figure 13-5. Software Reset**




### 13.4.4.5 Watchdog Reset

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

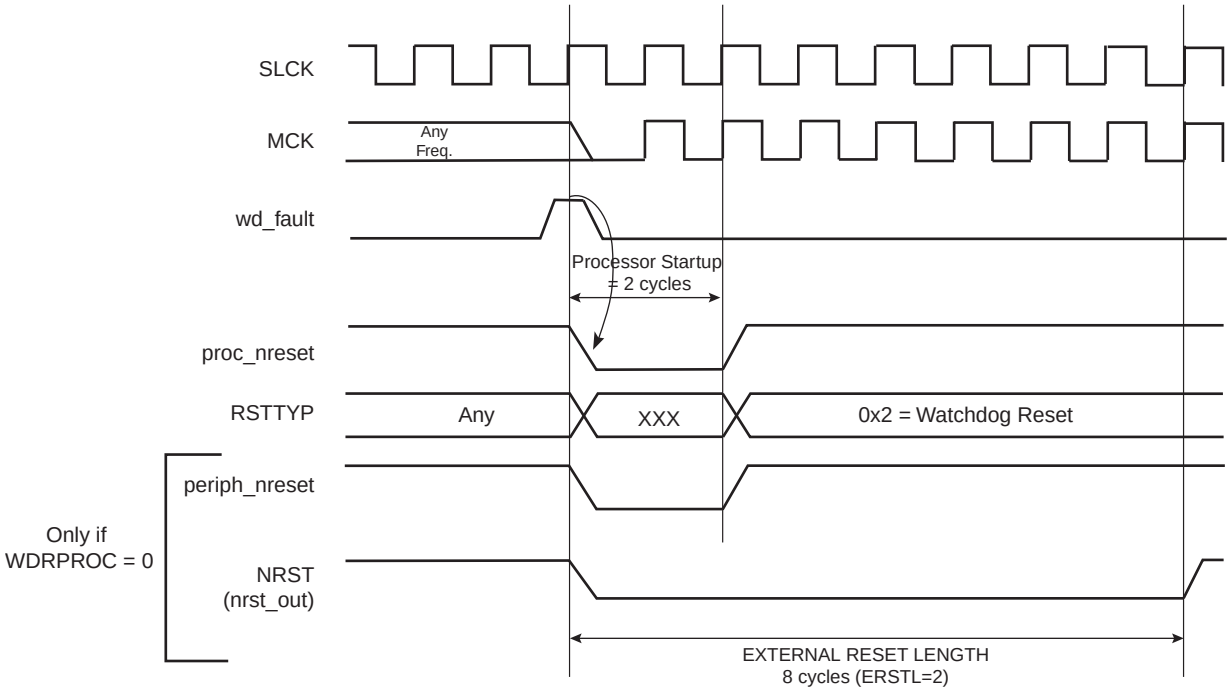
When in Watchdog Reset, assertion of the reset signals depends on the WDRPROC bit in WDT_MR:

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

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

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

**Figure 13-6. Watchdog Reset**




### 13.4.5 Reset State Priorities

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

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

Particular cases are listed below:

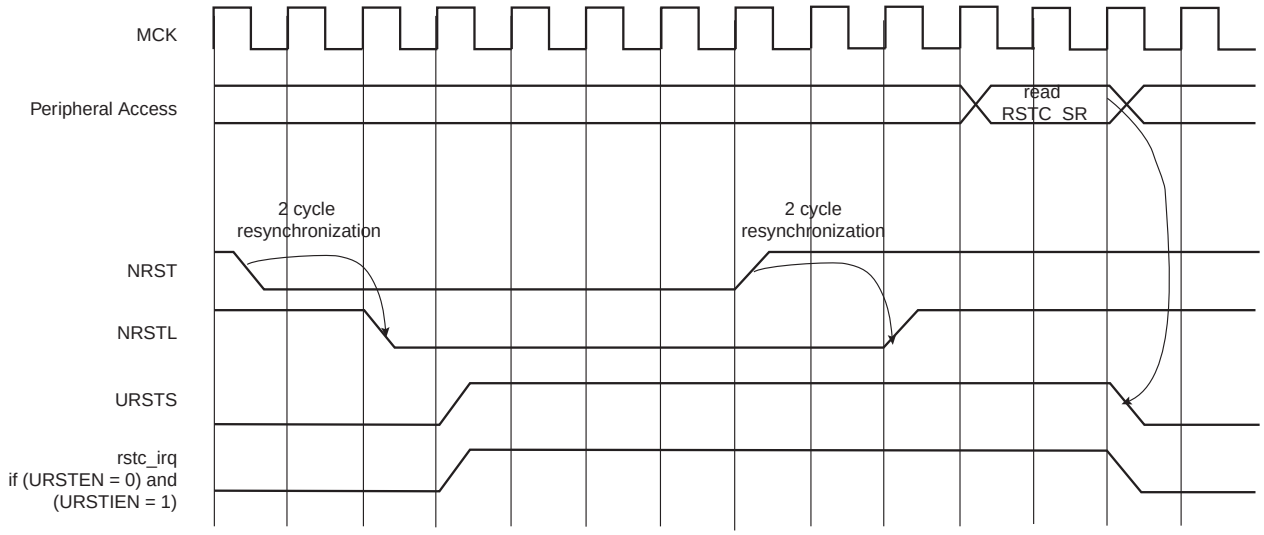
- When in User Reset:
  - A watchdog event is impossible because the Watchdog Timer is being reset by the proc_nreset signal.
  - A software reset is impossible, since the processor reset is being activated.
- When in Software Reset:
  - A watchdog event has priority over the current state.
  - The NRST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

### 13.4.6 Reset Controller Status Register

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

- RSTTYP field: This field gives the type of the last reset, as explained in previous sections.
- SRCMP bit: This field indicates that a Software Reset Command is in progress and that no further software reset should be performed until the end of the current one. This bit is automatically cleared at the end of the current software reset.
- NRSTL bit: The NRSTL bit of the Status Register gives the level of the NRST pin sampled on each MCK rising edge.
- URSTS bit: A high-to-low transition of the NRST pin sets the URSTS bit of the RSTC_SR register. This transition is also detected on the Master Clock (MCK) rising edge (see Figure 13-7). If the User Reset is disabled (URSTEN = 0) and if the interruption is enabled by the URSTIEN bit in the RSTC_MR register, the URSTS bit triggers an interrupt. Reading the RSTC_SR status register resets the URSTS bit and clears the interrupt.

**Figure 13-7. Reset Controller Status and Interrupt**



## 13.5 Reset Controller (RSTC) User Interface

**Table 13-1. Register Mapping**

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

### 13.5.1 Reset Controller 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 is correct, resets the processor.

- **PERRST: Peripheral Reset**

0 = No effect.

1 = If KEY is correct, resets the peripherals.

- **EXTRST: External Reset**

0 = No effect.

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

- **KEY: Password**

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

### 13.5.2 Reset Controller 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**

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

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

- **RSTTYP: Reset Type**

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

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

- **NRSTL: NRST Pin Level**

Registers the NRST Pin Level at Master Clock (MCK).

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

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

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

### 13.5.3 Reset Controller Mode Register

**Name:** RSTC_MR

**Address:** 0x400E1408

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| KEY | | | | | | | |

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | ERSTL | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | | URSTIEN | – | – | – | URSTEN |

- **URSTEN: User Reset Enable**

0 = The detection of a low level on the pin NRST does not generate a User Reset.

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

- **URSTIEN: User Reset Interrupt Enable**

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

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

- **ERSTL: External Reset Length**

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

- **KEY: Password**

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

# 14. Real-time Timer (RTT)

## 14.1 Description

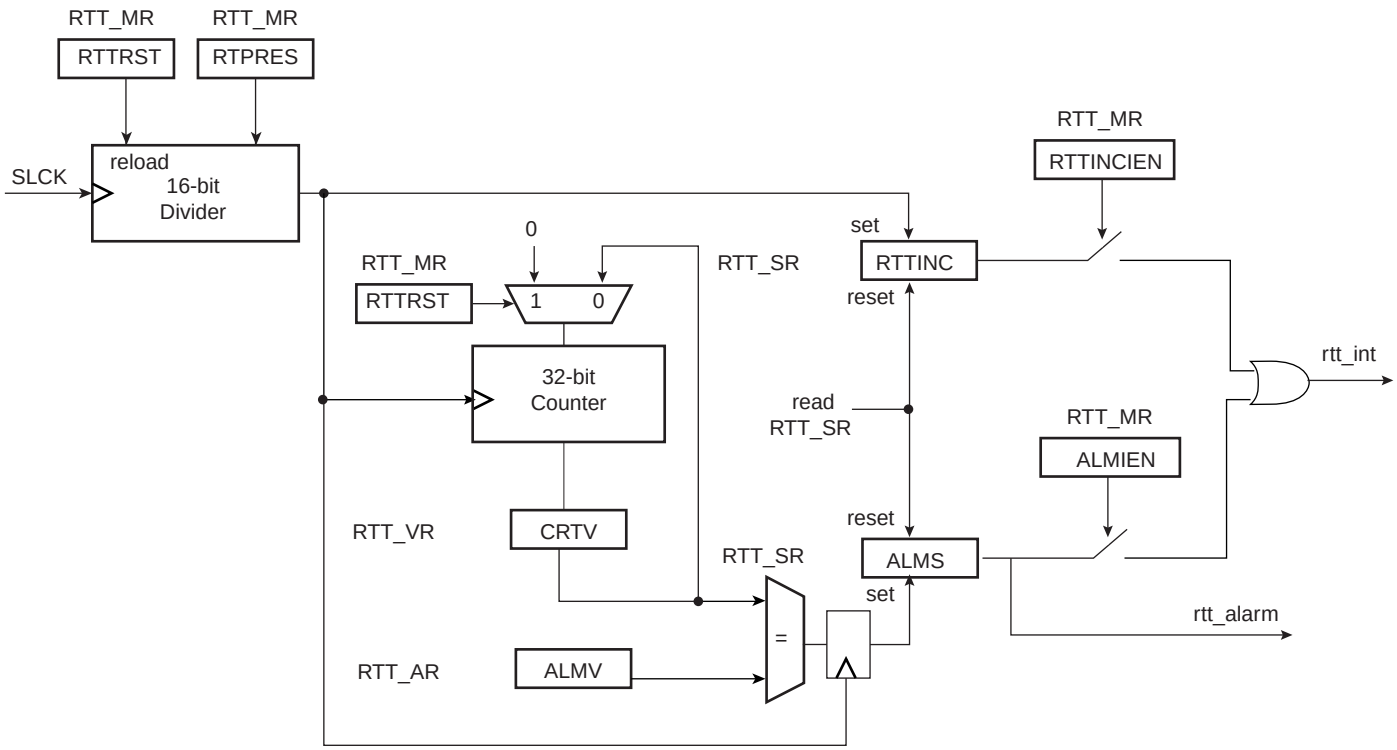
The Real-time Timer is built around a 32-bit counter used to count roll-over events of the programmable16-bit prescaler which enables counting elapsed seconds from a 32 kHz slow clock source. It generates a periodic interrupt and/or triggers an alarm on a programmed value.

## 14.2 Embedded Characteristics

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

## 14.3 Block Diagram

**Figure 14-1. Real-time Timer**



## 14.4 Functional Description

The Real-time Timer can be used to count elapsed seconds. It is built around a 32-bit counter fed by Slow Clock divided by a programmable 16-bit value. The value can be programmed in the field RTPRES of the Real-time Mode Register (RTT_MR).

Programming RTPRES at 0x00008000 corresponds to feeding the real-time counter with a 1 Hz signal (if the Slow Clock is 32.768 kHz). The 32-bit counter can count up to $2^{32}$ seconds, corresponding to more than 136 years, then roll over to 0.

The Real-time Timer can also be used as a free-running timer with a lower time-base. The best accuracy is achieved by writing RTPRES to 3. Programming RTPRES to 1 or 2 is possible, but may result in losing status events because the status register is cleared two Slow Clock cycles after read. Thus if the RTT is configured to trigger an interrupt, the interrupt occurs during 2 Slow Clock cycles after reading RTT_SR. To prevent several executions of the interrupt handler, the interrupt must be disabled in the interrupt handler and re-enabled when the status register is clear.

The Real-time Timer value (CRTV) can be read at any time in the register RTT_VR (Real-time Value Register). As this value can be updated asynchronously from the Master Clock, it is advisable to read this register twice at the same value to improve accuracy of the returned value.

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

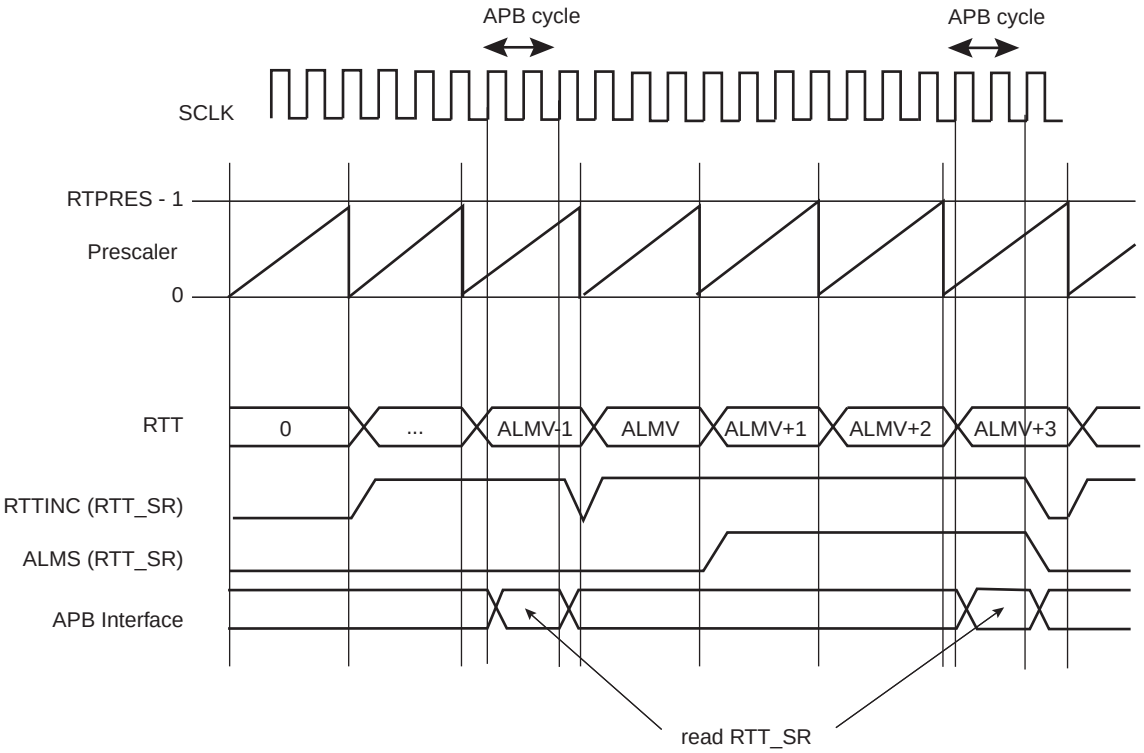
The bit RTTINC in RTT_SR is set each time the Real-time Timer counter is incremented. This bit can be used to start a periodic interrupt, the period being one second when the RTPRES is programmed with 0x8000 and Slow Clock equal to 32.768 Hz.

Reading the RTT_SR status register resets the RTTINC and ALMS fields.

Writing the bit RTTRST in RTT_MR immediately reloads and restarts the clock divider with the new programmed value. This also resets the 32-bit counter.

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

**Figure 14-2. RTT Counting**



## 14.5 Real-time Timer (RTT) User Interface

**Table 14-1. Register Mapping**

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

### 14.5.1 Real-time Timer Mode Register

**Register Name:** RTT_MR

**Address:** 0x400E1430

**Access Type:** Read-write

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

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | RTTRST | RTTINCIEN | ALMIEN |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| RTPRES | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| RTPRES | | | | | | | |

- **RTPRES: Real-time Timer Prescaler Value**

Defines the number of SLCK periods required to increment the Real-time timer. RTPRES is defined as follows:

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

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

- **ALMIEN: Alarm Interrupt Enable**

0 = The bit ALMS in RTT_SR has no effect on interrupt.

1 = The bit ALMS in RTT_SR asserts interrupt.

- **RTTINCIEN: Real-time Timer Increment Interrupt Enable**

0 = The bit RTTINC in RTT_SR has no effect on interrupt.

1 = The bit RTTINC in RTT_SR asserts interrupt.

- **RTTRST: Real-time Timer Restart**

0 = No effect.

1 = Reloads and restarts the clock divider with the new programmed value. This also resets the 32-bit counter.

### 14.5.2 Real-time Timer Alarm Register

**Register Name:** RTT_AR

**Address:** 0x400E1434

**Access Type:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| ALMV | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| ALMV | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| ALMV | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ALMV | | | | | | | |

- **ALMV: Alarm Value**

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

### 14.5.3 Real-time Timer Value Register

**Register Name:** RTT_VR

**Address:** 0x400E1438

**Access Type:** 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.

### 14.5.4 Real-time Timer Status Register

**Register Name:** RTT_SR

**Address:** 0x400E143C

**Access Type:** Read-only

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

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

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

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

- **RTTINC: Real-time Timer Increment**

0 = The Real-time Timer has not been incremented since the last read of the RTT_SR.

1 = The Real-time Timer has been incremented since the last read of the RTT_SR.

# 15. Real Time Clock (RTC)

## 15.1 Description

The Real-time Clock (RTC) peripheral is designed for very low power consumption.

It combines a complete time-of-day clock with alarm and a two-hundred-year Gregorian calendar, complemented by a programmable periodic interrupt. The alarm and calendar registers are accessed by a 32-bit data bus.

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

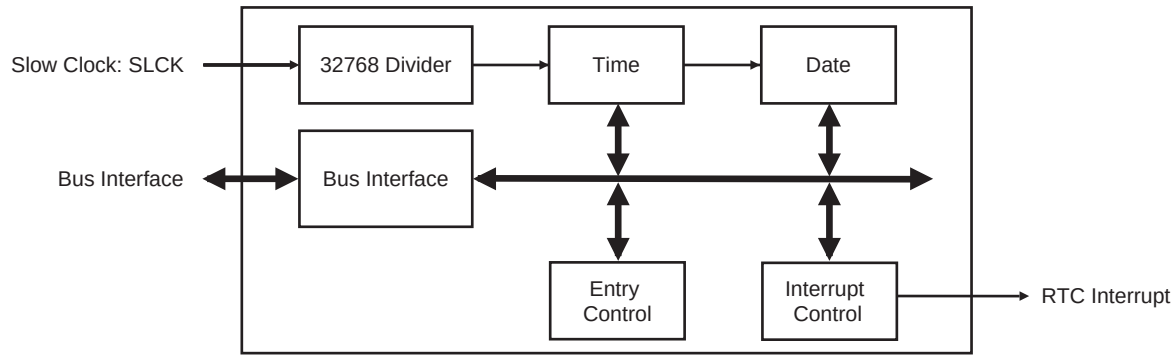
Updating time and calendar fields and configuring the alarm fields are performed by a parallel capture on the 32-bit data bus. An entry control is performed to avoid loading registers with incompatible BCD format data or with an incompatible date according to the current month/year/century.

## 15.2 Embedded Characteristics

- Low Power Consumption
- Full asynchronous design
- Two hundred year calendar
- Programmable Periodic Interrupt
- Alarm and update parallel load
- Control of alarm and update Time/Calendar Data In

## 15.3 Block Diagram

**Figure 15-1. RTC Block Diagram**



## 15.4 Product Dependencies

### 15.4.1 Power Management

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

### 15.4.2 Interrupt

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

## 15.5 Functional Description

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

The valid year range is 1900 to 2099 in Gregorian mode, a two-hundred-year calendar.

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

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

### 15.5.1 Reference Clock

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

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

### 15.5.2 Timing

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

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

### 15.5.3 Alarm

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

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

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

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

### 15.5.4 Error Checking

Verification on user interface data is performed when accessing the century, year, month, date, day, hours, minutes, seconds and alarms. A check is performed on illegal BCD entries such as illegal date of the month with regard to the year and century configured.

If one of the time fields is not correct, the data is not loaded into the register/counter and a flag is set in the validity register. The user can not reset this flag. It is reset as soon as an acceptable value is programmed. This avoids any further side effects in the hardware. The same procedure is done for the alarm.

The following checks are performed:

1. Century (check if it is in range 19 - 20 )
2. Year (BCD entry check)
3. Date (check range 01 - 31)
4. Month (check if it is in BCD range 01 - 12, check validity regarding “date”)
5. Day (check range 1 - 7)
6. Hour (BCD checks: in 24-hour mode, check range 00 - 23 and check that AM/PM flag is not set if RTC is set in 24-hour mode; in 12-hour mode check range 01 - 12)
7. Minute (check BCD and range 00 - 59)
8. Second (check BCD and range 00 - 59)

Note: If the 12-hour mode is selected by means of the RTC_MODE register, a 12-hour value can be programmed and the returned value on RTC_TIME will be the corresponding 24-hour value. The entry control checks the value of the AM/PM indicator (bit 22 of RTC_TIME register) to determine the range to be checked.

### 15.5.5 Updating Time/Calendar

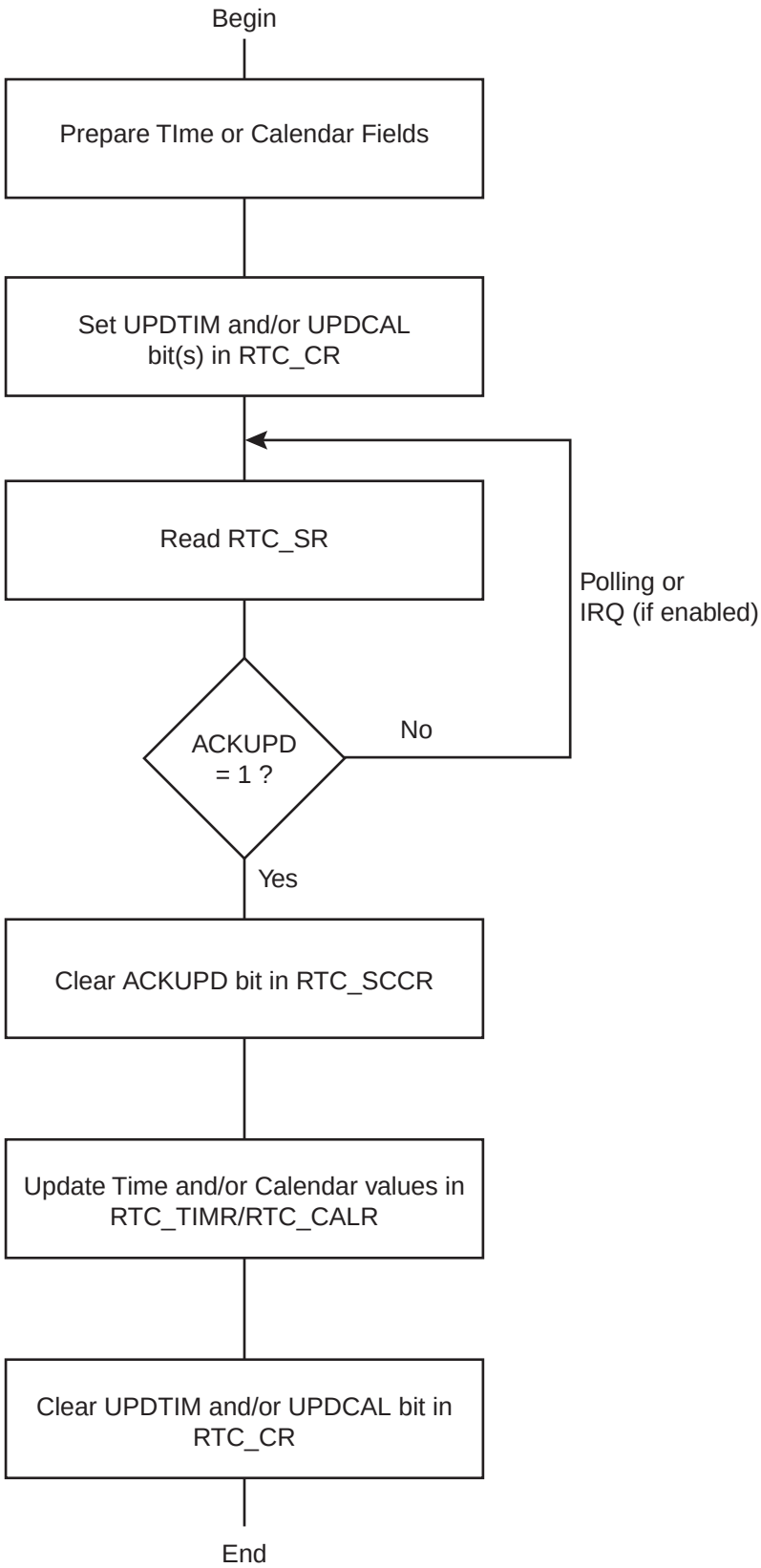
To update any of the time/calendar fields, the user must first stop the RTC by setting the corresponding field in the Control Register. Bit UPDTIM must be set to update time fields (hour, minute, second) and bit UPDCAL must be set to update calendar fields (century, year, month, date, day).

Then the user must poll or wait for the interrupt (if enabled) of bit ACKUPD in the Status Register. Once the bit reads 1, it is mandatory to clear this flag by writing the corresponding bit in RTC_SCCR. The user can now write to the appropriate Time and Calendar register.

Once the update is finished, the user must reset (0) UPDTIM and/or UPDCAL in the Control

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

**Figure 15-2. Update Sequence**



## 15.6 Real Time Clock (RTC) User Interface

**Table 15-1. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x00 | Control Register | RTC_CR | Read-write | 0x0 |
| 0x04 | Mode Register | RTC_MR | Read-write | 0x0 |
| 0x08 | Time Register | RTC_TIMR | Read-write | 0x0 |
| 0x0C | Calendar Register | RTC_CALR | Read-write | 0x01210720 |
| 0x10 | Time Alarm Register | RTC_TIMALR | Read-write | 0x0 |
| 0x14 | Calendar Alarm Register | RTC_CALALR | Read-write | 0x01010000 |
| 0x18 | Status Register | RTC_SR | Read-only | 0x0 |
| 0x1C | Status Clear Command Register | RTC_SCCR | Write-only | – |
| 0x20 | Interrupt Enable Register | RTC_IER | Write-only | – |
| 0x24 | Interrupt Disable Register | RTC_IDR | Write-only | – |
| 0x28 | Interrupt Mask Register | RTC_IMR | Read-only | 0x0 |
| 0x2C | Valid Entry Register | RTC_VER | Read-only | 0x0 |
| 0x30–0xE0 | Reserved Register | – | – | – |
| 0xE4 | Write Protect Mode Register | RTC_WPMR | Read-write | 0x00000000 |
| 0xE8–0xF8 | Reserved Register | – | – | – |
| 0xFC | Reserved Register | – | – | – |

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

### 15.6.1 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 “RTC Write Protect Mode Register” on page 241.

- **UPDTIM: Update Request Time Register**

0 = No effect.

1 = Stops the RTC time counting.

Time counting consists of second, minute and hour counters. Time counters can be programmed once this bit is set and acknowledged by the bit ACKUPD of the Status Register.

- **UPDCAL: Update Request Calendar Register**

0 = No effect.

1 = Stops the RTC calendar counting.

Calendar counting consists of day, date, month, year and century counters. Calendar counters can be programmed once this bit is set.

- **TIMEVSEL: Time Event Selection**

The event that generates the flag TIMEV in RTC_SR (Status Register) depends on the value of TIMEVSEL.

| Value | Name | Description |
|---|---|---|
| 0 | MINUTE | Minute change |
| 1 | HOUR | Hour change |
| 2 | MIDNIGHT | Every day at midnight |
| 3 | NOON | Every day at noon |

- **CALEVSEL: Calendar Event Selection**

The event that generates the flag CALEV in RTC_SR depends on the value of CALEVSEL

| Value | Name | Description |
|---|---|---|
| 0 | WEEK | Week change (every Monday at time 00:00:00) |
| 1 | MONTH | Month change (every 01 of each month at time 00:00:00) |
| 2 | YEAR | Year change (every January 1 at time 00:00:00) |
| 3 | – | |

### 15.6.2 RTC Mode Register

**Name:** RTC_MR

**Address:** 0x400E1464

**Access:** Read-write

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

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

0 = 24-hour mode is selected.

1 = 12-hour mode is selected.

All non-significant bits read zero.

### 15.6.3 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.

All non-significant bits read zero.

### 15.6.4 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 (BCD).

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

- **YEAR: Current Year**

The range that can be set is 00 - 99 (BCD).

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

- **MONTH: Current Month**

The range that can be set is 01 - 12 (BCD).

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

- **DAY: Current Day in Current Week**

The range that can be set is 1 - 7 (BCD).

The coding of the number (which number represents which day) is user-defined as it has no effect on the date counter.

- **DATE: Current Day in Current Month**

The range that can be set is 01 - 31 (BCD).

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

All non-significant bits read zero.

### 15.6.5 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 “RTC Write Protect Mode Register” on page 241.

- **SEC: Second Alarm**

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

- **SECEN: Second Alarm Enable**

0 = The second-matching alarm is disabled.

1 = The second-matching alarm is enabled.

- **MIN: Minute Alarm**

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

- **MINEN: Minute Alarm Enable**

0 = The minute-matching alarm is disabled.

1 = The minute-matching alarm is enabled.

- **HOUR: Hour Alarm**

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

- **AMPM: AM/PM Indicator**

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

- **HOUREN: Hour Alarm Enable**

0 = The hour-matching alarm is disabled.

1 = The hour-matching alarm is enabled.

### 15.6.6 RTC Calendar Alarm Register

**Name:** RTC_CALALR

**Address:** 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 “RTC Write Protect Mode Register” on page 241.

- **MONTH: Month Alarm**

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

- **MTHEN: Month Alarm Enable**

0 = The month-matching alarm is disabled.

1 = The month-matching alarm is enabled.

- **DATE: Date Alarm**

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

- **DATEEN: Date Alarm Enable**

0 = The date-matching alarm is disabled.

1 = The date-matching alarm is enabled.

### 15.6.7 RTC Status Register

**Name:** RTC_SR

**Address:** 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 |
| – | – | – | CALEV | TIMEV | SEC | ALARM | ACKUPD |

- **ACKUPD: Acknowledge for Update**

0 = Time and calendar registers cannot be updated.

1 = Time and calendar registers can be updated.

- **ALARM: Alarm Flag**

0 = No alarm matching condition occurred.

1 = An alarm matching condition has occurred.

- **SEC: Second Event**

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

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

- **TIMEV: Time Event**

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

1 = At least one time event has occurred since the last clear.

The time event is selected in the TIMEVSEL field in RTC_CR (Control Register) and can be any one of the following events: minute change, hour change, noon, midnight (day change).

- **CALEV: Calendar Event**

0 = No calendar event has occurred since the last clear.

1 = At least one calendar event has occurred since the last clear.

The calendar event is selected in the CALEVSEL field in RTC_CR and can be any one of the following events: week change, month change and year change.

### 15.6.8 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 |
| – | – | – | 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).

### 15.6.9 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 |
| – | – | – | 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.

### 15.6.10 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 |
| – | – | – | 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.

### 15.6.11 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 |
| – | – | – | 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.

### 15.6.12 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.

### 15.6.13 RTC Write Protect 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 Protect Enable**

0 = Disables the Write Protect if WPKEY corresponds to 0x525443 (“RTC” in ASCII).

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

Protects the registers:

“RTC Mode Register”

“RTC Mode Register”

“RTC Time Alarm Register”

“RTC Calendar Alarm Register”

# 16. Watchdog Timer (WDT)

## 16.1 Description

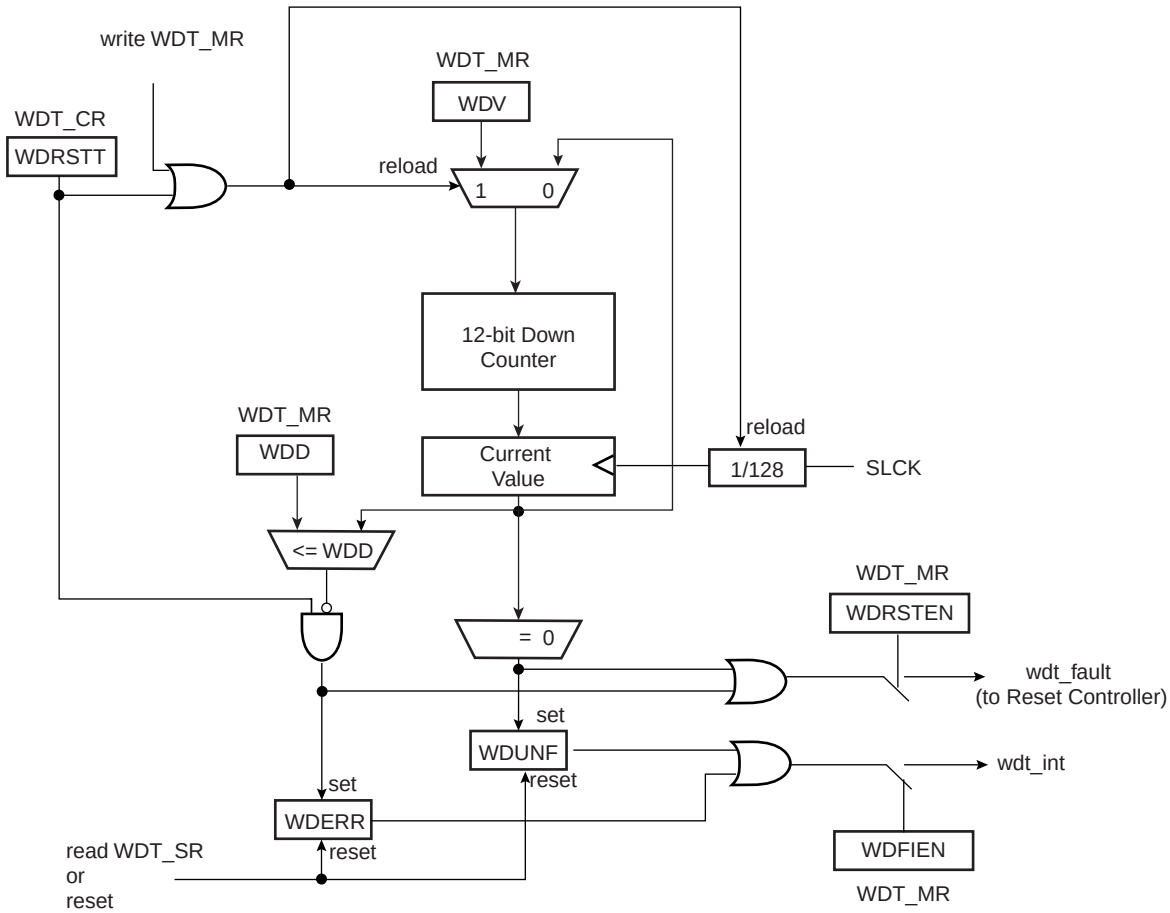
The Watchdog Timer can be used to prevent system lock-up if the software becomes trapped in a deadlock. It features a 12-bit down counter that allows a watchdog period of up to 16 seconds (slow clock at 32.768 kHz). It can generate a general reset or a processor reset only. In addition, it can be stopped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

## 16.2 Embedded Characteristics

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

## 16.3 Block Diagram

**Figure 16-1. Watchdog Timer Block Diagram**



## 16.4 Functional Description

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

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

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

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

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

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

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

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

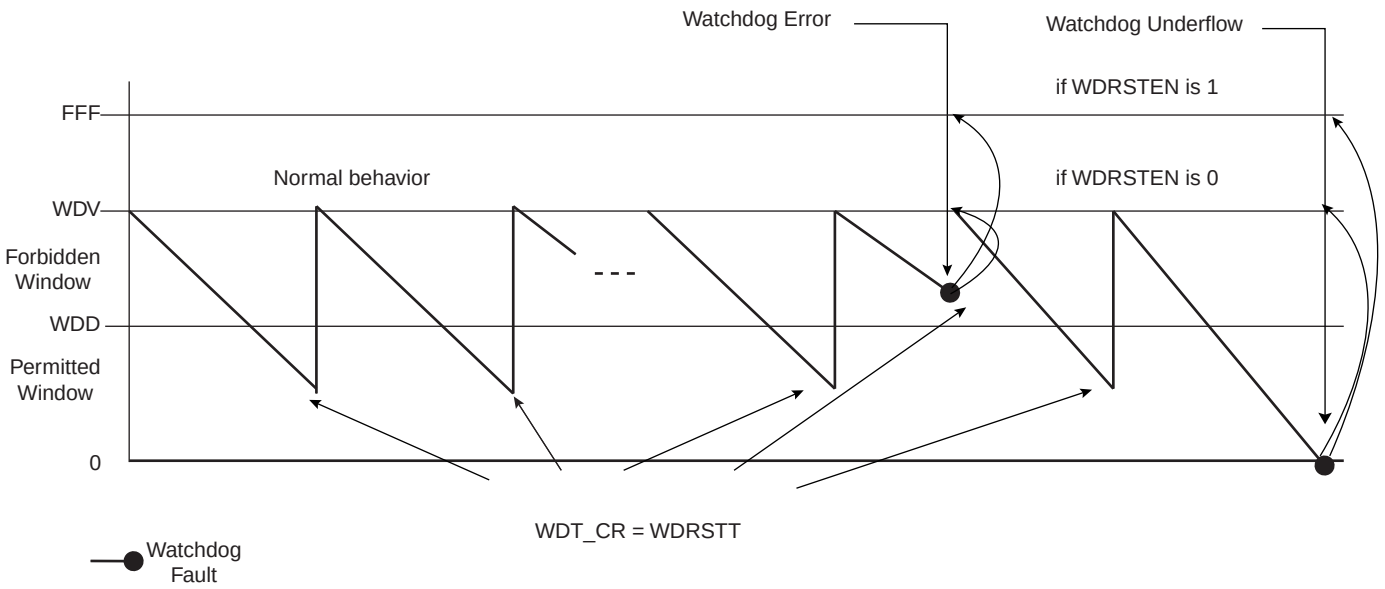
The status bits WDUNF (Watchdog Underflow) and WDERR (Watchdog Error) trigger an interrupt, provided the bit WDFIEN is set in the mode register. The signal “wdt_fault” to the reset controller causes a Watchdog reset if the WDRSTEN bit is set as already explained in the reset controller programmer Datasheet.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

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

Writing the WDT_MR reloads and restarts the down counter.

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

**Figure 16-2. Watchdog Behavior**



## 16.5 Watchdog Timer (WDT) User Interface

**Table 16-1. Register Mapping**

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

### 16.5.1 Watchdog Timer Control Register

**Name:** WDT_CR

**Address:** 0x400E1450

**Access:** Write-only

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

- **WDRSTT: Watchdog Restart**

0: No effect.

1: Restarts the Watchdog.

- **KEY: Password**

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

### 16.5.2 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 | | | | | | | |

- **WDV: Watchdog Counter Value**

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

- **WDFIEN: Watchdog Fault Interrupt Enable**

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

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

- **WDRSTEN: Watchdog Reset Enable**

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

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

- **WDRPROC: Watchdog Reset Processor**

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

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

- **WDD: Watchdog Delta Value**

Defines the permitted range for reloading the Watchdog Timer.

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

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

- **WDDBGHLT: Watchdog Debug Halt**

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

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

- **WDIDLEHLT: Watchdog Idle Halt**

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

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

- **WDDIS: Watchdog Disable**

0: Enables the Watchdog Timer.

1: Disables the Watchdog Timer.

### 16.5.3 Watchdog Timer Status Register

**Name:** WDT_SR

**Address:** 0x400E1458

**Access:** Read-only

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

- **WDUNF: Watchdog Underflow**

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

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

- **WDERR: Watchdog Error**

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

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

# 17. Supply Controller (SUPC)

## 17.1 Description

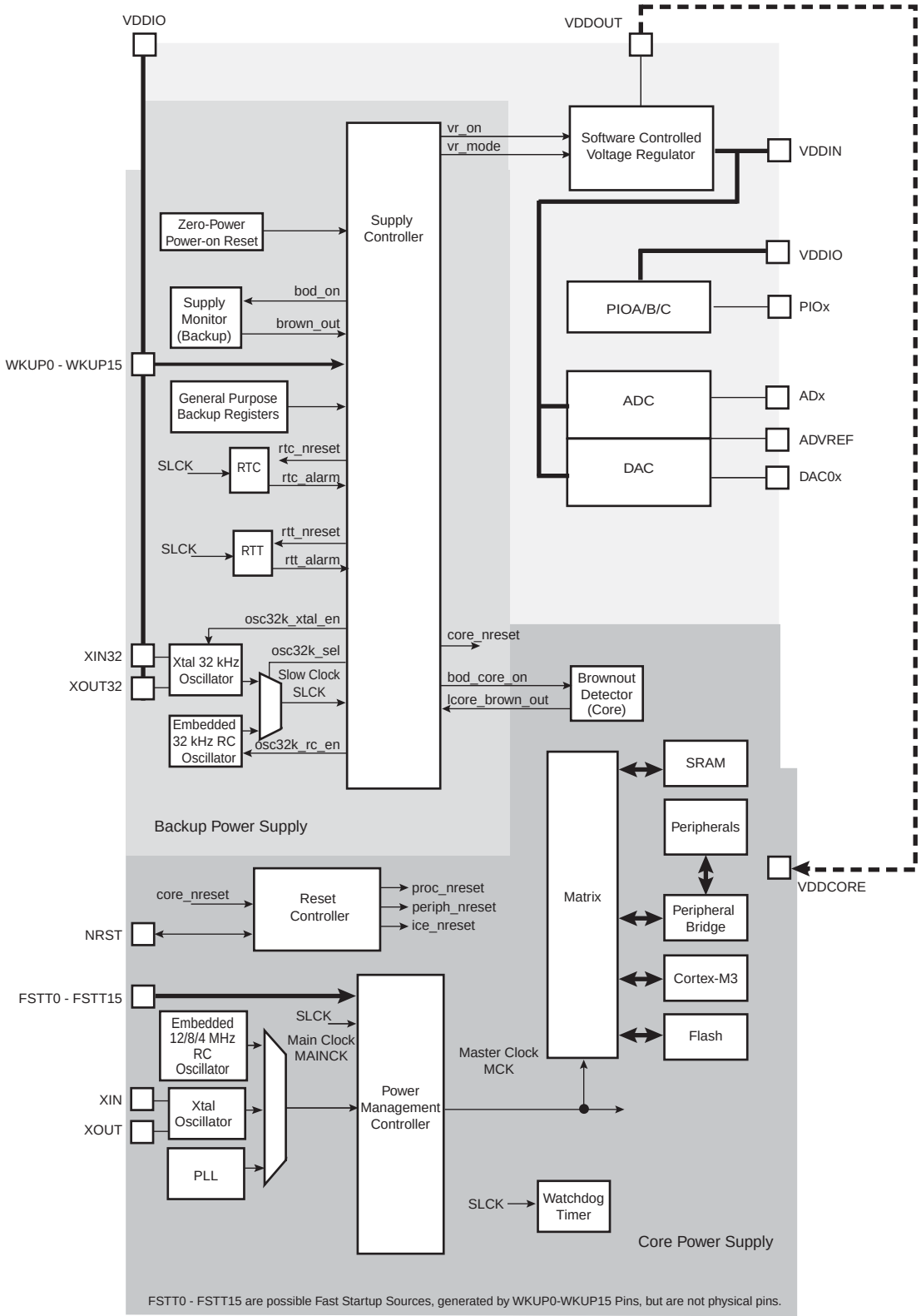
The Supply Controller (SUPC) controls the supply voltage of the Core of the system and manages the Backup Low Power Mode. In this mode, the current consumption is reduced to a few microamps for Backup power retention. Exit from this mode is possible on multiple wake-up sources including events on WKUP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

## 17.2 Embedded Characteristics

- Manages the Core Power Supply VDDCORE and the Backup Low Power Mode by Controlling the Embedded Voltage Regulator
- Generates the Slow Clock SLCK, by Selecting Either the 22-42 kHz Low Power RC Oscillator or the 32 kHz Low Power Crystal Oscillator
- Supports Multiple Wake Up Sources, for Exit from Backup Low Power Mode
  - Force Wake Up Pin, with Programmable Debouncing
  - 16 Wake Up Inputs, with Programmable Debouncing
  - Real Time Clock Alarm
  - Real Time Timer Alarm
  - Supply Monitor Detection on VDDIO, with Programmable Scan Period and Voltage Threshold
- A Supply Monitor Detection on VDDIO or a Brownout Detection on VDDCORE can Trigger a Core Reset
- Embeds:
  - One 22 to 42 kHz Low Power RC Oscillator
  - One 32 kHz Low Power Crystal Oscillator
  - One Zero-Power Power-On Reset Cell
  - One Software Programmable Supply Monitor, on VDDIO Located in Backup Section
  - One Brownout Detector on VDDCORE Located in the Core

## 17.3 Block Diagram

**Figure 17-1. Supply Controller Block Diagram**



## 17.4 Supply Controller Functional Description

### 17.4.1 Supply Controller Overview

The device can be divided into two power supply areas:

- The  VDDIO Power Supply: including the Supply Controller, a part of the Reset Controller, the Slow Clock switch, the General Purpose Backup Registers, the Supply Monitor and the Clock which includes the Real Time Timer and the Real Time Clock
- The Core Power Supply: including the other part of the Reset Controller, the Brownout Detector, the Processor, the SRAM memory, the FLASH memory and the Peripherals

The Supply Controller (SUPC) controls the supply voltage of the core power supply. The SUPC intervenes when the VDDIO power supply rises (when the system is starting) or when the Backup Low Power Mode is entered.

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

The Supply Controller and the 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  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 that the core voltage VDDCORE is valid, then releasing the reset signal of the core “vddcore_nreset” signal.

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

When the Backup Low Power Mode is entered, the SUPC sequentially asserts the reset signal of the core power supply “vddcore_nreset” and disables the voltage regulator, in order to supply only the 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 wake-up sources including an event on WKUP pins, or a Clock alarm. To exit this mode, the SUPC operates in the same way as system startup.

### 17.4.2 Slow Clock Generator

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

The user can select the crystal oscillator to be the source of the slow clock, as it provides a more accurate frequency. The command is made by writing the Supply Controller Control Register (SUPC_CR) with the XTALSEL bit at 1.This results in a sequence which first configures the PIO lines multiplexed with XIN32 and XOUT32 to be driven by the oscillator, then enables the crystal oscillator. then waits for 32,768 slow clock cycles, then switches the slow clock on the output of the crystal oscillator and then disables the RC oscillator to save power. The switch of the slow clock source is glitch free. The OSCSEL bit of the Supply Controller Status Register (SUPC_SR) allows knowing when the switch sequence is done.

Coming back on the RC oscillator is only possible by shutting down the 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 Supply Controller Mode Register (SUPC_MR) needs to be set at 1.

### 17.4.3 Voltage Regulator Control/Backup Low Power Mode

The Supply Controller can be used to control the embedded 1.8V voltage regulator.

The voltage regulator automatically adapts its quiescent current depending on the required load current. Please refer to the electrical characteristics section.

The programmer can switch off the voltage regulator, and thus put the device in Backup mode, by writing the Supply Controller Control Register (SUPC_CR) with the VROFF bit at 1.

This can be done also by using WFE (Wait for Event) Cortex-M3 instruction with the deep mode bit set to 1.

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

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

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

When the user does not use the internal voltage regulator and wants to supply VDDCORE by an external supply, it is possible to disable the voltage regulator. Note that it is different from the Backup mode. Depending on the application, disabling the voltage regulator can reduce power consumption as the voltage regulator input (VDDIN) is shared with the ADC and DAC. This is done through ONREG bit in SUPC_MR.

### 17.4.4 Supply Monitor

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

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

The threshold of the supply monitor is programmable. It can be selected from 1.9V to 3.4V by steps of 100 mV. This threshold is programmed in the SMTH field of the Supply Controller Supply Monitor Mode Register (SUPC_SMMR).

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

Enabling the supply monitor for such reduced times allows to divide the typical supply monitor power consumption respectively by factors of 32, 256 or 2048, if the user does not need a continuous monitoring of the VDDIO power supply.

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

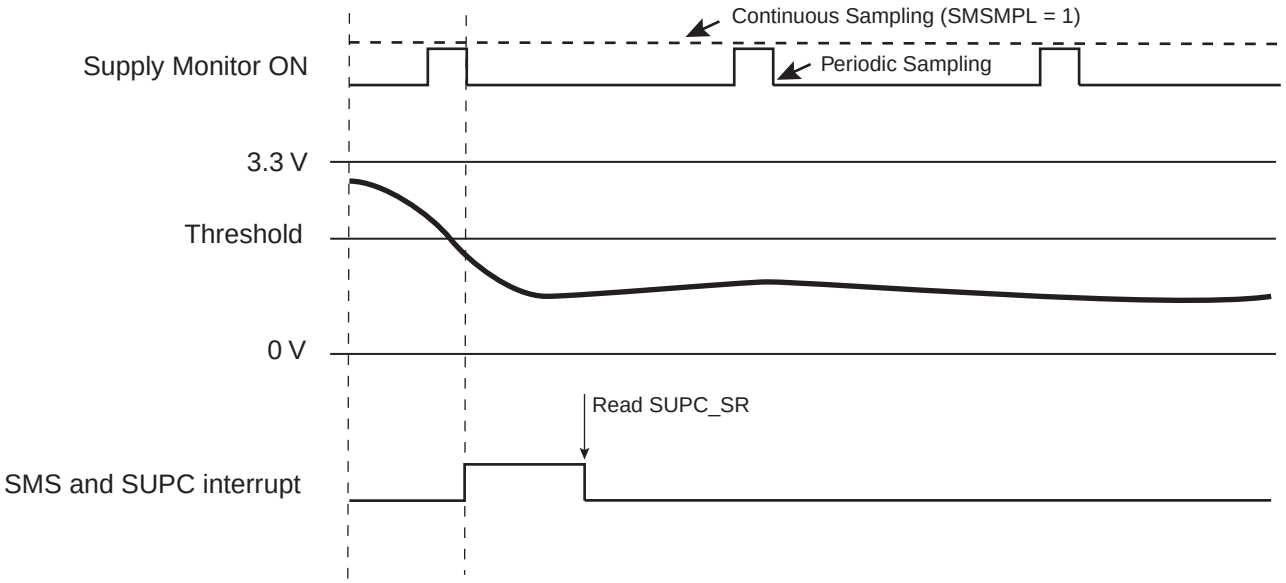
Waking up the core power supply when a supply monitor detection occurs can be enabled by programming the SMEN bit to 1 in the Supply Controller Wake Up Mode Register (SUPC_WUMR).

The Supply Controller provides two status bits in the Supply Controller Status Register for the supply monitor which allows to determine whether the last wake up was due to the supply monitor:

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

The SMS bit can generate an interrupt if the SMIEN bit is set to 1 in the Supply Controller Supply Monitor Mode Register (SUPC_SMMR).

**Figure 17-2. Supply Monitor Status Bit and Associated Interrupt**



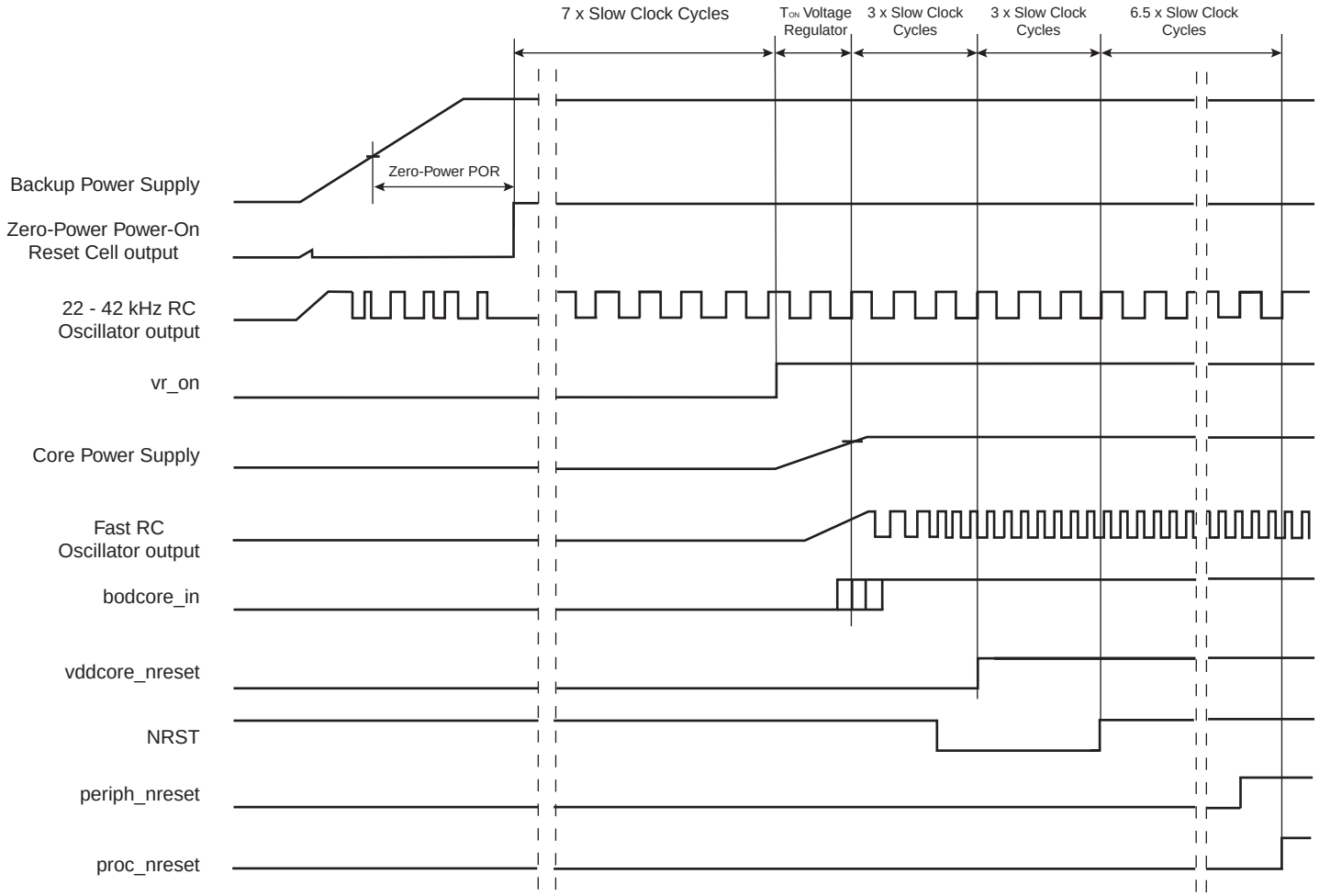
### 17.4.5 Power Supply Reset

#### 17.4.5.1 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 Supply Controller 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 brownout detector provides the bodcore_in signal as soon as the core voltage VDDCORE is valid. This results in releasing the vddcore_nreset signal to the Reset Controller after the bodcore_in signal has been confirmed as being valid for at least one slow clock cycle.

**Figure 17-3. Raising the VDDIO Power Supply**




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

### 17.4.6 Core Reset

The Supply Controller manages the vddcore_nreset signal to the Reset Controller, as described previously in Section 17.4.5 ”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:

- a supply monitor detection
- a brownout detection

#### 17.4.6.1 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 the Supply Controller Supply Monitor Mode Register (SUPC_SMMR).

If SMRSTEN is set and if a supply monitor detection occurs, the vddcore_nreset signal is immediately activated for a minimum of 1 slow clock cycle.

#### 17.4.6.2 Brownout Detector Reset

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

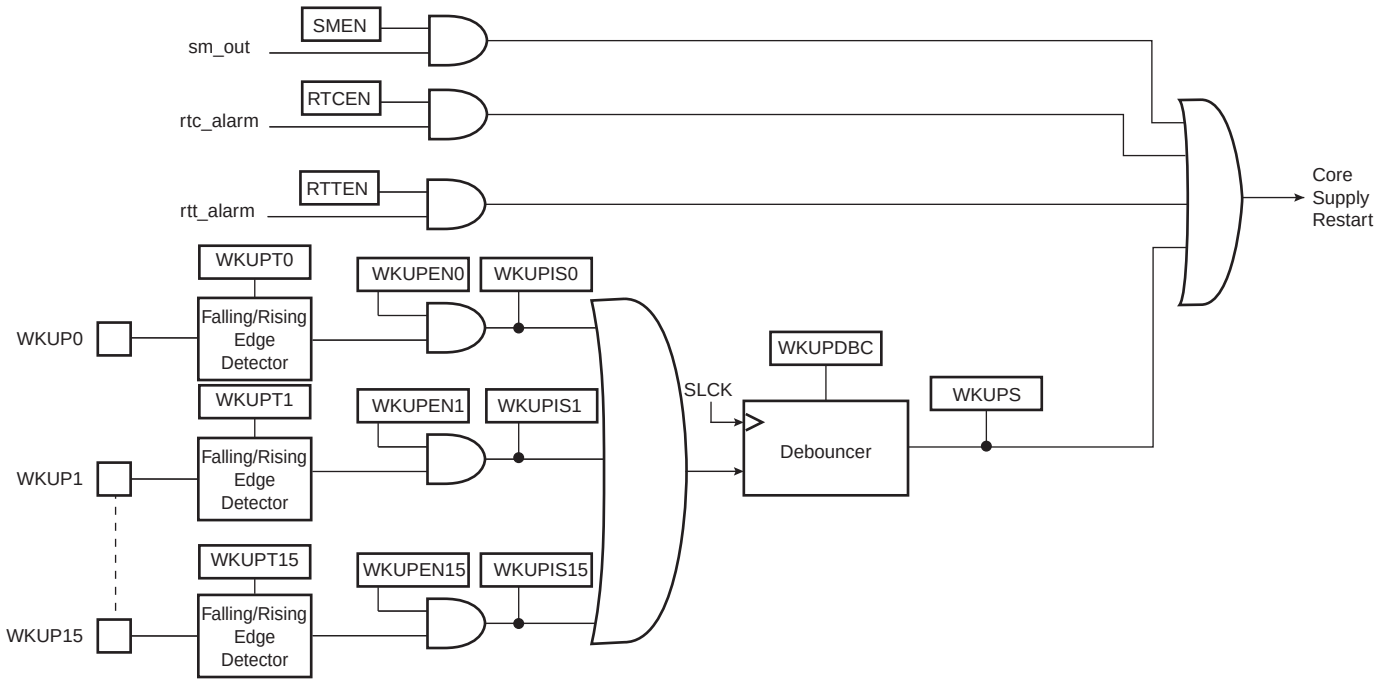
If BODRSTEN is set and the voltage regulation is lost (output voltage of the regulator too low), the vddcore_nreset signal is asserted for a minimum of 1 slow clock cycle and then released if bodcore_in has been reactivated. The BODRSTS bit is set in the Supply Controller Status Register (SUPC_SR) so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

Until bodcore_in is deactivated, the vddcore_nreset signal remains active.

### 17.4.7 Wake Up Sources

The wake up events allow the device to exit backup mode. When a wake up event is detected, the Supply Controller performs a sequence which automatically reenables the core power supply.

**Figure 17-4. Wake Up Sources**



#### 17.4.7.1 Wake Up Inputs

The wake up inputs, WKUP0 to WKUP15, can be programmed to perform a wake up of the core power supply. Each input can be enabled by writing to 1 the corresponding bit, WKUPEN0 to WKUPEN 15, in the Wake Up Inputs Register (SUPC_WUIR). The wake up level can be selected with the corresponding polarity bit, WKUPPL0 to WKUPPL15, also located in SUPC_WUIR.

All the resulting signals are wired-ORed to trigger a debounce counter, which can be programmed with the WKUPDBC field in the Supply Controller Wake Up Mode Register (SUPC_WUMR). The WKUPDBC field can select a debouncing period of 3, 32, 512, 4,096 or 32,768 slow clock cycles. This corresponds respectively to about 100 µs, about 1 ms, about 16 ms, about 128 ms and about 1 second (for a typical slow clock frequency of 32 kHz). Programming WKUPDBC to 0x0 selects an immediate wake up, i.e., an enabled WKUP pin must be active according to its polarity during a minimum of one slow clock period to wake up the core power supply.

If an enabled WKUP pin is asserted for a time longer than the debouncing period, a wake up of the core power supply is started and the signals, WKUP0 to WKUP15 as shown in Figure 17-4, are latched in the Supply Controller Status Register (SUPC_SR). This allows the user to identify the source of the wake up, however, if a new wake up condition occurs, the primary information is lost. No new wake up can be detected since the primary wake up condition has disappeared.

#### 17.4.7.2 Clock Alarms

The RTC and the RTT alarms can generate a wake up of the core power supply. This can be enabled by writing respectively, the bits RTCEN and RTTEN to 1 in the Supply Controller Wake Up Mode Register (SUPC_WUMR).

The Supply Controller does not provide any status as the information is available in the User Interface of either the Real Time Timer or the Real Time Clock.

#### 17.4.7.3 Supply Monitor Detection

The supply monitor can generate a wakeup of the core power supply. See Section 17.4.4 ”Supply Monitor”.

## 17.5 Supply Controller (SUPC) User Interface

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

### 17.5.1 System Controller (SYSC) User Interface

**Table 17-1. System Controller Registers**

| Offset | System Controller Peripheral | Name |
|---|---|---|
| 0x00-0x0c | Reset Controller | RSTC |
| 0x10-0x2C | Supply Controller | SUPC |
| 0x30-0x3C | Real Time Timer | RTT |
| 0x50-0x5C | Watchdog | WDT |
| 0x60-0x7C | Real Time Clock | RTC |
| 0x90-0xDC | General Purpose Backup Register | GPBR |

### 17.5.2 Supply Controller (SUPC) User Interface

**Table 17-2. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x00 | Supply Controller Control Register | SUPC_CR | Write-only | N/A |
| 0x04 | Supply Controller Supply Monitor Mode Register | SUPC_SMMR | Read-write | 0x0000_0000 |
| 0x08 | Supply Controller Mode Register | SUPC_MR | Read-write | 0x0000_5A00 |
| 0x0C | Supply Controller Wake Up Mode Register | SUPC_WUMR | Read-write | 0x0000_0000 |
| 0x10 | Supply Controller Wake Up Inputs Register | SUPC_WUIR | Read-write | 0x0000_0000 |
| 0x14 | Supply Controller Status Register | SUPC_SR | Read-only | 0x0000_0800 |
| 0x18 | Reserved | | | |

### 17.5.3 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 | – | – |

- **VROFF: Voltage Regulator Off**

0 (NO_EFFECT) = no effect.

1 (STOP_VREG) = if KEY is correct, asserts vddcore_nreset and stops the voltage regulator.

- **XTALSEL: Crystal Oscillator Select**

0 (NO_EFFECT) = no effect.

1 (CRYSTAL_SEL) = if KEY is correct, switches the slow clock on the crystal oscillator output.

- **KEY: Password**

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

### 17.5.4 Supply Controller Supply Monitor Mode Register

**Name:** SUPC_SMMR

**Address:** 0x400E1414

**Access:** Read-write

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

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | SMIEN | SMRSTEN | – | SMSMPL | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | SMTH | | | |

- **SMTH: Supply Monitor Threshold**

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

- **SMSMPL: Supply Monitor Sampling Period**

| Value | Name | Description |
|---|---|---|
| 0x0 | SMD | Supply Monitor disabled |
| 0x1 | CSM | Continuous Supply Monitor |
| 0x2 | 32SLCK | Supply Monitor enabled one SLCK period every 32 SLCK periods |
| 0x3 | 256SLCK | Supply Monitor enabled one SLCK period every 256 SLCK periods |
| 0x4 | 2048SLCK | Supply Monitor enabled one SLCK period every 2,048 SLCK periods |
| 0x5-0x7 | Reserved | Reserved |

- **SMRSTEN: Supply Monitor Reset Enable**

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

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

- **SMIEN: Supply Monitor Interrupt Enable**

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

1 (ENABLE) = the SUPC interrupt signal is asserted when a supply monitor detection occurs.

### 17.5.5 Supply Controller Mode Register

**Name:** SUPC_MR

**Address:** 0x400E1418

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| KEY | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | – | – | OSCBYPASS | – | – | – | – |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | ONREG | BODDIS | BODRSTEN | – | – | – | – |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

- **BODRSTEN: Brownout Detector Reset Enable**

0 (NOT_ENABLE) = the core reset signal “vddcore_nreset” is not affected when a brownout detection occurs.

1 (ENABLE) = the core reset signal, vddcore_nreset is asserted when a brownout detection occurs.

- **BODDIS: Brownout Detector Disable**

0 (ENABLE) = the core brownout detector is enabled.

1 (DISABLE) = the core brownout detector is disabled.

- **ONREG: Voltage Regulator enable**

0 (ONREG_UNUSED) = Voltage Regulator is not used

1 (ONREG_USED) = Voltage Regulator is used

- **OSCBYPASS: Oscillator Bypass**

0 (NO_EFFECT) = no effect. Clock selection depends on XTALSEL value.

1 (BYPASS) = the 32-KHz XTAL oscillator is selected and is put in bypass mode.

- **KEY: Password Key**

Should be written to value 0xA5. Writing any other value in this field aborts the write operation.

### 17.5.6 Supply Controller Wake Up 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 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | WKUPDBC | | | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | RTCEN | RTTEN | SMEN | – |

- **SMEN: Supply Monitor Wake Up Enable**

0 (NOT_ENABLE) = the supply monitor detection has no wake up effect.

1 (ENABLE) = the supply monitor detection forces the wake up of the core power supply.

- **RTTEN: Real Time Timer Wake Up Enable**

0 (NOT_ENABLE) = the RTT alarm signal has no wake up effect.

1 (ENABLE) = the RTT alarm signal forces the wake up of the core power supply.

- **RTCEN: Real Time Clock Wake Up Enable**

0 (NOT_ENABLE) = the RTC alarm signal has no wake up effect.

1 (ENABLE) = the RTC alarm signal forces the wake up of the core power supply.

- **WKUPDBC: Wake Up Inputs Debouncer Period**

| Value | Name | Description |
|---|---|---|
| 0 | IMMEDIATE | Immediate, no debouncing, detected active at least on one Slow Clock edge. |
| 1 | 3_SCLK | WKUPx shall be in its active state for at least 3 SLCK periods |
| 2 | 32_SCLK | WKUPx shall be in its active state for at least 32 SLCK periods |
| 3 | 512_SCLK | WKUPx shall be in its active state for at least 512 SLCK periods |
| 4 | 4096_SCLK | WKUPx shall be in its active state for at least 4,096 SLCK periods |
| 5 | 32768_SCLK | WKUPx shall be in its active state for at least 32,768 SLCK periods |
| 6 | Reserved | Reserved |
| 7 | Reserved | Reserved |

### 17.5.7 System Controller Wake Up Inputs Register

**Name:** SUPC_WUIR

**Address:** 0x400E1420

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WKUPT15 | WKUPT14 | WKUPT13 | WKUPT12 | WKUPT11 | WKUPT10 | WKUPT9 | WKUPT8 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| WKUPT7 | WKUPT6 | WKUPT5 | WKUPT4 | WKUPT3 | WKUPT2 | WKUPT1 | WKUPT0 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| WKUPEN15 | WKUPEN14 | WKUPEN13 | WKUPEN12 | WKUPEN11 | WKUPEN10 | WKUPEN9 | WKUPEN8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| WKUPEN7 | WKUPEN6 | WKUPEN5 | WKUPEN4 | WKUPEN3 | WKUPEN2 | WKUPEN1 | WKUPEN0 |

- **WKUPEN0 - WKUPEN15: Wake Up Input Enable 0 to 15**

0 (NOT_ENABLE) = the corresponding wake-up input has no wake up effect.

1 (ENABLE) = the corresponding wake-up input forces the wake up of the core power supply.

- **WKUPT0 - WKUPT15: Wake Up Input Transition 0 to 15**

0 (HIGH_TO_LOW) = a high to low level transition on the corresponding wake-up input forces the wake up of the core power supply.

1 (LOW_TO_HIGH) = a low to high level transition on the corresponding wake-up input forces the wake up of the core power supply.

### 17.5.8 Supply Controller Status Register

**Name:** SUPC_SR

**Address:** 0x400E1424

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WKUPIS15 | WKUPIS14 | WKUPIS13 | WKUPIS12 | WKUPIS11 | WKUPIS10 | WKUPIS9 | WKUPIS8 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| WKUPIS7 | WKUPIS6 | WKUPIS5 | WKUPIS4 | WKUPIS3 | WKUPIS2 | WKUPIS1 | WKUPIS0 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| OSCSEL | SMOS | SMS | SMRSTS | BODRSTS | SMWS | WKUPS | – |

Note: Because of the asynchronism between the Slow Clock (SCLK) and the System Clock (MCK), the status register flag reset is taken into account only 2 slow clock cycles after the read of the SUPC_SR.

- **WKUPS: WKUP Wake Up Status**

0 (NO) = no wake up due to the assertion of the WKUP pins has occurred since the last read of SUPC_SR.

1 (PRESENT) = at least one wake up due to the assertion of the WKUP pins has occurred since the last read of SUPC_SR.

- **SMWS: Supply Monitor Detection Wake Up Status**

0 (NO) = no wake up due to a supply monitor detection has occurred since the last read of SUPC_SR.

1 (PRESENT) = at least one wake up due to a supply monitor detection has occurred since the last read of SUPC_SR.

- **BODRSTS: Brownout Detector Reset Status**

0 (NO) = no core brownout rising edge event has been detected since the last read of the SUPC_SR.

1 (PRESENT) = at least one brownout output rising edge event has been detected since the last read of the SUPC_SR.

When the voltage remains below the defined threshold, there is no rising edge event at the output of the brownout detection cell. The rising edge event occurs only when there is a voltage transition below the threshold.

- **SMRSTS: Supply Monitor Reset Status**

0 (NO) = no supply monitor detection has generated a core reset since the last read of the SUPC_SR.

1 (PRESENT) = at least one supply monitor detection has generated a core reset since the last read of the SUPC_SR.

- **SMS: Supply Monitor Status**

0 (NO) = no supply monitor detection since the last read of SUPC_SR.

1 (PRESENT) = at least one supply monitor detection since the last read of SUPC_SR.

- **SMOS: Supply Monitor Output Status**

0 (HIGH) = the supply monitor detected VDDIO higher than its threshold at its last measurement.

1 (LOW) = the supply monitor detected VDDIO lower than its threshold at its last measurement.

# 18. General Purpose Backup Registers (GPBR)

## 18.1 Description

The System Controller embeds Eight general-purpose backup registers.

## 18.2 Embedded Characteristics

Eight 32-bit General Purpose Backup Registers

## 18.3 General Purpose Backup Registers (GPBR) User Interface

**Table 18-1. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x0 | General Purpose Backup Register 0 | SYS_GPBR0 | Read-write | – |
| ... | ... | ... | ... | ... |
| 0x1C | General Purpose Backup Register 7 | SYS_GPBR7 | Read-write | – |

### 18.3.1 General Purpose Backup Register x

**Name:** SYS_GPBRx

**Addresses:** 0x400E1490 [0] .. 0x400E149C [3]

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| GPBR_VALUEx | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| GPBR_VALUEx | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| GPBR_VALUEx | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| GPBR_VALUEx | | | | | | | |

- **GPBR_VALUEx: Value of GPBR x**

# 19. Enhanced Embedded Flash Controller (EEFC)

## 19.1 Description

The Enhanced Embedded Flash Controller (EEFC) ensures the interface of the Flash block with the 32-bit internal bus.

Its 128-bit or 64-bit wide memory interface increases performance. It also manages the programming, erasing, locking and unlocking sequences of the Flash using a full set of commands. One of the commands returns the embedded Flash descriptor definition that informs the system about the Flash organization, thus making the software generic.

## 19.2 Product Dependencies

### 19.2.1 Power Management

The Enhanced Embedded Flash Controller (EEFC) is continuously clocked. The Power Management Controller has no effect on its behavior.

### 19.2.2 Interrupt Sources

The Enhanced Embedded Flash Controller (EEFC) interrupt line is connected to the Nested Vectored Interrupt Controller (NVIC). Using the Enhanced Embedded Flash Controller (EEFC) interrupt requires the NVIC to be programmed first. The EEFC interrupt is generated only on FRDY bit rising.

**Table 19-1. Peripheral IDs**

| Instance | ID |
|---|---|
| EFC | 6 |

## 19.3 Functional Description

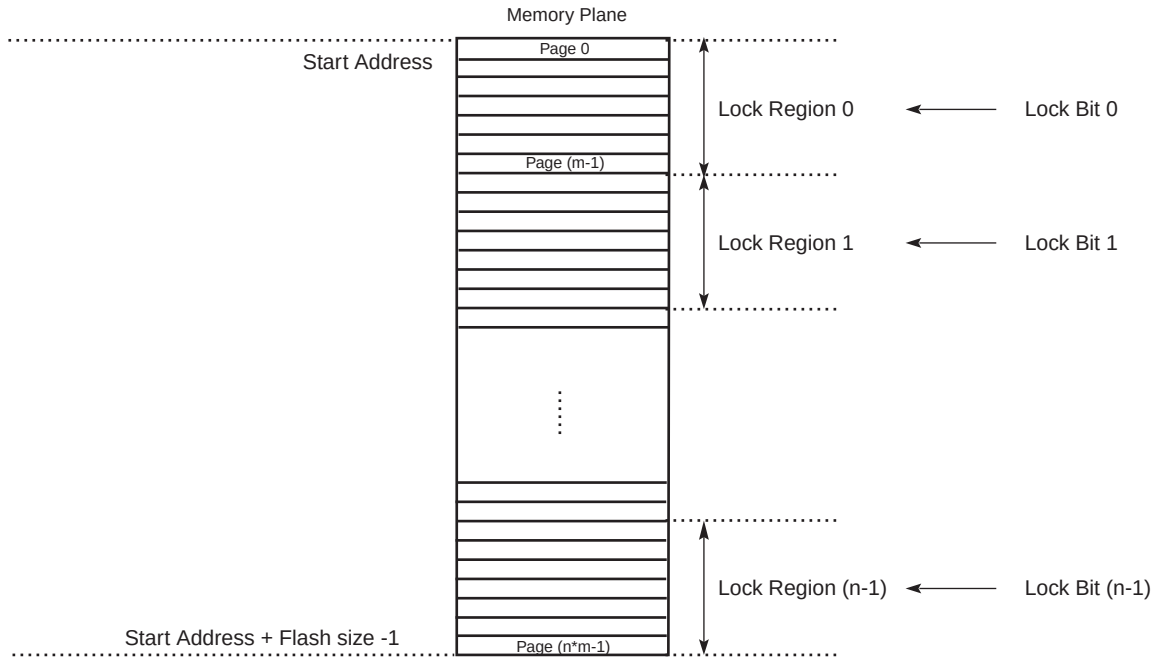
### 19.3.1 Embedded Flash Organization

The embedded Flash interfaces directly with the 32-bit internal bus. The embedded Flash is composed of:

- One memory plane organized in several pages of the same size.
- Two 128-bit or 64-bit read buffers used for code read optimization.
- One 128-bit or 64-bit read buffer used for data read optimization.
- One write buffer that manages page programming. The write buffer size is equal to the page size. This buffer is write-only and accessible all along the 1 MByte address space, so that each word can be written to its final address.
- Several lock bits used to protect write/erase operation on several pages (lock region). A lock bit is associated with a lock region composed of several pages in the memory plane.
- Several bits that may be set and cleared through the Enhanced Embedded Flash Controller (EEFC) interface, called General Purpose Non Volatile Memory bits (GPNVM bits).

The embedded Flash size, the page size, the lock regions organization and GPNVM bits definition are described in the product definition section. The Enhanced Embedded Flash Controller (EEFC) returns a descriptor of the Flash controlled after a get descriptor command issued by the application (see “Getting Embedded Flash Descriptor” on page 273).

**Figure 19-1. Embedded Flash Organization**



### 19.3.2 Read Operations

An optimized controller manages embedded Flash reads, thus increasing performance when the processor is running in Thumb2 mode by means of the 128- or 64- bit wide memory interface.

The Flash memory is accessible through 8-, 16- and 32-bit reads.

As the Flash block size is smaller than the address space reserved for the internal memory area, the embedded Flash wraps around the address space and appears to be repeated within it.

The read operations can be performed with or without wait states. Wait states must be programmed in the field FWS (Flash Read Wait State) in the Flash Mode Register (EEFC_FMR). Defining FWS to be 0 enables the single-cycle access of the embedded Flash. Refer to the Electrical Characteristics for more details.

#### 19.3.2.1 128-bit or 64-bit Access Mode

By default the read accesses of the Flash are performed through a 128-bit wide memory interface. It enables better system performance especially when 2 or 3 wait state needed.

For systems requiring only 1 wait state, or to privilege current consumption rather than performance, the user can select a 64-bit wide memory access via the FAM bit in the Flash Mode Register (EEFC_FMR)

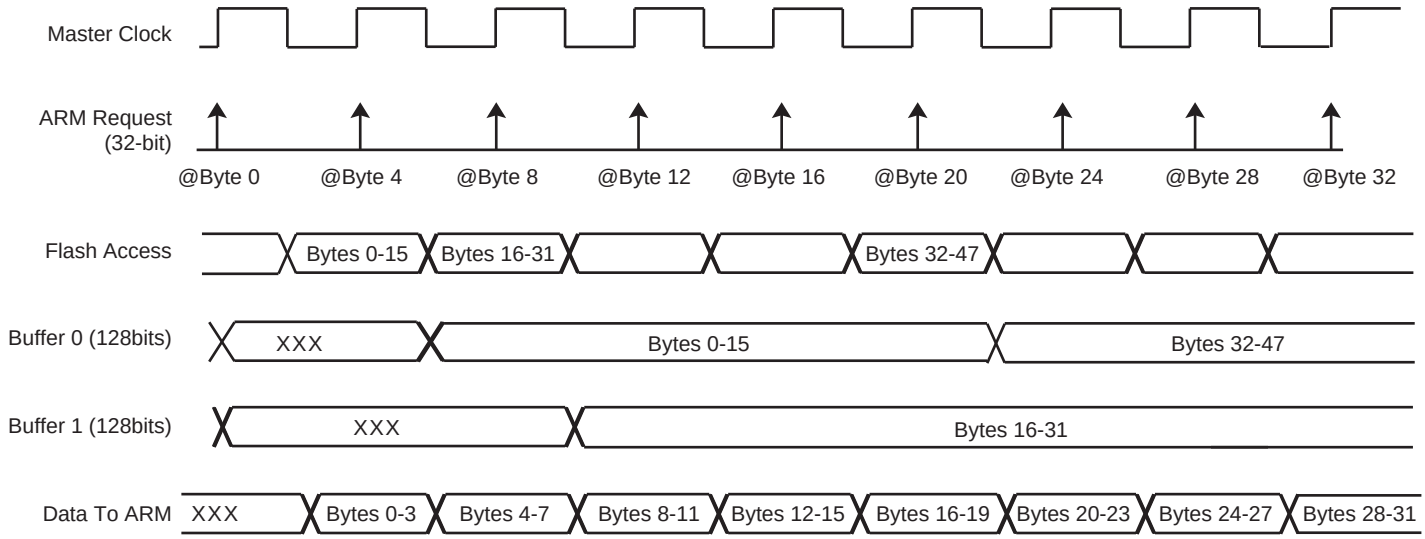
Please refer to the electrical characteristics section of the product datasheet for more details.

#### 19.3.2.2 Code Read Optimization

A system of 2 x 128-bit or 2 x 64-bit buffers is added in order to optimize sequential Code Fetch.

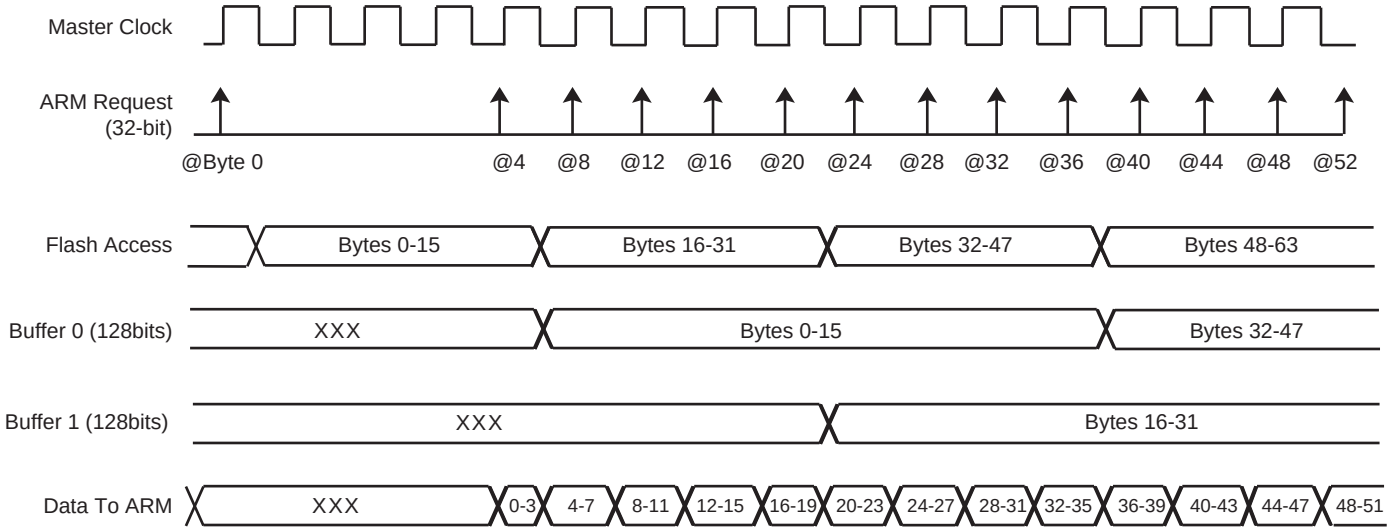
Note: Immediate consecutive code read accesses are not mandatory to benefit from this optimization.

**Figure 19-2. Code Read Optimization for FWS = 0**




Note: When FWS is equal to 0, all the accesses are performed in a single-cycle access.

**Figure 19-3. Code Read Optimization for FWS = 3**



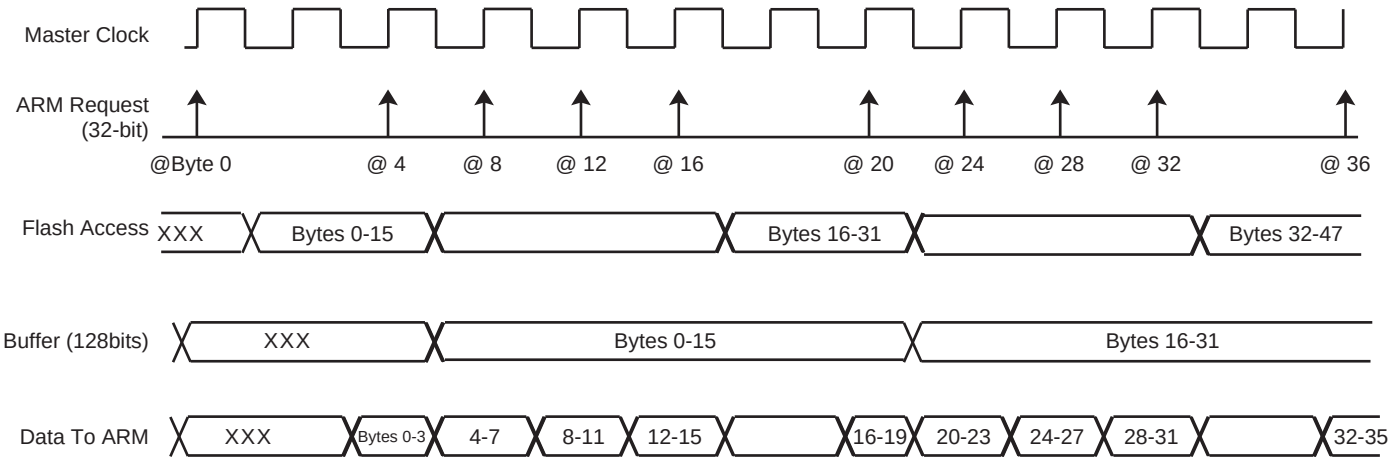

Note: When FWS is included between 1 and 3, in case of sequential reads, the first access takes (FWS+1) cycles, the other ones only 1 cycle.

### 19.3.2.3 Data Read Optimization

The organization of the Flash in 128 bits (or 64 bits) is associated with two 128-bit (or 64-bit) prefetch buffers and one 128-bit (or 64-bit) data read buffer, thus providing maximum system performance. This buffer is added in order to store the requested data plus all the data contained in the 128-bit (64-bit) aligned data. This speeds up sequential data reads if, for example, FWS is equal to 1 (see Figure 19-4).

Note: No consecutive data read accesses are mandatory to benefit from this optimization.

**Figure 19-4. Data Read Optimization for FWS = 1**



### 19.3.3 Flash Commands

The Enhanced Embedded Flash Controller (EEFC) offers a set of commands such as programming the memory Flash, locking and unlocking lock regions, consecutive programming and locking and full Flash erasing, etc.

Commands and read operations can be performed in parallel only on different memory planes. Code can be fetched from one memory plane while a write or an erase operation is performed on another.

**Table 19-2. Set of Commands**

| Command | Value | Mnemonic |
|---|---|---|
| Get Flash Descriptor | 0x00 | GETD |
| Write page | 0x01 | WP |
| Write page and lock | 0x02 | WPL |
| Erase page and write page | 0x03 | EWP |
| Erase page and write page then lock | 0x04 | EWPL |
| Erase all | 0x05 | EA |
| Set Lock Bit | 0x08 | SLB |
| Clear Lock Bit | 0x09 | CLB |
| Get Lock Bit | 0x0A | GLB |
| Set GPNVM Bit | 0x0B | SGPB |
| Clear GPNVM Bit | 0x0C | CGPB |
| Get GPNVM Bit | 0x0D | GGPB |
| Start Read Unique Identifier | 0x0E | STUI |
| Stop Read Unique Identifier | 0x0F | SPUI |
| Get CALIB Bit | 0x10 | GCALB |

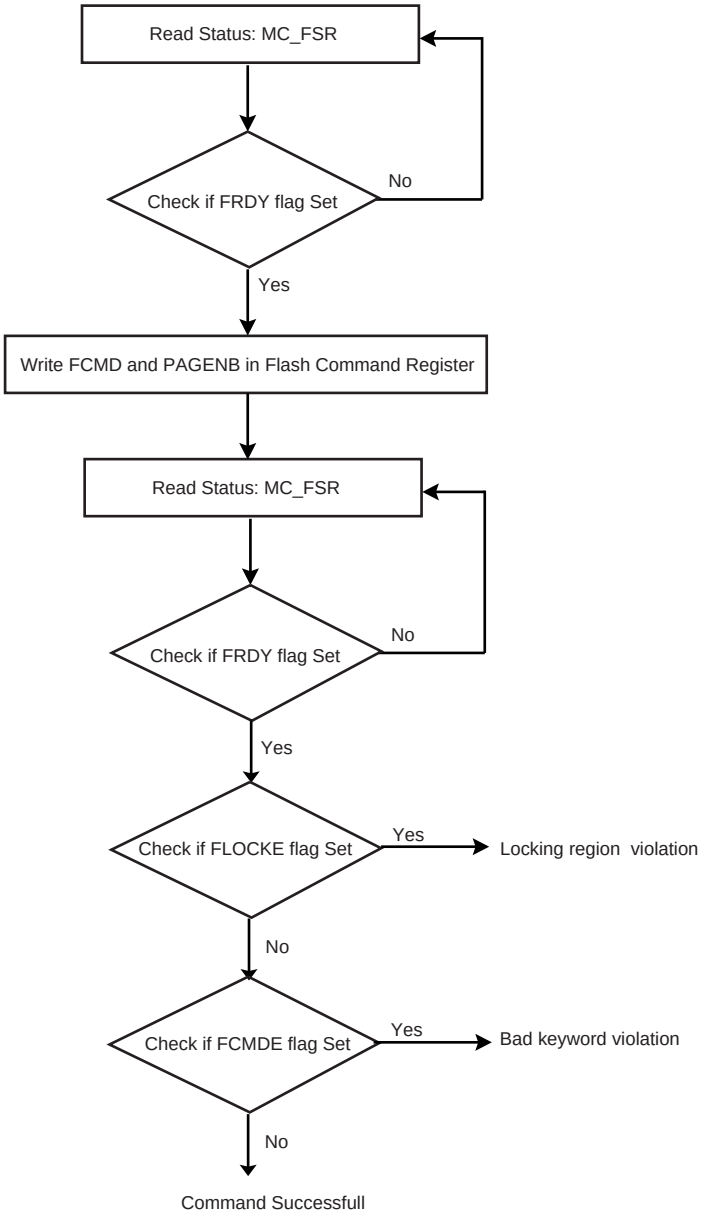
In order to perform one of these commands, the Flash Command Register (EEFC_FCR) has to be written with the correct command using the FCMD field. As soon as the EEFC_FCR register is written, **the FRDY flag and the FVALUE field in the EEFC_FRR register are automatically cleared**. Once the current command is achieved, then the FRDY flag is automatically set. If an interrupt has been enabled by setting the FRDY bit in EEFC_FMR, the corresponding interrupt line of the NVIC is activated. (Note that this is true for all commands except for the STUI Command. The FRDY flag is not set when the STUI command is achieved.)

All the commands are protected by the same keyword, which has to be written in the 8 highest bits of the EEFC_FCR register.

Writing EEFC_FCR with data that does not contain the correct key and/or with an invalid command has no effect on the whole memory plane, but the FCMDE flag is set in the EEFC_FSR register. This flag is automatically cleared by a read access to the EEFC_FSR register.

When the current command writes or erases a page in a locked region, the command has no effect on the whole memory plane, but the FLOCKE flag is set in the EEFC_FSR register. This flag is automatically cleared by a read access to the EEFC_FSR register.

**Figure 19-5. Command State Chart**




### 19.3.3.1 Getting Embedded Flash Descriptor

This command allows the system to learn about the Flash organization. The system can take full advantage of this information. For instance, a device could be replaced by one with more Flash capacity, and so the software is able to adapt itself to the new configuration.

To get the embedded Flash descriptor, the application writes the GETD command in the EEFC_FCR register. The first word of the descriptor can be read by the software application in the EEFC_FRR register as soon as the FRDY flag in the EEFC_FSR register rises. The next reads of the EEFC_FRR register provide the following word of the descriptor. If extra read operations to the EEFC_FRR register are done after the last word of the descriptor has been returned, then the EEFC_FRR register value is 0 until the next valid command.

**Table 19-3. Flash Descriptor Definition**

| Symbol | Word Index | Description |
|---|---|---|
| FL_ID | 0 | Flash Interface Description |
| FL_SIZE | 1 | Flash size in bytes |
| FL_PAGE_SIZE | 2 | Page size in bytes |
| FL_NB_PLANE | 3 | Number of planes. |
| FL_PLANE[0] | 4 | Number of bytes in the first plane. |
| ... | | |
| FL_PLANE[FL_NB_PLANE-1] | 4 + FL_NB_PLANE - 1 | Number of bytes in the last plane. |
| FL_NB_LOCK | 4 + FL_NB_PLANE | Number of lock bits. A bit is associated with a lock region. A lock bit is used to prevent write or erase operations in the lock region. |
| FL_LOCK[0] | 4 + FL_NB_PLANE + 1 | Number of bytes in the first lock region. |
| ... | | |

#### 19.3.3.2 Write Commands

Several commands can be used to program the Flash.

Flash technology requires that an erase is done before programming. The full memory plane can be erased at the same time, or several pages can be erased at the same time (refer to Section "The Partial Programming mode works only with 128-bit (or higher) boundaries. It cannot be used with boundaries lower than 128 bits (8, 16 or 32-bit for example)."). Also, a page erase can be automatically done before a page write using EWP or EWPL commands.

After programming, the page (the whole lock region) can be locked to prevent miscellaneous write or erase sequences. The lock bit can be automatically set after page programming using WPL or EWPL commands.

Data to be written are stored in an internal latch buffer. The size of the latch buffer corresponds to the page size. The latch buffer wraps around within the internal memory area address space and is repeated as many times as the number of pages within this address space.

Note: Writing of 8-bit and 16-bit data is not allowed and may lead to unpredictable data corruption.

Write operations are performed in a number of wait states equal to the number of wait states for read operations.

Data are written to the latch buffer before the programming command is written to the Flash Command Register EEFC_FCR. The sequence is as follows:

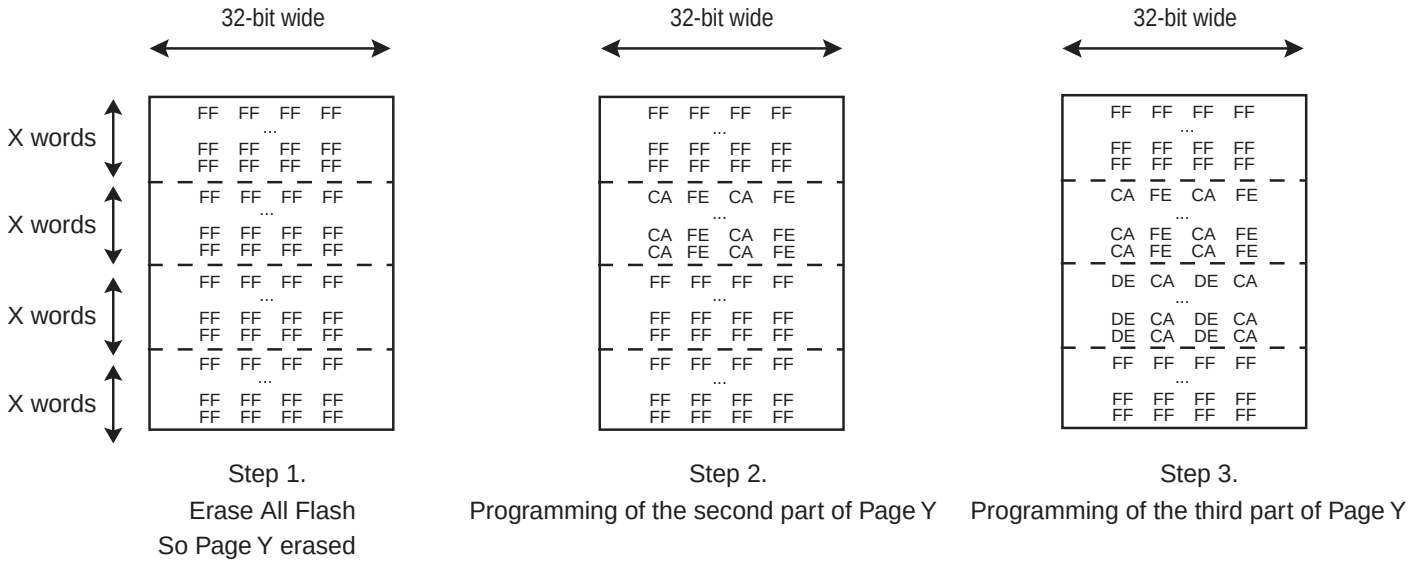
- Write the full page, at any page address, within the internal memory area address space.
- Programming starts as soon as the page number and the programming command are written to the Flash Command Register. The FRDY bit in the Flash Programming Status Register (EEFC_FSR) is automatically cleared.
- When programming is completed, the FRDY bit in the Flash Programming Status Register (EEFC_FSR) rises. If an interrupt has been enabled by setting the bit FRDY in EEFC_FMR, the corresponding interrupt line of the NVIC is activated.

Two errors can be detected in the EEFC_FSR register after a programming sequence:

- a Command Error: a bad keyword has been written in the EEFC_FCR register.
- a Lock Error: the page to be programmed belongs to a locked region. A command must be previously run to unlock the corresponding region.

By using the WP command, a page can be programmed in several steps if it has been erased before (see Figure 19-6).

**Figure 19-6. Example of Partial Page Programming**




The Partial Programming mode works only with 128-bit (or higher) boundaries. It cannot be used with boundaries lower than 128 bits (8, 16 or 32-bit for example).

#### 19.3.3.3 Erase Commands

Erase commands are allowed only on unlocked regions.

The erase sequence is:

- Erase starts as soon as one of the erase commands and the FARG field are written in the Flash Command Register.
- When the programming completes, the FRDY bit in the Flash Programming Status Register (EEFC_FSR) rises. If an interrupt has been enabled by setting the FRDY bit in EEFC_FMR, the interrupt line of the NVIC is activated.

Two errors can be detected in the EEFC_FSR register after a programming sequence:

- a Command Error: a bad keyword has been written in the EEFC_FCR register.
- a Lock Error: at least one page to be erased belongs to a locked region. The erase command has been refused, no page has been erased. A command must be run previously to unlock the corresponding region.

#### 19.3.3.4 Lock Bit Protection

Lock bits are associated with several pages in the embedded Flash memory plane. This defines lock regions in the embedded Flash memory plane. They prevent writing/erasing protected pages.

The lock sequence is:

- The Set Lock command (SLB) and a page number to be protected are written in the Flash Command Register.
- When the locking completes, the FRDY bit in the Flash Programming Status Register (EEFC_FSR) rises. If an interrupt has been enabled by setting the FRDY bit in EEFC_FMR, the interrupt line of the NVIC is activated.
- If the lock bit number is greater than the total number of lock bits, then the command has no effect. The result of the SLB command can be checked running a GLB (Get Lock Bit) command.

One error can be detected in the EEFC_FSR register after a programming sequence:

- a Command Error: a bad keyword has been written in the EEFC_FCR register.

It is possible to clear lock bits previously set. Then the locked region can be erased or programmed. The unlock sequence is:

- The Clear Lock command (CLB) and a page number to be unprotected are written in the Flash Command Register.
- When the unlock completes, the FRDY bit in the Flash Programming Status Register (EEFC_FSR) rises. If an interrupt has been enabled by setting the FRDY bit in EEFC_FMR, the interrupt line of the NVIC is activated.
- If the lock bit number is greater than the total number of lock bits, then the command has no effect.

One error can be detected in the EEFC_FSR register after a programming sequence:

- a Command Error: a bad keyword has been written in the EEFC_FCR register.

The status of lock bits can be returned by the Enhanced Embedded Flash Controller (EEFC). The Get Lock Bit status sequence is:

- The Get Lock Bit command (GLB) is written in the Flash Command Register, FARG field is meaningless.
- Lock bits can be read by the software application in the EEFC_FRR register. The first word read corresponds to the 32 first lock bits, next reads providing the next 32 lock bits as long as it is meaningful. Extra reads to the EEFC_FRR register return 0.

For example, if the third bit of the first word read in the EEFC_FRR is set, then the third lock region is locked.

One error can be detected in the EEFC_FSR register after a programming sequence:

- a Command Error: a bad keyword has been written in the EEFC_FCR register.

Note: Access to the Flash in read is permitted when a set, clear or get lock bit command is performed.

#### 19.3.3.5 GPNVM Bit

GPNVM bits do not interfere with the embedded Flash memory plane. Refer to the product definition section for information on the GPNVM Bit Action.

The set GPNVM bit sequence is:

- Start the Set GPNVM Bit command (SGPB) by writing the Flash Command Register with the SGPB command and the number of the GPNVM bit to be set.
- When the GPVNM bit is set, the bit FRDY in the Flash Programming Status Register (EEFC_FSR) rises. If an interrupt was enabled by setting the FRDY bit in EEFC_FMR, the interrupt line of the NVIC is activated.
- If the GPNVM bit number is greater than the total number of GPNVM bits, then the command has no effect. The result of the SGPB command can be checked by running a GGPB (Get GPNVM Bit) command.

One error can be detected in the EEFC_FSR register after a programming sequence:

- A Command Error: a bad keyword has been written in the EEFC_FCR register.

It is possible to clear GPNVM bits previously set. The clear GPNVM bit sequence is:

- Start the Clear GPNVM Bit command (CGPB) by writing the Flash Command Register with CGPB and the number of the GPNVM bit to be cleared.
- When the clear completes, the FRDY bit in the Flash Programming Status Register (EEFC_FSR) rises. If an interrupt has been enabled by setting the FRDY bit in EEFC_FMR, the interrupt line of the NVIC is activated.
- If the GPNVM bit number is greater than the total number of GPNVM bits, then the command has no effect.

One error can be detected in the EEFC_FSR register after a programming sequence:

- A Command Error: a bad keyword has been written in the EEFC_FCR register.

The status of GPNVM bits can be returned by the Enhanced Embedded Flash Controller (EEFC). The sequence is:

- Start the Get GPNVM bit command by writing the Flash Command Register with GGPB. The FARG field is meaningless.
- GPNVM bits can be read by the software application in the EEFC_FRR register. The first word read corresponds to the 32 first GPNVM bits, following reads provide the next 32 GPNVM bits as long as it is meaningful. Extra reads to the EEFC_FRR register return 0.

For example, if the third bit of the first word read in the EEFC_FRR is set, then the third GPNVM bit is active.

One error can be detected in the EEFC_FSR register after a programming sequence:

- a Command Error: a bad keyword has been written in the EEFC_FCR register.

Note: Access to the Flash in read is permitted when a set, clear or get GPNVM bit command is performed.

#### 19.3.3.6 Calibration Bit

Calibration bits do not interfere with the embedded Flash memory plane.

It is impossible to modify the calibration bits.

The status of calibration bits can be returned by the Enhanced Embedded Flash Controller (EEFC). The sequence is:

- Issue the Get CALIB Bit command by writing the Flash Command Register with GCALB (see Table 19-2). The FARG field is meaningless.
- Calibration bits can be read by the software application in the EEFC_FRR register. The first word read corresponds to the 32 first calibration bits, following reads provide the next 32 calibration bits as long as it is meaningful. Extra reads to the EEFC_FRR register return 0.

The 4/8/12 MHz Fast RC oscillator is calibrated in production. This calibration can be read through the Get CALIB Bit command. The table below shows the bit implementation for each frequency:

| RC Calibration Frequency | EEFC_FRR Bits |
|---|---|
| 8 MHz output | [28 - 22] |
| 12 MHz output | [38 - 32] |

The RC calibration for 4 MHz is set to 1,000,000.

#### 19.3.3.7 Security Bit Protection

When the security is enabled, access to the Flash, either through the JTAG/SWD interface or through the Fast Flash Programming Interface, is forbidden. This ensures the confidentiality of the code programmed in the Flash.

The security bit is GPNVM0.

Disabling the security bit can only be achieved by asserting the ERASE pin at 1, and after a full Flash erase is performed. When the security bit is deactivated, all accesses to the Flash are permitted.

#### 19.3.3.8 Unique Identifier

Each part is programmed with a 128-bit Unique Identifier. It can be used to generate keys for example.

To read the Unique Identifier the sequence is:

- Send the Start Read unique Identifier command (STUI) by writing the Flash Command Register with the STUI command.
- When the Unique Identifier is ready to be read, the FRDY bit in the Flash Programming Status Register (EEFC_FSR) falls.

- The Unique Identifier is located in the first 128 bits of the Flash memory mapping. So, at the address 0x80000-0x8000F.
- To stop the Unique Identifier mode, the user needs to send the Stop Read unique Identifier command (SPUI) by writing the Flash Command Register with the SPUI command.
- When the Stop read Unique Identifier command (SPUI) has been performed, the FRDY bit in the Flash Programming Status Register (EEFC_FSR) rises. If an interrupt was enabled by setting the FRDY bit in EEFC_FMR, the interrupt line of the NVIC is activated.

Note that during the sequence, the software can not run out of Flash (or the second plane in case of dual plane).

## 19.4 Enhanced Embedded Flash Controller (EEFC) User Interface

The User Interface of the Enhanced Embedded Flash Controller (EEFC) is integrated within the System Controller with base address 0x400E0800.

**Table 19-4. Register Mapping**

| Offset | Register | Name | Access | Reset State |
|---|---|---|---|---|
| 0x00 | EEFC Flash Mode Register | EEFC_FMR | Read-write | 0x0 |
| 0x04 | EEFC Flash Command Register | EEFC_FCR | Write-only | – |
| 0x08 | EEFC Flash Status Register | EEFC_FSR | Read-only | 0x00000001 |
| 0x0C | EEFC Flash Result Register | EEFC_FRR | Read-only | 0x0 |
| 0x10 | Reserved | – | – | – |

### 19.4.1 EEFC Flash Mode Register

**Name:** EEFC_FMR

**Address:** 0x400E0A00

**Access:** Read-write

**Offset**: 0x00

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | FAM |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | FWS | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | | – | – | – | – | – | FRDY |

- **FRDY: Ready Interrupt Enable**

0: Flash Ready does not generate an interrupt.

1: Flash Ready (to accept a new command) generates an interrupt.

- **FWS: Flash Wait State**

This field defines the number of wait states for read and write operations:

Number of cycles for Read/Write operations = FWS+1

- **FAM: Flash Access Mode**

0: 128-bit access in read Mode only, to enhance access speed.

1: 64-bit access in read Mode only, to enhance power consumption.

No Flash read should be done during change of this register.

### 19.4.2 EEFC Flash Command Register

**Name:** EEFC_FCR

**Address:** 0x400E0A04

**Access:** Write-only

**Offset**: 0x04

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| FKEY | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| FARG | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| FARG | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| FCMD | | | | | | | |

- **FCMD: Flash Command**

This field defines the flash commands. Refer to “Flash Commands” on page 272.

- **FARG: Flash Command Argument**

| | |
|---|---|
| Erase command | For erase all command, this field is meaningless. |
| Programming command | FARG defines the page number to be programmed. |
| Lock command | FARG defines the page number to be locked. |
| GPNVM command | FARG defines the GPNVM number. |
| Get commands | Field is meaningless. |
| Unique Identifier commands | Field is meaningless. |

- **FKEY: Flash Writing Protection Key**

This field should be written with the value 0x5A to enable the command defined by the bits of the register. If the field is written with a different value, the write is not performed and no action is started.

### 19.4.3 EEFC Flash Status Register

**Name:** EEFC_FSR

**Address:** 0x400E0A08

**Access:** Read-only

**Offset**: 0x08

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | FLOCKE | FCMDE | FRDY |

- **FRDY: Flash Ready Status**

0: The Enhanced Embedded Flash Controller (EEFC) is busy.

1: The Enhanced Embedded Flash Controller (EEFC) is ready to start a new command.

When it is set, this flags triggers an interrupt if the FRDY flag is set in the EEFC_FMR register.

This flag is automatically cleared when the Enhanced Embedded Flash Controller (EEFC) is busy.

- **FCMDE: Flash Command Error Status**

0: No invalid commands and no bad keywords were written in the Flash Mode Register EEFC_FMR.

1: An invalid command and/or a bad keyword was/were written in the Flash Mode Register EEFC_FMR.

This flag is automatically cleared when EEFC_FSR is read or EEFC_FCR is written.

- **FLOCKE: Flash Lock Error Status**

0: No programming/erase of at least one locked region has happened since the last read of EEFC_FSR.

1: Programming/erase of at least one locked region has happened since the last read of EEFC_FSR.

This flag is automatically cleared when EEFC_FSR is read or EEFC_FCR is written.

### 19.4.4 EEFC Flash Result Register

**Name:** EEFC_FRR

**Address:** 0x400E0A0C

**Access:** Read-only

**Offset**: 0x0C

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| FVALUE | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| FVALUE | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| FVALUE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| FVALUE | | | | | | | |

- **FVALUE: Flash Result Value**

The result of a Flash command is returned in this register. If the size of the result is greater than 32 bits, then the next resulting value is accessible at the next register read.

# 20. Fast Flash Programming Interface (FFPI)

## 20.1 Description

The Fast Flash Programming Interface provides parallel high-volume programming using a standard gang programmer. The parallel interface is fully handshaked and the device is considered to be a standard EEPROM. Additionally, the parallel protocol offers an optimized access to all the embedded Flash functionalities.

Although the Fast Flash Programming Mode is a dedicated mode for high volume programming, this mode is not designed for in-situ programming.

## 20.2 Parallel Fast Flash Programming

### 20.2.1 Device Configuration

In Fast Flash Programming Mode, the device is in a specific test mode. Only a certain set of pins is significant. The rest of the PIOs are used as inputs with a pull-up. The crystal oscillator is in bypass mode. Other pins must be left unconnected.

**Figure 20-1. SAM3NxA (48 bits) Parallel Programming Interface**

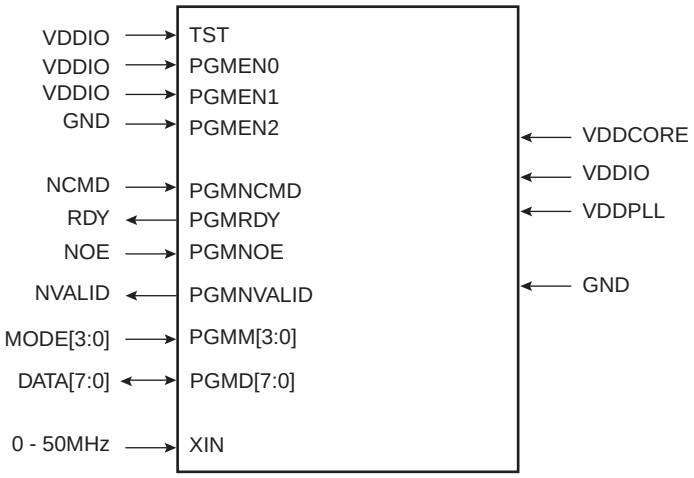



**Figure 20-2. SAM3NxB/C (64/100 pins) Parallel Programming Interface**

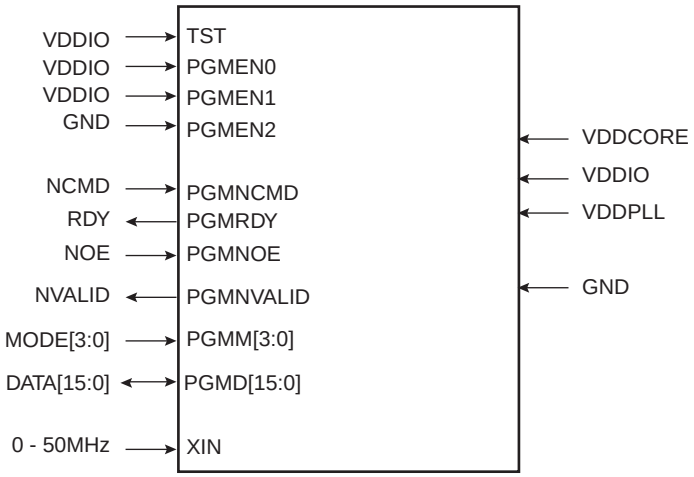


**Table 20-1. Signal Description List**

| Signal Name | Function | Type | Active Level | Comments |
|---|---|---|---|---|
| **Power** | | | | |
| VDDIO | I/O Lines Power Supply | Power | | |
| VDDCORE | Core Power Supply | Power | | |
| VDDPLL | PLL Power Supply | Power | | |
| GND | Ground | Ground | | |
| **Clocks** | | | | |
| XIN | Main Clock Input | Input | | 32KHz to 50MHz |
| **Test** | | | | |
| TST | Test Mode Select | Input | High | Must be connected to VDDIO |
| PGMEN0 | Test Mode Select | Input | High | Must be connected to VDDIO |
| PGMEN1 | Test Mode Select | Input | High | Must be connected to VDDIO |
| PGMEN2 | Test Mode Select | Input | Low | Must be connected to GND |
| **PIO** | | | | |
| PGMNCMD | Valid command available | Input | Low | Pulled-up input at reset |
| PGMRDY | 0: Device is busy<br>1: Device is ready for a new command | Output | High | Pulled-up input at reset |
| PGMNOE | Output Enable (active high) | Input | Low | Pulled-up input at reset |
| PGMNVALID | 0: DATA[15:0] or DATA[7:0](1) is in input mode<br>1: DATA[15:0] or DATA[7:0](1) is in output mode | Output | Low | Pulled-up input at reset |
| PGMM[3:0] | Specifies DATA type (See Table 20-2) | Input | | Pulled-up input at reset |
| PGMD[15:0] or [7:0](2) | Bi-directional data bus | Input/Output | | Pulled-up input at reset |

Notes: 1. DATA[7:0] pertains to the SAM3NxA (48 bits).
2. PGMD[7:0] pertains to the SAM3NxA (48 bits).

### 20.2.2 Signal Names

Depending on the MODE settings, DATA is latched in different internal registers.

**Table 20-2. Mode Coding**

| MODE[3:0] | Symbol | Data |
|---|---|---|
| 0000 | CMDE | Command Register |
| 0001 | ADDR0 | Address Register LSBs |
| 0010 | ADDR1 | |
| 0011 | ADDR2 | |
| 0100 | ADDR3 | Address Register MSBs |
| 0101 | DATA | Data Register |
| Default | IDLE | No register |

When MODE is equal to CMDE, then a new command (strobed on DATA[15:0] or DATA[7:0] signals) is stored in the command register.

Note: DATA[7:0] pertains to SAM3NxA (48 pins).

**Table 20-3. Command Bit Coding**

| DATA[15:0] | Symbol | Command Executed |
|---|---|---|
| 0x0011 | READ | Read Flash |
| 0x0012 | WP | Write Page Flash |
| 0x0022 | WPL | Write Page and Lock Flash |
| 0x0032 | EWP | Erase Page and Write Page |
| 0x0042 | EWPL | Erase Page and Write Page then Lock |
| 0x0013 | EA | Erase All |
| 0x0014 | SLB | Set Lock Bit |
| 0x0024 | CLB | Clear Lock Bit |
| 0x0015 | GLB | Get Lock Bit |
| 0x0034 | SGPB | Set General Purpose NVM bit |
| 0x0044 | CGPB | Clear General Purpose NVM bit |
| 0x0025 | GGPB | Get General Purpose NVM bit |
| 0x0054 | SSE | Set Security Bit |
| 0x0035 | GSE | Get Security Bit |
| 0x001F | WRAM | Write Memory |
| 0x001E | GVE | Get Version |

### 20.2.3 Entering Programming Mode

The following algorithm puts the device in Parallel Programming Mode:

- Apply GND, VDDIO, VDDCORE and VDDPLL.
- Apply XIN clock within $T_{POR\_RESET}$ if an external clock is available.
- Wait for $T_{POR\_RESET}$
- Start a read or write handshaking.

### 20.2.4 Programmer Handshaking

An handshake is defined for read and write operations. When the device is ready to start a new operation (RDY signal set), the programmer starts the handshake by clearing the NCMD signal. The handshaking is achieved once NCMD signal is high and RDY is high.

#### 20.2.4.1 Write Handshaking

For details on the write handshaking sequence, refer to Figure 20-3 Figure 20-4 and Table 20-4.

**Figure 20-3. SAM3NxB/C (64/100 pins) Parallel Programming Timing, Write Sequence**

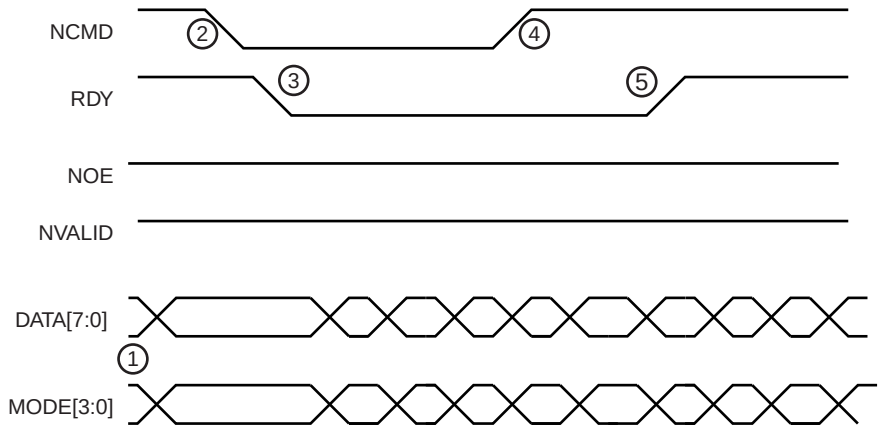



**Figure 20-4. SAM3NxA (48 pins) Parallel Programming Timing, Write Sequence**

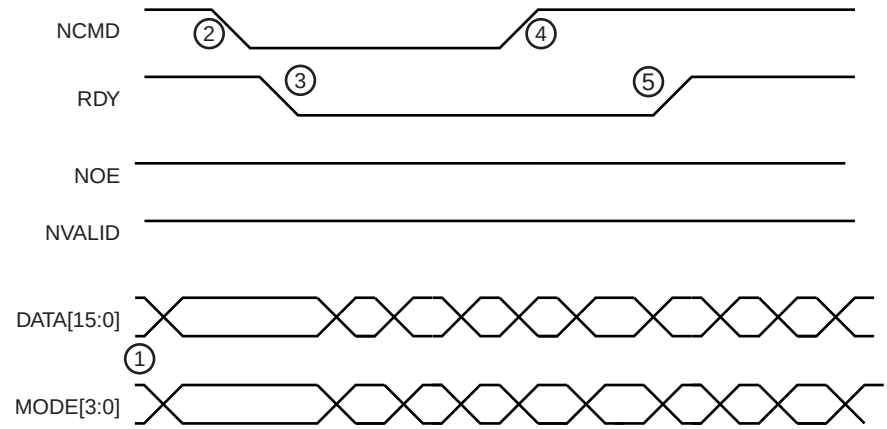


**Table 20-4. Write Handshake**

| Step | Programmer Action | Device Action | Data I/O |
|---|---|---|---|
| 1 | Sets MODE and DATA signals | Waits for NCMD low | Input |
| 2 | Clears NCMD signal | Latches MODE and DATA | Input |
| 3 | Waits for RDY low | Clears RDY signal | Input |
| 4 | Releases MODE and DATA signals | Executes command and polls NCMD high | Input |
| 5 | Sets NCMD signal | Executes command and polls NCMD high | Input |
| 6 | Waits for RDY high | Sets RDY | Input |

#### 20.2.4.2 Read Handshaking

For details on the read handshaking sequence, refer toFigure 20-5 Figure 20-6 and Table 20-5.

**Figure 20-5. SAM3NxB/C (64/100 pins) Parallel Programming Timing, Read Sequence**

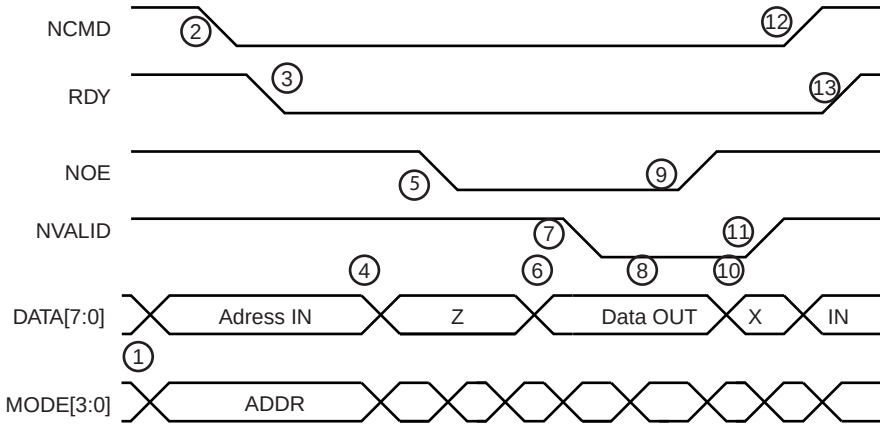



**Figure 20-6. SAM3NxA (48 pins) Parallel Programming Timing, Read Sequence**

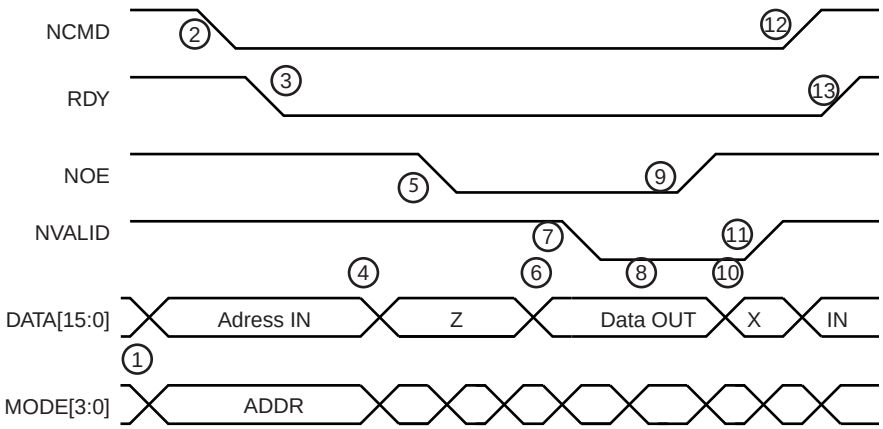


**Table 20-5. Read Handshake**

| Step | Programmer Action | Device Action | DATA I/O |
|---|---|---|---|
| 1 | Sets MODE and DATA signals | Waits for NCMD low | Input |
| 2 | Clears NCMD signal | Latch MODE and DATA | Input |
| 3 | Waits for RDY low | Clears RDY signal | Input |
| 4 | Sets DATA signal in tristate | Waits for NOE Low | Input |
| 5 | Clears NOE signal | | Tristate |
| 6 | Waits for NVALID low | Sets DATA bus in output mode and outputs the flash contents. | Output |
| 7 | | Clears NVALID signal | Output |
| 8 | Reads value on DATA Bus | Waits for NOE high | Output |
| 9 | Sets NOE signal | | Output |
| 10 | Waits for NVALID high | Sets DATA bus in input mode | X |
| 11 | Sets DATA in output mode | Sets NVALID signal | Input |
| 12 | Sets NCMD signal | Waits for NCMD high | Input |
| 13 | Waits for RDY high | Sets RDY signal | Input |

### 20.2.5 Device Operations

Several commands on the Flash memory are available. These commands are summarized in Table 20-3 on page 286. Each command is driven by the programmer through the parallel interface running several read/write handshaking sequences.

When a new command is executed, the previous one is automatically achieved. Thus, chaining a read command after a write automatically flushes the load buffer in the Flash.

In the following tables, Table 20-6 through Table 20-17

- **DATA[15:0] pertains to ASAM3NxB/C (64/100 pins)**
- **DATA[7:0] pertains to SAM3BxA (48 pins)**

#### 20.2.5.1 Flash Read Command

This command is used to read the contents of the Flash memory. The read command can start at any valid address in the memory plane and is optimized for consecutive reads. Read handshaking can be chained; an internal address buffer is automatically increased.

**Table 20-6. Read Command**

| Step | Handshake Sequence | MODE[3:0] | DATA[15:0] |
|---|---|---|---|
| 1 | Write handshaking | CMDE | READ |
| 2 | Write handshaking | ADDR0 | Memory Address LSB |
| 3 | Write handshaking | ADDR1 | Memory Address |
| 4 | Read handshaking | DATA | *Memory Address++ |
| 5 | Read handshaking | DATA | *Memory Address++ |
| ... | ... | ... | ... |
| n | Write handshaking | ADDR0 | Memory Address LSB |
| n+1 | Write handshaking | ADDR1 | Memory Address |
| n+2 | Read handshaking | DATA | *Memory Address++ |
| n+3 | Read handshaking | DATA | *Memory Address++ |
| ... | ... | ... | ... |

**Table 20-7. Read Command**

| Step | Handshake Sequence | MODE[3:0] | DATA[7:0] |
|---|---|---|---|
| 1 | Write handshaking | CMDE | READ |
| 2 | Write handshaking | ADDR0 | Memory Address LSB |
| 3 | Write handshaking | ADDR1 | Memory Address |
| 4 | Write handshaking | ADDR2 | Memory Address |
| 5 | Write handshaking | ADDR3 | Memory Address |
| 6 | Read handshaking | DATA | *Memory Address++ |
| 7 | Read handshaking | DATA | *Memory Address++ |
| ... | ... | ... | ... |
| n | Write handshaking | ADDR0 | Memory Address LSB |
| n+1 | Write handshaking | ADDR1 | Memory Address |
| n+2 | Write handshaking | ADDR2 | Memory Address |
| n+3 | Write handshaking | ADDR3 | Memory Address |
| n+4 | Read handshaking | DATA | *Memory Address++ |
| n+5 | Read handshaking | DATA | *Memory Address++ |
| ... | ... | ... | ... |

#### 20.2.5.2 Flash Write Command

This command is used to write the Flash contents.

The Flash memory plane is organized into several pages. Data to be written are stored in a load buffer that corresponds to a Flash memory page. The load buffer is automatically flushed to the Flash:

- before access to any page other than the current one
- when a new command is validated (MODE = CMDE)

The **Write Page** command **(WP)** is optimized for consecutive writes. Write handshaking can be chained; an internal address buffer is automatically increased.

**Table 20-8. Write Command**

| Step | Handshake Sequence | MODE[3:0] | DATA[15:0] |
|---|---|---|---|
| 1 | Write handshaking | CMDE | WP or WPL or EWP or EWPL |
| 2 | Write handshaking | ADDR0 | Memory Address LSB |
| 3 | Write handshaking | ADDR1 | Memory Address |
| 4 | Write handshaking | DATA | *Memory Address++ |
| 5 | Write handshaking | DATA | *Memory Address++ |
| ... | ... | ... | ... |
| n | Write handshaking | ADDR0 | Memory Address LSB |
| n+1 | Write handshaking | ADDR1 | Memory Address |
| n+2 | Write handshaking | DATA | *Memory Address++ |
| n+3 | Write handshaking | DATA | *Memory Address++ |
| ... | ... | ... | ... |

**Table 20-9. Write Command**

| Step | Handshake Sequence | MODE[3:0] | DATA[7:0] |
|---|---|---|---|
| 1 | Write handshaking | CMDE | WP or WPL or EWP or EWPL |
| 2 | Write handshaking | ADDR0 | Memory Address LSB |
| 3 | Write handshaking | ADDR1 | Memory Address |
| 4 | Write handshaking | ADDR2 | Memory Address |
| 5 | Write handshaking | ADDR3 | Memory Address |
| 6 | Write handshaking | DATA | *Memory Address++ |
| 7 | Write handshaking | DATA | *Memory Address++ |
| ... | ... | ... | ... |
| n | Write handshaking | ADDR0 | Memory Address LSB |
| n+1 | Write handshaking | ADDR1 | Memory Address |
| n+2 | Write handshaking | ADDR2 | Memory Address |
| n+3 | Write handshaking | ADDR3 | Memory Address |
| n+4 | Write handshaking | DATA | *Memory Address++ |
| n+5 | Write handshaking | DATA | *Memory Address++ |
| ... | ... | ... | ... |

The Flash command **Write Page and Lock (WPL)** is equivalent to the Flash Write Command. However, the lock bit is automatically set at the end of the Flash write operation. As a lock region is composed of several pages, the programmer writes to the first pages of the lock region using Flash write commands and writes to the last page of the lock region using a Flash write and lock command.

The Flash command **Erase Page and Write (EWP)** is equivalent to the Flash Write Command. However, before programming the load buffer, the page is erased.

The Flash command **Erase Page and Write the Lock (EWPL)** combines EWP and WPL commands.

#### 20.2.5.3 Flash Full Erase Command

This command is used to erase the Flash memory planes.

All lock regions must be unlocked before the Full Erase command by using the CLB command. Otherwise, the erase command is aborted and no page is erased.

**Table 20-10. Full Erase Command**

| Step | Handshake Sequence | MODE[3:0] | DATA[15:0] or DATA[7:0] |
|---|---|---|---|
| 1 | Write handshaking | CMDE | EA |
| 2 | Write handshaking | DATA | 0 |

#### 20.2.5.4 Flash Lock Commands

Lock bits can be set using WPL or EWPL commands. They can also be set by using the **Set Lock** command **(SLB)**. With this command, several lock bits can be activated. A Bit Mask is provided as argument to the command. When bit 0 of the bit mask is set, then the first lock bit is activated.

In the same way, the **Clear Lock** command **(CLB)** is used to clear lock bits.

**Table 20-11. Set and Clear Lock Bit Command**

| Step | Handshake Sequence | MODE[3:0] | DATA[15:0] or DATA[7:0] |
|---|---|---|---|
| 1 | Write handshaking | CMDE | SLB or CLB |
| 2 | Write handshaking | DATA | Bit Mask |

Lock bits can be read using **Get Lock Bit** command **(GLB)**. The $n^{th}$ lock bit is active when the bit n of the bit mask is set..

**Table 20-12. Get Lock Bit Command**

| Step | Handshake Sequence | MODE[3:0] | DATA[15:0] or DATA[7:0] |
|---|---|---|---|
| 1 | Write handshaking | CMDE | GLB |
| 2 | Read handshaking | DATA | Lock Bit Mask Status<br>0 = Lock bit is cleared<br>1 = Lock bit is set |

#### 20.2.5.5 Flash General-purpose NVM Commands

General-purpose NVM bits (GP NVM bits) can be set using the **Set GPNVM** command **(SGPB)**. This command also activates GP NVM bits. A bit mask is provided as argument to the command. When bit 0 of the bit mask is set, then the first GP NVM bit is activated.

In the same way, the **Clear GPNVM** command **(CGPB)** is used to clear general-purpose NVM bits. The general-purpose NVM bit is deactivated when the corresponding bit in the pattern value is set to 1.

**Table 20-13. Set/Clear GP NVM Command**

| Step | Handshake Sequence | MODE[3:0] | DATA[15:0] or DATA[7:0] |
|---|---|---|---|
| 1 | Write handshaking | CMDE | SGPB or CGPB |
| 2 | Write handshaking | DATA | GP NVM bit pattern value |

General-purpose NVM bits can be read using the **Get GPNVM Bit** command **(GGPB)**. The $n^{th}$ GP NVM bit is active when bit n of the bit mask is set..

**Table 20-14. Get GP NVM Bit Command**

| Step | Handshake Sequence | MODE[3:0] | DATA[15:0] or DATA[7:0] |
|---|---|---|---|
| 1 | Write handshaking | CMDE | GGPB |
| 2 | Read handshaking | DATA | GP NVM Bit Mask Status<br>0 = GP NVM bit is cleared<br>1 = GP NVM bit is set |

#### 20.2.5.6 Flash Security Bit Command

A security bit can be set using the **Set Security Bit** command (SSE). Once the security bit is active, the Fast Flash programming is disabled. No other command can be run. An event on the Erase pin can erase the security bit once the contents of the Flash have been erased.

**Table 20-15. Set Security Bit Command**

| Step | Handshake Sequence | MODE[3:0] | DATA[15:0] or DATA[7:0] |
|---|---|---|---|
| 1 | Write handshaking | CMDE | SSE |
| 2 | Write handshaking | DATA | 0 |

Once the security bit is set, it is not possible to access FFPI. The only way to erase the security bit is to erase the Flash.

In order to erase the Flash, the user must perform the following:

- Power-off the chip
- Power-on the chip with TST = 0
- Assert Erase during a period of more than 220 ms
- Power-off the chip

Then it is possible to return to FFPI mode and check that Flash is erased.

#### 20.2.5.7 Memory Write Command

This command is used to perform a write access to any memory location.

The **Memory Write** command **(WRAM)** is optimized for consecutive writes. Write handshaking can be chained; an internal address buffer is automatically increased.

**Table 20-16. Write Command**

| Step | Handshake Sequence | MODE[3:0] | DATA[15:0] |
|---|---|---|---|
| 1 | Write handshaking | CMDE | WRAM |
| 2 | Write handshaking | ADDR0 | Memory Address LSB |
| 3 | Write handshaking | ADDR1 | Memory Address |
| 4 | Write handshaking | DATA | *Memory Address++ |
| 5 | Write handshaking | DATA | *Memory Address++ |
| ... | ... | ... | ... |
| n | Write handshaking | ADDR0 | Memory Address LSB |
| n+1 | Write handshaking | ADDR1 | Memory Address |
| n+2 | Write handshaking | DATA | *Memory Address++ |
| n+3 | Write handshaking | DATA | *Memory Address++ |
| ... | ... | ... | ... |

**Table 20-17. Write Command**

| Step | Handshake Sequence | MODE[3:0] | DATA[7:0] |
|---|---|---|---|
| 1 | Write handshaking | CMDE | WRAM |
| 2 | Write handshaking | ADDR0 | Memory Address LSB |
| 3 | Write handshaking | ADDR1 | Memory Address |
| 4 | Write handshaking | ADDR2 | Memory Address |
| 5 | Write handshaking | ADDR3 | Memory Address |
| 6 | Write handshaking | DATA | *Memory Address++ |
| 7 | Write handshaking | DATA | *Memory Address++ |
| ... | ... | ... | ... |
| n | Write handshaking | ADDR0 | Memory Address LSB |
| n+1 | Write handshaking | ADDR1 | Memory Address |
| n+2 | Write handshaking | ADDR2 | Memory Address |
| n+3 | Write handshaking | ADDR3 | Memory Address |
| n+4 | Write handshaking | DATA | *Memory Address++ |
| n+5 | Write handshaking | DATA | *Memory Address++ |
| ... | ... | ... | ... |

#### 20.2.5.8 Get Version Command

The **Get Version** (GVE) command retrieves the version of the FFPI interface.

**Table 20-18. Get Version Command**

| Step | Handshake Sequence | MODE[3:0] | DATA[15:0] or DATA[7:0] |
|---|---|---|---|
| 1 | Write handshaking | CMDE | GVE |
| 2 | Write handshaking | DATA | Version |

# 21. SAM3N Boot Program

## 21.1 Description

The SAM-BA Boot Program integrates an array of programs permitting download and/or upload into the different memories of the product.

## 21.2 Hardware and Software Constraints

- SAM-BA Boot uses the first 2048 bytes of the SRAM for variables and stacks. The remaining available size can be used for user's code.
- UART0 requirements: None

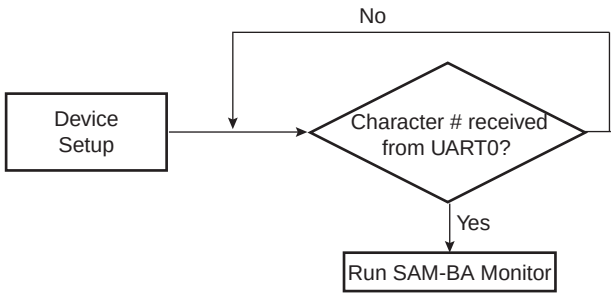
**Table 21-1. Pins Driven during Boot Program Execution**

| Peripheral | Pin | PIO Line |
|---|---|---|
| UART0 | URXD0 | PA9 |
| UART0 | UTXD0 | PA10 |

## 21.3 Flow Diagram

The Boot Program implements the algorithm in Figure 21-1.

**Figure 21-1. Boot Program Algorithm Flow Diagram**




The SAM-BA Boot program uses the internal 12 MHz RC oscillator as source clock for PLL. The MCK runs from PLL divided by 2. The core runs at 48 MHz.

## 21.4 Device Initialization

The initialization sequence is the following:

1. Stack setup
2. Setup the Embedded Flash Controller
3. Switch on internal 12 MHz RC oscillator
4. Configure PLL to run at 96 MHz
5. Switch MCK to run on PLL divided by 2
6. Configure UART0
7. Disable Watchdog
8. Wait for a character on UART0
9. Jump to SAM-BA monitor (see Section 21.5 ”SAM-BA Monitor”)

## 21.5 SAM-BA Monitor

Once the communication interface is identified, the monitor runs in an infinite loop waiting for different commands as shown in Table 21-2.

**Table 21-2. Commands Available through the SAM-BA Boot**

| Command | Action | Argument(s) | Example |
|---|---|---|---|
| **N** | set Normal mode | No argument | **N**# |
| **T** | set Terminal mode | No argument | **T**# |
| **O** | write a byte | Address, Value# | **O**200001,CA# |
| **o** | read a byte | Address,# | **o**200001,# |
| **H** | write a half word | Address, Value# | **H**200002,CAFE# |
| **h** | read a half word | Address,# | **h**200002,# |
| **W** | write a word | Address, Value# | **W**200000,CAFEDECA# |
| **w** | read a word | Address,# | **w**200000,# |
| **S** | send a file | Address,# | **S**200000,# |
| **R** | receive a file | Address, NbOfBytes# | **R**200000,1234# |
| **G** | go | Address# | **G**200200# |
| **V** | display version | No argument | **V**# |

- Mode commands:
  - Normal mode configures SAM-BA Monitor to send/receive data in binary format,
  - Terminal mode configures SAM-BA Monitor to send/receive data in ascii format.
- Write commands: Write a byte (**O**), a halfword (**H**) or a word (**W**) to the target.
  - *Address*: Address in hexadecimal.
  - *Value*: Byte, halfword or word to write in hexadecimal.
  - *Output*: '>'.
- Read commands: Read a byte (**o**), a halfword (**h**) or a word (**w**) from the target.
  - *Address*: Address in hexadecimal
  - *Output*: The byte, halfword or word read in hexadecimal following by '>'
- Send a file (**S**): Send a file to a specified address
  - *Address*: Address in hexadecimal
  - *Output*: '>'.

Note: There is a time-out on this command which is reached when the prompt '>' appears before the end of the command execution.

- Receive a file (**R**): Receive data into a file from a specified address
  - *Address*: Address in hexadecimal
  - *NbOfBytes*: Number of bytes in hexadecimal to receive
  - *Output*: '>'
- Go (**G**): Jump to a specified address and execute the code
  - *Address*: Address to jump in hexadecimal
  - *Output*: '>'
- Get Version (**V**): Return the SAM-BA boot version
  - *Output*: '>'

### 21.5.1 UART0 Serial Port

Communication is performed through the UART0 initialized to 115200 Baud, 8, n, 1.

The Send and Receive File commands use the Xmodem protocol to communicate. Any terminal performing this protocol can be used to send the application file to the target. The size of the binary file to send depends on the SRAM size embedded in the product. In all cases, the size of the binary file must be lower than the SRAM size because the Xmodem protocol requires some SRAM memory to work. See Section 21.2 ”Hardware and Software Constraints”

### 21.5.2 Xmodem Protocol

The Xmodem protocol supported is the 128-byte length block. This protocol uses a two-character CRC-16 to guarantee detection of a maximum bit error.

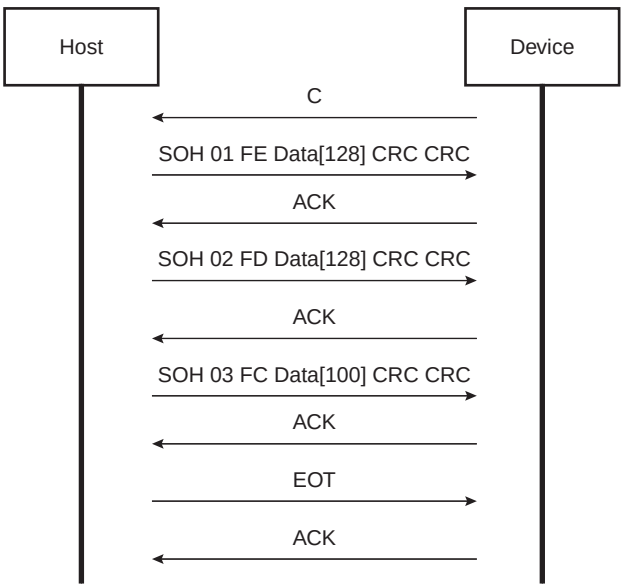
Xmodem protocol with CRC is accurate provided both sender and receiver report successful transmission. Each block of the transfer looks like:

<SOH><blk #><255-blk #><--128 data bytes--><checksum> in which:

- <SOH> = 01 hex
- <blk #> = binary number, starts at 01, increments by 1, and wraps 0FFH to 00H (not to 01)
- <255-blk #> = 1’s complement of the blk#.
- <checksum> = 2 bytes CRC16

Figure 21-2 shows a transmission using this protocol.

**Figure 21-2. Xmodem Transfer Example**




### 21.5.3 In Application Programming (IAP) Feature

The IAP feature is a function located in ROM that can be called by any software application.

When called, this function sends the desired FLASH command to the EEFC and waits for the Flash to be ready (looping while the FRDY bit is not set in the EEFC_FSR).

Since this function is executed from ROM, this allows Flash programming (such as sector write) to be done by code running in Flash.

The IAP function entry point is retrieved by reading the NMI vector in ROM (0x00800008).

This function takes two arguments in parameters:

- the index of the flash bank to be programmed: 0 for EEFC0, 1 for EEFC1. For device SAM3N, which has only one bank, this parameter has no effect and can be either 0 or 1, only EEFC0 will be accessed.
- the command to be sent to the EEFC Command register.

This function returns the value of the EEFC_FSR.

IAP software code example:

```
(unsigned int) (*IAP_Function)(unsigned long);
void main (void){

                  unsigned long FlashSectorNum = 200; //
                  unsigned long flash_cmd = 0;
                  unsigned long flash_status = 0;
                  unsigned long EFCIndex = 0; // 0:EEFC0, 1: EEFC1

/* Initialize the function pointer (retrieve function address from NMI vector)
*/

                  IAP_Function = ((unsigned long) (*)(unsigned long))
0x00800008;

/* Send your data to the sector here */

/* build the command to send to EEFC */

                  flash_cmd =  (0x5A << 24) | (FlashSectorNum << 8) |
AT91C_MC_FCMD_EWP;

/* Call the IAP function with appropriate command */

                  flash_status = IAP_Function (EFCIndex, flash_cmd);

}
```

# 22. Bus Matrix (MATRIX)

## 22.1 Description

The Bus Matrix implements a multi-layer AHB that enables parallel access paths between multiple AHB masters and slaves in a system, which increases the overall bandwidth. Bus Matrix interconnects 3 AHB Masters to 4 AHB Slaves. The normal latency to connect a master to a slave is one cycle except for the default master of the accessed slave which is connected directly (zero cycle latency).

The Bus Matrix user interface also provides a Chip Configuration User Interface with Registers that allow to support application specific features.

## 22.2 Embedded Characteristics

### 22.2.1 Matrix Masters

The Bus Matrix of the SAM3N product manages 3 masters, which means that each master can perform an access concurrently with others, to an available slave.

Each master has its own decoder, which is defined specifically for each master. In order to simplify the addressing, all the masters have the same decodings.

**Table 22-1. List of Bus Matrix Masters**

| | |
|---|---|
| Master 0 | Cortex-M3 Instruction/Data |
| Master 1 | Cortex-M3 System |
| Master 2 | Peripheral DMA Controller (PDC) |

### 22.2.2 Matrix Slaves

The Bus Matrix of the SAM3N product manages 4 slaves. Each slave has its own arbiter, allowing a different arbitration per slave.

List of Bus Matrix Slaves

| | |
|---|---|
| Slave 0 | Internal SRAM |
| Slave 1 | Internal ROM |
| Slave 2 | Internal Flash |
| Slave 3 | Peripheral Bridge |

### 22.2.3 Master to Slave Access

All the Masters can normally access all the Slaves. However, some paths do not make sense, for example allowing access from the Cortex-M3 S Bus to the Internal ROM. Thus, these paths are forbidden or simply not wired and shown as “-” in the following table.

**Table 22-2. SAM3N Master to Slave Access**

| Slaves | Masters | 0 | 1 | 2 |
|---|---|---|---|---|
| | | Cortex-M3 I/D Bus | Cortex-M3 S Bus | PDC |
| **0** | Internal SRAM | - | X | X |
| **1** | Internal ROM | X | - | X |
| **2** | Internal Flash | X | - | - |
| **3** | Peripheral Bridge | - | X | X |

## 22.3 Memory Mapping

Bus Matrix provides one decoder for every AHB Master Interface. The decoder offers each AHB Master several memory mappings. In fact, depending on the product, each memory area may be assigned to several slaves. Booting at the same address while using different AHB slaves (i.e. internal ROM or internal Flash) becomes possible.

## 22.4 Special Bus Granting Techniques

The Bus Matrix provides some speculative bus granting techniques in order to anticipate access requests from some masters. This mechanism allows to reduce latency at first accesses of a burst or single transfer. The bus granting mechanism allows to set a default master for every slave.

At the end of the current access, if no other request is pending, the slave remains connected to its associated default master. A slave can be associated with three kinds of default masters: no default master, last access master and fixed default master.

### 22.4.1 No Default Master

At the end of the current access, if no other request is pending, the slave is disconnected from all masters. No Default Master suits low power mode.

### 22.4.2 Last Access Master

At the end of the current access, if no other request is pending, the slave remains connected to the last master that performed an access request.

### 22.4.3 Fixed Default Master

At the end of the current access, if no other request is pending, the slave connects to its fixed default master. Unlike last access master, the fixed master doesn't change unless the user modifies it by a software action (field FIXED_DEFMSTR of the related MATRIX_SCFG).

To change from one kind of default master to another, the Bus Matrix user interface provides the Slave Configuration Registers, one for each slave, that allow to set a default master for each slave. The Slave Configuration Register contains two fields:

DEFMSTR_TYPE and FIXED_DEFMSTR. The 2-bit DEFMSTR_TYPE field allows to choose the default master type (no default, last access master, fixed default master) whereas the 4-bit FIXED_DEFMSTR field allows to choose a fixed default master provided that DEFMSTR_TYPE is set to fixed default master. Please refer to the Bus Matrix user interface description.

## 22.5 Arbitration

The Bus Matrix provides an arbitration mechanism that allows to reduce latency when conflict cases occur, basically when two or more masters try to access the same slave at the same time. One arbiter per AHB slave is provided, allowing to arbitrate each slave differently.

The Bus Matrix provides to the user the possibility to choose between 2 arbitration types, and this for each slave:

1. Round-Robin Arbitration (the default)
2. Fixed Priority Arbitration

This choice is given through the field ARBT of the Slave Configuration Registers (MATRIX_SCFG).

Each algorithm may be complemented by selecting a default master configuration for each slave.

When a re-arbitration has to be done, it is realized only under some specific conditions detailed in the following paragraph.

### 22.5.1 Arbitration Rules

Each arbiter has the ability to arbitrate between two or more different master's requests. In order to avoid burst breaking and also to provide the maximum throughput for slave interfaces, arbitration may only take place during the following cycles:

1. Idle Cycles: when a slave is not connected to any master or is connected to a master which is not currently accessing it.
2. Single Cycles: when a slave is currently doing a single access.
3. End of Burst Cycles: when the current cycle is the last cycle of a burst transfer. For defined length burst, predicted end of burst matches the size of the transfer but is managed differently for undefined length burst (See Section 22.5.1.1 "Undefined Length Burst Arbitration" on page 302").
4. Slot Cycle Limit: when the slot cycle counter has reached the limit value indicating that the current master access is too long and must be broken (See Section 22.5.1.2 "Slot Cycle Limit Arbitration" on page 302).

#### 22.5.1.1 Undefined Length Burst Arbitration

In order to avoid too long slave handling during undefined length bursts (INCR), the Bus Matrix provides specific logic in order to re-arbitrate before the end of the INCR transfer.

A predicted end of burst is used as for defined length burst transfer, which is selected between the following:

1. Infinite: no predicted end of burst is generated and therefore INCR burst transfer will never be broken.
2. Four beat bursts: predicted end of burst is generated at the end of each four beat boundary inside INCR transfer.
3. Eight beat bursts: predicted end of burst is generated at the end of each eight beat boundary inside INCR transfer.
4. Sixteen beat bursts: predicted end of burst is generated at the end of each sixteen beat boundary inside INCR transfer.

This selection can be done through the field ULBT of the Master Configuration Registers (MATRIX_MCFG).

#### 22.5.1.2 Slot Cycle Limit Arbitration

The Bus Matrix contains specific logic to break too long accesses such as very long bursts on a very slow slave (e.g. an external low speed memory). At the beginning of the burst access, a counter is loaded with the value previously written in the SLOT_CYCLE field of the related Slave Configuration Register (MATRIX_SCFG) and decreased at each clock cycle. When the counter reaches zero, the arbiter has the ability to re-arbitrate at the end of the current byte, half word or word transfer.

### 22.5.2 Round-Robin Arbitration

This algorithm allows the Bus Matrix arbiters to dispatch the requests from different masters to the same slave in a round-robin manner. If two or more master's requests arise at the same time, the master with the lowest number is first serviced then the others are serviced in a round-robin manner.

There are three round-robin algorithm implemented:

- Round-Robin arbitration without default master
- Round-Robin arbitration with last access master
- Round-Robin arbitration with fixed default master

#### 22.5.2.1 Round-Robin arbitration without default master

This is the main algorithm used by Bus Matrix arbiters. It allows the Bus Matrix to dispatch requests from different masters to the same slave in a pure round-robin manner. At the end of the current access, if no other request is pending, the slave is disconnected from all masters. This configuration incurs one latency cycle for the first access of a burst. Arbitration without default master can be used for masters that perform significant bursts.

#### 22.5.2.2 Round-Robin arbitration with last access master

This is a biased round-robin algorithm used by Bus Matrix arbiters. It allows the Bus Matrix to remove the one latency cycle for the last master that accessed the slave. In fact, at the end of the current transfer, if no other master request is pending, the slave remains connected to the last master that performs the access. Other non privileged masters will still get one latency cycle if they want to access the same slave. This technique can be used for masters that mainly perform single accesses.

#### 22.5.2.3 Round-Robin arbitration with fixed default master

This is another biased round-robin algorithm, it allows the Bus Matrix arbiters to remove the one latency cycle for the fixed default master per slave. At the end of the current access, the slave remains connected to its fixed default master. Every request attempted by this fixed default master will not cause any latency whereas other non privileged masters will still get one latency cycle. This technique can be used for masters that mainly perform single accesses.

### 22.5.3 Fixed Priority Arbitration

This algorithm allows the Bus Matrix arbiters to dispatch the requests from different masters to the same slave by using the fixed priority defined by the user. If two or more master's requests are active at the same time, the master with the highest priority number is serviced first. If two or more master's requests with the same priority are active at the same time, the master with the highest number is serviced first.

For each slave, the priority of each master may be defined through the Priority Registers for Slaves (MATRIX_PRAS and MATRIX_PRBS).

## 22.6 System I/O Configuration

The System I/O Configuration register (CCFG_SYSIO) allows to configure some I/O lines in System I/O mode (such as JTAG, ERASE, etc...) or as general purpose I/O lines. Enabling or disabling the corresponding I/O lines in peripheral mode or in PIO mode (PIO_PER or PIO_PDR registers) in the PIO controller as no effect. However, the direction (input or output), pull-up, pull-down and other mode control is still managed by the PIO controller.

## 22.7 Write Protect Registers

To prevent any single software error that may corrupt MATRIX behavior, **the entire MATRIX address space from address offset 0x000 to 0x1FC can be write-protected** by setting the WPEN bit in the MATRIX Write Protect Mode Register (MATRIX_WPMR).

If a write access to anywhere in the MATRIX address space from address offset 0x000 to 0x1FC is detected, then the WPVS flag in the MATRIX Write Protect Status Register (MATRIX_WPSR) is set and the field WPVSRC indicates in which register the write access has been attempted.

The WPVS flag is reset by writing the MATRIX Write Protect Mode Register (MATRIX_WPMR) with the appropriate access key WPKEY.

## 22.8 Bus Matrix (MATRIX) User Interface

**Table 22-3. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x0000 | Master Configuration Register 0 | MATRIX_MCFG0 | Read-write | 0x00000000 |
| 0x0004 | Master Configuration Register 1 | MATRIX_MCFG1 | Read-write | 0x00000000 |
| 0x0008 | Master Configuration Register 2 | MATRIX_MCFG2 | Read-write | 0x00000000 |
| 0x000C - 0x003C | Reserved | – | – | – |
| 0x0040 | Slave Configuration Register 0 | MATRIX_SCFG0 | Read-write | 0x00010010 |
| 0x0044 | Slave Configuration Register 1 | MATRIX_SCFG1 | Read-write | 0x00050010 |
| 0x0048 | Slave Configuration Register 2 | MATRIX_SCFG2 | Read-write | 0x00000010 |
| 0x004C | Slave Configuration Register 3 | MATRIX_SCFG3 | Read-write | 0x00000010 |
| 0x0050 - 0x007C | Reserved | – | – | – |
| 0x0080 | Priority Register A for Slave 0 | MATRIX_PRAS0 | Read-write | 0x00000000 |
| 0x0084 | Reserved | – | – | – |
| 0x0088 | Priority Register A for Slave 1 | MATRIX_PRAS1 | Read-write | 0x00000000 |
| 0x008C | Reserved | – | – | – |
| 0x0090 | Priority Register A for Slave 2 | MATRIX_PRAS2 | Read-write | 0x00000000 |
| 0x0094 | Reserved | – | – | – |
| 0x0098 | Priority Register A for Slave 3 | MATRIX_PRAS3 | Read-write | 0x00000000 |
| 0x009C - 0x0110 | Reserved | – | – | – |
| 0x0114 | System I/O Configuration register | CCFG_SYSIO | Read/Write | 0x00000000 |
| 0x0118- 0x011C | Reserved | – | – | – |
| 0x0120 - 0x010C | Reserved | – | – | – |
| 0x1E4 | Write Protect Mode Register | MATRIX_WPMR | Read-write | 0x0 |
| 0x1E8 | Write Protect Status Register | MATRIX_WPSR | Read-only | 0x0 |
| 0x0110 - 0x01FC | Reserved | – | – | – |

### 22.8.1 Bus Matrix Master Configuration Registers

Name: MATRIX_MCFG0..MATRIX_MCFG2

**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 | | |

- **ULBT: Undefined Length Burst Type**

0: Infinite Length Burst

No predicted end of burst is generated and therefore INCR bursts coming from this master cannot be broken.

1: Single Access

The undefined length burst is treated as a succession of single access allowing rearbitration at each beat of the INCR burst.

2: Four Beat Burst

The undefined length burst is split into a 4-beat bursts allowing rearbitration at each 4-beat burst end.

3: Eight Beat Burst

The undefined length burst is split into 8-beat bursts allowing rearbitration at each 8-beat burst end.

4: Sixteen Beat Burst

The undefined length burst is split into 16-beat bursts allowing rearbitration at each 16-beat burst end.

### 22.8.2 Bus Matrix Slave Configuration Registers

Name: MATRIX_SCFG0..MATRIX_SCFG3

**Address:** 0x400E0240

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | ARBT | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | – | – | FIXED_DEFMSTR | | | DEFMSTR_TYPE | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SLOT_CYCLE | | | | | | | |

- **SLOT_CYCLE: Maximum Number of Allowed Cycles for a Burst**

When the SLOT_CYCLE limit is reach for a burst it may be broken by another master trying to access this slave.

This limit has been placed to avoid locking very slow slaves when very long bursts are used.

This limit should not be very small though. An unreasonable small value will break every burst and the Bus Matrix will spend its time to arbitrate without performing any data transfer. 16 cycles is a reasonable value for SLOT_CYCLE.

- **DEFMSTR_TYPE: Default Master Type**

0: No Default Master

At the end of current slave access, if no other master request is pending, the slave is disconnected from all masters.

This results in having a one cycle latency for the first access of a burst transfer or for a single access.

1: Last Default Master

At the end of current slave access, if no other master request is pending, the slave stays connected to the last master having accessed it.

This results in not having the one cycle latency when the last master re-tries access on the slave again.

2: Fixed Default Master

At the end of the current slave access, if no other master request is pending, the slave connects to the fixed master the number that has been written in the FIXED_DEFMSTR field.

This results in not having the one cycle latency when the fixed master re-tries access on the slave again.

- **FIXED_DEFMSTR: Fixed Default Master**

This is the number of the Default Master for this slave. Only used if DEFMSTR_TYPE is 2. Specifying the number of a master which is not connected to the selected slave is equivalent to setting DEFMSTR_TYPE to 0.

- **ARBT: Arbitration Type**

0: Round-Robin Arbitration

1: Fixed Priority Arbitration

2: Reserved

3: Reserved

### 22.8.3 Bus Matrix Priority Registers For Slaves

**Name:** MATRIX_PRAS0..MATRIX_PRAS3

**Addresses:** 0x400E0280 [0], 0x400E0288 [1], 0x400E0290 [2], 0x400E0298 [3]

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | M3PR | | – | – | M2PR | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | M1PR | | – | – | M0PR | |

- **MxPR: Master x Priority**

Fixed priority of Master x for accessing the selected slave. The higher the number, the higher the priority.

### 22.8.4 System I/O Configuration Register

**Name:** CCFG_SYSIO

**Address:** 0x400E0314

**Access** Read-write

**Reset:** 0x0000_0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | SYSIO12 | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SYSIO7 | SYSIO6 | SYSIO5 | SYSIO4 | – | – | – | – |

- **SYSIO4: PB4 or TDI Assignment**

0 = TDI function selected.

1 = PB4 function selected.

- **SYSIO5: PB5 or TDO/TRACESWO Assignment**

0 = TDO/TRACESWO function selected.

1 = PB5 function selected.

- **SYSIO6: PB6 or TMS/SWDIO Assignment**

0 = TMS/SWDIO function selected.

1 = PB6 function selected.

- **SYSIO7: PB7 or TCK/SWCLK Assignment**

0 = TCK/SWCLK function selected.

1 = PB7 function selected.

- **SYSIO12: PB12 or ERASE Assignment**

0 = ERASE function selected.

1 = PB12 function selected.

### 22.8.5 Write Protect Mode Register

**Name:** MATRIX_WPMR

**Address:** 0x400E03E4

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | WPEN |

For more details on MATRIX_WPMR, refer to Section 22.7 “Write Protect Registers” on page 303.

- **WPEN: Write Protect ENable**

0 = Disables the Write Protect if WPKEY corresponds to 0x4D4154 (“MAT” in ASCII).

1 = Enables the Write Protect if WPKEY corresponds to 0x4D4154 (“MAT” in ASCII).

Protects the entire MATRIX address space from address offset 0x000 to 0x1FC.

- **WPKEY: Write Protect KEY** (Write-only)

Should be written at value 0x4D4154 (“MAT” in ASCII). Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

### 22.8.6 Write Protect Status Register

**Name:** MATRIX_WPSR

**Address:** 0x400E03E8

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| WPVSRC | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| WPVSRC | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | WPVS |

For more details on MATRIX_WPSR, refer to Section 22.7 “Write Protect Registers” on page 303.

- **WPVS: Write Protect Violation Status**

0: No Write Protect Violation has occurred since the last write of MATRIX_WPMR.

1: At least one Write Protect Violation has occurred since the last write of MATRIX_WPMR.

- **WPVSRC: Write Protect Violation Source**

Should be written at value 0x4D4154 (“MAT” in ASCII). Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

# 23. Peripheral DMA Controller (PDC)

## 23.1 Description

The Peripheral DMA Controller (PDC) transfers data between on-chip serial peripherals and the on- and/or off-chip memories. The link between the PDC and a serial peripheral is operated by the AHB to ABP bridge.

The user interface of each PDC channel is integrated into the user interface of the peripheral it serves. The user interface of mono directional channels (receive only or transmit only), contains two 32-bit memory pointers and two 16-bit counters, one set (pointer, counter) for current transfer and one set (pointer, counter) for next transfer. The bi-directional channel user interface contains four 32-bit memory pointers and four 16-bit counters. Each set (pointer, counter) is used by current transmit, next transmit, current receive and next receive.

Using the PDC removes processor overhead by reducing its intervention during the transfer. This significantly reduces the number of clock cycles required for a data transfer, which improves microcontroller performance.

To launch a transfer, the peripheral triggers its associated PDC channels by using transmit and receive signals. When the programmed data is transferred, an end of transfer interrupt is generated by the peripheral itself.

## 23.2 Embedded Characteristics

- Handles data transfer between peripherals and memories
- Low bus arbitration overhead
  - One Master Clock cycle needed for a transfer from memory to peripheral
  - Two Master Clock cycles needed for a transfer from peripheral to memory
- Next Pointer management for reducing interrupt latency requirement

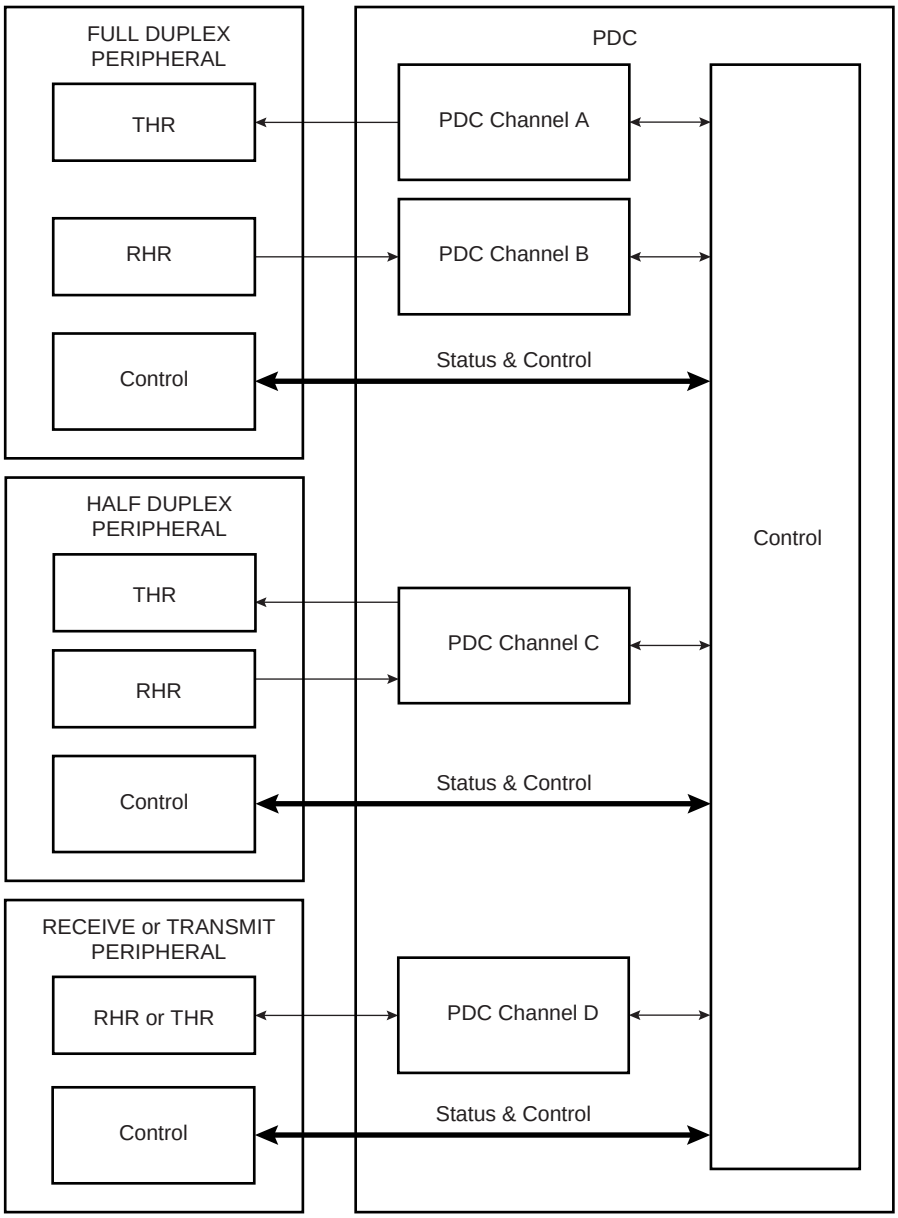
The Peripheral DMA Controller handles transfer requests from the channel according to the following priorities (Low to High priorities):

**Table 23-1. Peripheral DMA Controller**

| Instance name | Channel T/R | 100 & 64 Pins | 48 Pins |
|---|---|---|---|
| TWI0 | Transmit | x | x |
| UART0 | Transmit | x | x |
| USART0 | Transmit | x | x |
| DAC | Transmit | x | N/A |
| SPI | Transmit | x | x |
| TWI0 | Receive | x | x |
| UART0 | Receive | x | x |
| USART0 | Receive | x | x |
| ADC | Receive | x | x |
| SPI | Receive | x | x |

## 23.3 Block Diagram

**Figure 23-1. Block Diagram**



## 23.4 Functional Description

### 23.4.1 Configuration

The PDC channel user interface enables the user to configure and control data transfers for each channel. The user interface of each PDC channel is integrated into the associated peripheral user interface.

The user interface of a serial peripheral, whether it is full or half duplex, contains four 32-bit pointers (RPR, RNPR, TPR, TNPR) and four 16-bit counter registers (RCR, RNCR, TCR, TNCR). However, the transmit and receive parts of each type are programmed differently: the transmit and receive parts of a full duplex peripheral can be programmed at the same time, whereas only one part (transmit or receive) of a half duplex peripheral can be programmed at a time.

32-bit pointers define the access location in memory for current and next transfer, whether it is for read (transmit) or write (receive). 16-bit counters define the size of current and next transfers. It is possible, at any moment, to read the number of transfers left for each channel.

The PDC has dedicated status registers which indicate if the transfer is enabled or disabled for each channel. The status for each channel is located in the associated peripheral status register. Transfers can be enabled and/or disabled by setting TXTEN/TXTDIS and RXTEN/RXTDIS in the peripheral's Transfer Control Register.

At the end of a transfer, the PDC channel sends status flags to its associated peripheral. These flags are visible in the peripheral status register (ENDRX, ENDTX, RXBUFF, and TXBUFE). Refer to Section 23.4.3 and to the associated peripheral user interface.

### 23.4.2 Memory Pointers

Each full duplex peripheral is connected to the PDC by a receive channel and a transmit channel. Both channels have 32-bit memory pointers that point respectively to a receive area and to a transmit area in on- and/or off-chip memory.

Each half duplex peripheral is connected to the PDC by a bidirectional channel. This channel has two 32-bit memory pointers, one for current transfer and the other for next transfer. These pointers point to transmit or receive data depending on the operating mode of the peripheral.

Depending on the type of transfer (byte, half-word or word), the memory pointer is incremented respectively by 1, 2 or 4 bytes.

If a memory pointer address changes in the middle of a transfer, the PDC channel continues operating using the new address.

### 23.4.3 Transfer Counters

Each channel has two 16-bit counters, one for current transfer and the other one for next transfer. These counters define the size of data to be transferred by the channel. The current transfer counter is decremented first as the data addressed by current memory pointer starts to be transferred. When the current transfer counter reaches zero, the channel checks its next transfer counter. If the value of next counter is zero, the channel stops transferring data and sets the appropriate flag. But if the next counter value is greater then zero, the values of the next pointer/next counter are copied into the current pointer/current counter and the channel resumes the transfer whereas next pointer/next counter get zero/zero as values. At the end of this transfer the PDC channel sets the appropriate flags in the Peripheral Status Register.

The following list gives an overview of how status register flags behave depending on the counters' values:

- ENDRX flag is set when the PERIPH_RCR register reaches zero.
- RXBUFF flag is set when both PERIPH_RCR and PERIPH_RNCR reach zero.
- ENDTX flag is set when the PERIPH_TCR register reaches zero.
- TXBUFE flag is set when both PERIPH_TCR and PERIPH_TNCR reach zero.

These status flags are described in the Peripheral Status Register.

### 23.4.4 Data Transfers

The serial peripheral triggers its associated PDC channels' transfers using transmit enable (TXEN) and receive enable (RXEN) flags in the transfer control register integrated in the peripheral's user interface.

When the peripheral receives an external data, it sends a Receive Ready signal to its PDC receive channel which then requests access to the Matrix. When access is granted, the PDC receive channel starts reading the peripheral Receive Holding Register (RHR). The read data are stored in an internal buffer and then written to memory.

When the peripheral is about to send data, it sends a Transmit Ready to its PDC transmit channel which then requests access to the Matrix. When access is granted, the PDC transmit channel reads data from memory and puts them to Transmit Holding Register (THR) of its associated peripheral. The same peripheral sends data according to its mechanism.

### 23.4.5 PDC Flags and Peripheral Status Register

Each peripheral connected to the PDC sends out receive ready and transmit ready flags and the PDC sends back flags to the peripheral. All these flags are only visible in the Peripheral Status Register.

Depending on the type of peripheral, half or full duplex, the flags belong to either one single channel or two different channels.

#### 23.4.5.1 Receive Transfer End

This flag is set when PERIPH_RCR register reaches zero and the last data has been transferred to memory.

It is reset by writing a non zero value in PERIPH_RCR or PERIPH_RNCR.

#### 23.4.5.2 Transmit Transfer End

This flag is set when PERIPH_TCR register reaches zero and the last data has been written into peripheral THR.

It is reset by writing a non zero value in PERIPH_TCR or PERIPH_TNCR.

#### 23.4.5.3 Receive Buffer Full

This flag is set when PERIPH_RCR register reaches zero with PERIPH_RNCR also set to zero and the last data has been transferred to memory.

It is reset by writing a non zero value in PERIPH_TCR or PERIPH_TNCR.

#### 23.4.5.4 Transmit Buffer Empty

This flag is set when PERIPH_TCR register reaches zero with PERIPH_TNCR also set to zero and the last data has been written into peripheral THR.

It is reset by writing a non zero value in PERIPH_TCR or PERIPH_TNCR.

## 23.5 Peripheral DMA Controller (PDC) User Interface

**Table 23-2. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x100 | Receive Pointer Register | PERIPH[(1)]_RPR | Read-write | 0 |
| 0x104 | Receive Counter Register | PERIPH_RCR | Read-write | 0 |
| 0x108 | Transmit Pointer Register | PERIPH_TPR | Read-write | 0 |
| 0x10C | Transmit Counter Register | PERIPH_TCR | Read-write | 0 |
| 0x110 | Receive Next Pointer Register | PERIPH_RNPR | Read-write | 0 |
| 0x114 | Receive Next Counter Register | PERIPH_RNCR | Read-write | 0 |
| 0x118 | Transmit Next Pointer Register | PERIPH_TNPR | Read-write | 0 |
| 0x11C | Transmit Next Counter Register | PERIPH_TNCR | Read-write | 0 |
| 0x120 | Transfer Control Register | PERIPH_PTCR | Write-only | 0 |
| 0x124 | Transfer Status Register | PERIPH_PTSR | Read-only | 0 |

Note: 1. PERIPH: Ten registers are mapped in the peripheral memory space at the same offset. These can be defined by the user according to the function and the desired peripheral.)

### 23.5.1 Receive Pointer Register

**Name:** PERIPH_RPR

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RXPTR | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| RXPTR | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| RXPTR | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXPTR | | | | | | | |

- **RXPTR: Receive Pointer Register**

RXPTR must be set to receive buffer address.

When a half duplex peripheral is connected to the PDC, RXPTR = TXPTR.

### 23.5.2 Receive Counter Register

**Name:** PERIPH_RCR

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| RXCTR | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXCTR | | | | | | | |

- **RXCTR: Receive Counter Register**

RXCTR must be set to receive buffer size.

When a half duplex peripheral is connected to the PDC, RXCTR = TXCTR.

0 = Stops peripheral data transfer to the receiver

1 - 65535 = Starts peripheral data transfer if corresponding channel is active

### 23.5.3 Transmit Pointer Register

**Name:** PERIPH_TPR

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| TXPTR | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| TXPTR | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TXPTR | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXPTR | | | | | | | |

- **TXPTR: Transmit Counter Register**

TXPTR must be set to transmit buffer address.

When a half duplex peripheral is connected to the PDC, RXPTR = TXPTR.

### 23.5.4 Transmit Counter Register

**Name:** PERIPH_TCR

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TXCTR | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXCTR | | | | | | | |

- **TXCTR: Transmit Counter Register**

TXCTR must be set to transmit buffer size.

When a half duplex peripheral is connected to the PDC, RXCTR = TXCTR.

0 = Stops peripheral data transfer to the transmitter

1- 65535 = Starts peripheral data transfer if corresponding channel is active

### 23.5.5 Receive Next Pointer Register

**Name:** PERIPH_RNPR

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RXNPTR | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| RXNPTR | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| RXNPTR | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXNPTR | | | | | | | |

- **RXNPTR: Receive Next Pointer**

RXNPTR contains next receive buffer address.

When a half duplex peripheral is connected to the PDC, RXNPTR = TXNPTR.

### 23.5.6 Receive Next Counter Register

**Name:** PERIPH_RNCR

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| RXNCTR | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXNCTR | | | | | | | |

- **RXNCTR: Receive Next Counter**

RXNCTR contains next receive buffer size.

When a half duplex peripheral is connected to the PDC, RXNCTR = TXNCTR.

### 23.5.7 Transmit Next Pointer Register

**Name:** PERIPH_TNPR

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| TXNPTR | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| TXNPTR | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TXNPTR | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXNPTR | | | | | | | |

- **TXNPTR: Transmit Next Pointer**

TXNPTR contains next transmit buffer address.

When a half duplex peripheral is connected to the PDC, RXNPTR = TXNPTR.

### 23.5.8 Transmit Next Counter Register

**Name:** PERIPH_TNCR

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TXNCTR | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXNCTR | | | | | | | |

- **TXNCTR: Transmit Counter Next**

TXNCTR contains next transmit buffer size.

When a half duplex peripheral is connected to the PDC, RXNCTR = TXNCTR.

### 23.5.9 Transfer Control Register

**Name:** PERIPH_PTCR

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | TXTDIS | TXTEN |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | RXTDIS | RXTEN |

- **RXTEN: Receiver Transfer Enable**

0 = No effect.

1 = Enables PDC receiver channel requests if RXTDIS is not set.

When a half duplex peripheral is connected to the PDC, enabling the receiver channel requests automatically disables the transmitter channel requests. It is forbidden to set both TXTEN and RXTEN for a half duplex peripheral.

- **RXTDIS: Receiver Transfer Disable**

0 = No effect.

1 = Disables the PDC receiver channel requests.

When a half duplex peripheral is connected to the PDC, disabling the receiver channel requests also disables the transmitter channel requests.

- **TXTEN: Transmitter Transfer Enable**

0 = No effect.

1 = Enables the PDC transmitter channel requests.

When a half duplex peripheral is connected to the PDC, it enables the transmitter channel requests only if RXTEN is not set. It is forbidden to set both TXTEN and RXTEN for a half duplex peripheral.

- **TXTDIS: Transmitter Transfer Disable**

0 = No effect.

1 = Disables the PDC transmitter channel requests.

When a half duplex peripheral is connected to the PDC, disabling the transmitter channel requests disables the receiver channel requests.

### 23.5.10 Transfer Status Register

**Name:** PERIPH_PTSR

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | TXTEN |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | RXTEN |

- **RXTEN: Receiver Transfer Enable**

0 = PDC Receiver channel requests are disabled.

1 = PDC Receiver channel requests are enabled.

- **TXTEN: Transmitter Transfer Enable**

0 = PDC Transmitter channel requests are disabled.

1 = PDC Transmitter channel requests are enabled.

# 24. Clock Generator

## 24.1 Description

The Clock Generator User Interface is embedded within the Power Management Controller and is described in Section 25.15 ”Power Management Controller (PMC) User Interface”. However, the Clock Generator registers are named CKGR_.

## 24.2 Embedded Characteristics

The Clock Generator is made up of:

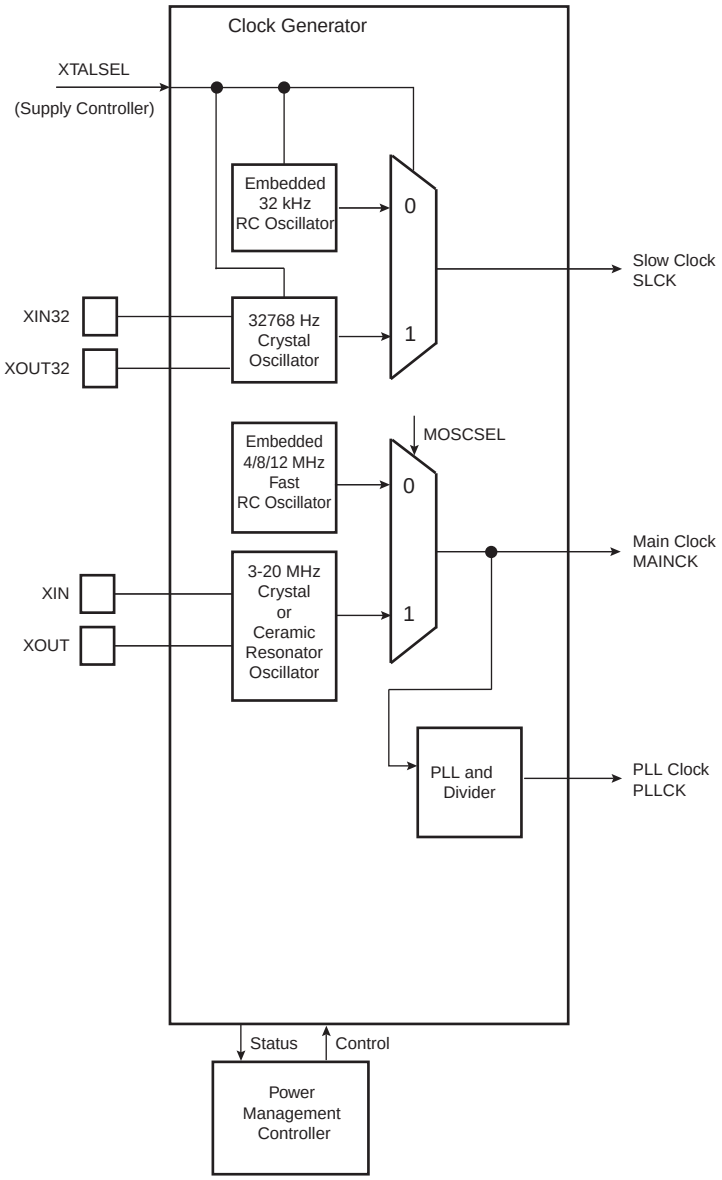
- A Low Power 32.768 kHz Slow Clock Oscillator with bypass mode
- A Low Power RC Oscillator
- A 3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator which can be bypassed
- A factory programmed Fast RC Oscillator, 3 output frequencies can be selected: 4, 8 or 12 MHz. By default 4 MHz is selected.
- A 60 to 130 MHz programmable PLL (input from 3.5 to 20 MHz), capable of providing the clock MCK to the processor and to the peripherals.

It provides the following clocks:

- SLCK, the Slow Clock, which is the only permanent clock within the system
- MAINCK is the output of the Main Clock Oscillator selection: either the Crystal or Ceramic Resonator-based Oscillator or 4/8/12 MHz Fast RC Oscillator
- PLLCK is the output of the Divider and 60 to 130 MHz programmable PLL

## 24.3 Block Diagram

**Figure 24-1. Clock Generator Block Diagram**



## 24.4 Slow Clock

The Supply Controller embeds a slow clock generator that is supplied with the VDDIO powersupply. As soon as VDDIO is supplied, both the crystal oscillator and the embedded RC oscillator are powered up, but only the embedded RC oscillator is enabled. This allows the slow clock to be valid in a short time (about 100 µs).

The Slow Clock is generated either by the Slow Clock Crystal Oscillator or by the Slow Clock RC Oscillator.

The selection between the RC or crystal oscillator is made by writing the XTALSEL bit in the Supply Controller Control Register (SUPC_CR).

### 24.4.1 Slow Clock RC Oscillator

By default, the Slow Clock RC Oscillator is enabled and selected. The user has to take into account the possible drifts of the RC Oscillator. More details are given in the section “DC Characteristics” of the product datasheet.

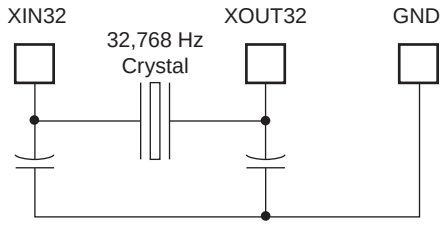
It can be disabled via the XTALSEL bit in the Supply Controller Control Register (SUPC_CR).

### 24.4.2 Slow Clock Crystal Oscillator

The Clock Generator integrates a 32.768 kHz low-power oscillator. In order to use this oscillator, the XIN32/PA7 and XOUT32/P8 pins must be connected to a 32.768 kHz crystal. Two external capacitors must be wired as shown in Figure 24-2. More details are given in the section “DC Characteristics” of the product datasheet.

Note that the user is not obliged to use the Slow Clock Crystal and can use the RC Oscillator instead.

**Figure 24-2. Typical Slow Clock Crystal Oscillator Connection**




The user can select the crystal oscillator to be the source of the slow clock, as it provides a more accurate frequency. The command is made by writing the Supply Controller Control Register (SUPC_CR) with the XTALSEL bit at 1. This results in a sequence which first configures the PIO lines multiplexed with XIN32 and XOUT32 to be driven by the oscillator, then enables the crystal oscillator and then disables the RC oscillator to save power. The switch of the slow clock source is glitch free. The OSCSEL bit of the Supply Controller Status Register (SUPC_SR) 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 can be left unconnected since by default the XIN32 and XOUT32 system I/O pins are in PIO input mode after reset.

The user can also set the crystal oscillator in bypass mode instead of connecting a crystal. In this case, the user has to provide the external clock signal on XIN32. The input characteristics of

the XIN32 pin are given in the product electrical characteristics section. In order to set the bypass mode, the OSCBYPASS bit of the Supply Controller Mode Register (SUPC_MR) needs to be set at 1.

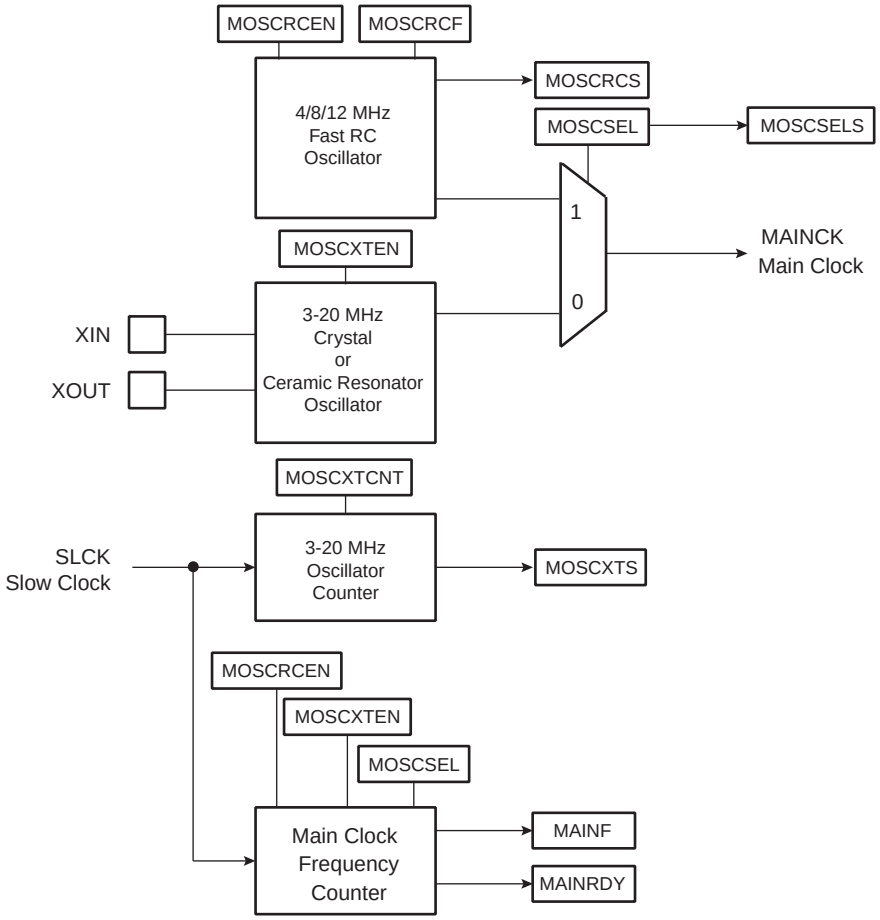
The user can set the Slow Clock Crystal Oscillator in bypass mode instead of connecting a crystal. In this case, the user has to provide the external clock signal on XIN32. The input characteristics of the XIN32 pin under these conditions are given in the product electrical characteristics section.

The programmer has to be sure to set the OSCBYPASS bit in the SUPC_MR and XTALSEL bit in the SUPC_CR.

## 24.5 Main Clock

Figure 24-3 shows the Main Clock block diagram.

**Figure 24-3. Main Clock Block Diagram**




The Main Clock has two sources:

- 4/8/12 MHz Fast RC Oscillator which starts very quickly and is used at startup
- 3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator which can be bypassed

### 24.5.1 4/8/12 MHz Fast RC Oscillator

After reset, the 4/8/12 MHz Fast RC Oscillator is enabled with the 4 MHz frequency selected and it is selected as the source of MAINCK. MAINCK is the default clock selected to start up the system.

The Fast RC Oscillator 8 and 12 MHz frequencies are calibrated in production. Note that is not the case for the 4 MHz frequency.

Please refer to the “DC Characteristics” section of the product datasheet.

The software can disable or enable the 4/8/12 MHz Fast RC Oscillator with the MOSCRCEN bit in the Clock Generator Main Oscillator Register (CKGR_MOR).

The user can also select the output frequency of the Fast RC Oscillator: either 4 MHz, 8 MHz or 12 MHz are available. It can be done through MOSCRCF bits in CKGR_MOR. When changing this frequency selection, the MOSCRCS bit in the Power Management Controller Status Register (PMC_SR) is automatically cleared and MAINCK is stopped until the oscillator is stabilized. Once the oscillator is stabilized, MAINCK restarts and MOSCRCS is set.

When disabling the Main Clock by clearing the MOSCRCEN bit in CKGR_MOR, the MOSCRCS bit in the PMC_SR is automatically cleared, indicating the Main Clock is off.

Setting the MOSCRCS bit in the Power Management Controller Interrupt Enable Register (PMC_IER) can trigger an interrupt to the processor.

It is recommended to disable the Main Clock as soon as the processor no longer uses it and runs out of SLCK or PLLCK.

The CAL4, CAL8 and CAL12 values in the PMC_OCR are the default values set by Atmel during production. These values are stored in a specific Flash memory area different from the main memory plane. These values cannot be modified by the user and cannot be erased by a Flash erase command or by the ERASE pin. Values written by the user's application in the PMC_OCR are reset after each power up or peripheral reset.

### 24.5.2 4/8/12 MHz Fast RC Oscillator Clock Frequency Adjustment

It is possible for the user to adjust the main RC oscillator frequency through PMC_OCR. By default, SEL4/8/12 are low, so the RC oscillator will be driven with Flash calibration bits which are programmed during chip production.

The user can adjust the trimming of the 4/8/12 MHz Fast RC oscillator through this register in order to obtain more accurate frequency (to compensate derating factors such as temperature and voltage).

In order to calibrate the 4 MHz Fast RC oscillator frequency, SEL4 must be set to 1 and a valid frequency value must be configured in CAL4. Likewise, SEL8/12 must be set to 1 and a trim value must be configured in CAL8/12 in order to adjust the 8/12 MHz frequency oscillator.

However, the adjustment can not be done to the frequency from which the oscillator is operating. For example, while running from a frequency of 8 MHz, the user can adjust the 4 and 12 MHz frequency but not the 8 MHz.

### 24.5.3 3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator

After reset, the 3 to 20 MHz Crystal or Ceramic Resonator-based oscillator is disabled and it is not selected as the source of MAINCK.

The user can select the 3 to 20 MHz Crystal or Ceramic Resonator-based oscillator to be the source of MAINCK, as it provides a more accurate frequency. The software enables or disables the main oscillator so as to reduce power consumption by clearing the MOSCXTEN bit in the CKGR_MOR.

When disabling the main oscillator by clearing the MOSCXTEN bit in CKGR_MOR, the MOSCXTS bit in PMC_SR is automatically cleared, indicating the Main Clock is off.

When enabling the main oscillator, the user must initiate the main oscillator counter with a value corresponding to the startup time of the oscillator. This startup time depends on the crystal frequency connected to the oscillator.

When the MOSCXTEN bit and the MOSCXTCNT are written in CKGR_MOR to enable the main oscillator, the MOSCXTS bit in the Power Management Controller Status Register (PMC_SR) is cleared and the counter starts counting down on the slow clock divided by 8 from the MOSCXTCNT value. Since the MOSCXTCNT value is coded with 8 bits, the maximum startup time is about 62 ms.

When the counter reaches 0, the MOSCXTS bit is set, indicating that the main clock is valid. Setting the MOSCXTS bit in PMC_IMR can trigger an interrupt to the processor.

### 24.5.4 Main Clock Oscillator Selection

The user can select either the 4/8/12 MHz Fast RC oscillator or the 3 to 20 MHz Crystal or Ceramic Resonator-based oscillator to be the source of Main Clock.

The advantage of the 4/8/12 MHz Fast RC oscillator is that it provides fast startup time, this is why it is selected by default (to start up the system) and when entering Wait Mode.

The advantage of the 3 to 20 MHz Crystal or Ceramic Resonator-based oscillator is that it is very accurate.

The selection is made by writing the MOSCSEL bit in the CKGR_MOR. The switch of the Main Clock source is glitch free, so there is no need to run out of SLCK or PLLCK in order to change the selection. The MOSCSELS bit of the Power Management Controller Status Register (PMC_SR) allows knowing when the switch sequence is done.

Setting the MOSCSELS bit in PMC_IMR can trigger an interrupt to the processor.

### 24.5.5 Main Clock Frequency Counter

The device features a Main Clock frequency counter that provides the frequency of the Main Clock.

The Main Clock frequency counter is reset and starts incrementing at the Main Clock speed after the next rising edge of the Slow Clock in the following cases:

- when the 4/8/12 MHz Fast RC oscillator clock is selected as the source of Main Clock and when this oscillator becomes stable (i.e., when the MOSCRCS bit is set)
- when the 3 to 20 MHz Crystal or Ceramic Resonator-based oscillator is selected as the source of Main Clock and when this oscillator becomes stable (i.e., when the MOSCXTS bit is set)
- when the Main Clock Oscillator selection is modified

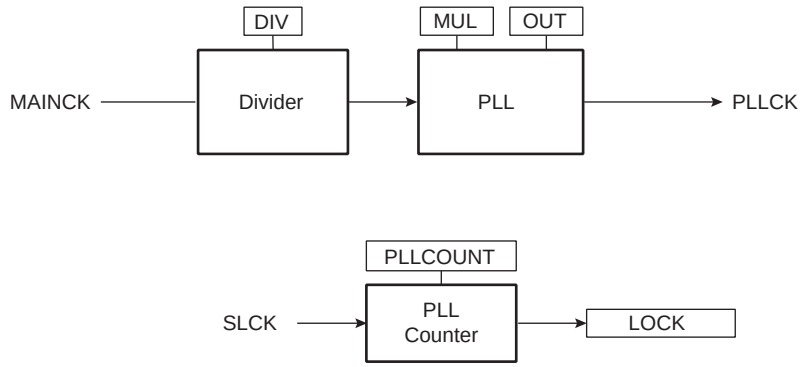
Then, at the 16th falling edge of Slow Clock, the MAINFRDY bit in the Clock Generator Main Clock Frequency Register (CKGR_MCFR) is set and the counter stops counting. Its value can be read in the MAINF field of CKGR_MCFR and gives the number of Main Clock cycles during 16 periods of Slow Clock, so that the frequency of the 4/8/12 MHz Fast RC oscillator or 3 to 20 MHz Crystal or Ceramic Resonator-based oscillator can be determined.

## 24.6 Divider and PLL Block

The device features a Divider/PLL Block that permits a wide range of frequencies to be selected on either the master clock, the processor clock or the programmable clock outputs.

Figure 24-4 shows the block diagram of the divider and PLL block.

**Figure 24-4. Divider and PLL Block Diagram**




### 24.6.1 Divider and Phase Lock Loop Programming

The divider can be set between 1 and 255 in steps of 1. When the divider field (DIV) is set to 0, the output of the divider and the PLL output is a continuous signal at level 0. On reset, the DIV field is set to 0, thus the PLL input clock is set to 0.

The PLL allows multiplication of the divider's output. The PLL clock signal has a frequency that depends on the respective source signal frequency and on the DIV and MUL parameters. The factor applied to the source signal frequency is (MUL + 1)/DIV. When MUL is written to 0, the PLL is disabled and its power consumption is saved. Re-enabling the PLL can be performed by writing a value higher than 0 in the MUL field.

Whenever the PLL is re-enabled or one of its parameters is changed, the LOCK bit in PMC_SR is automatically cleared. The value written in the PLLCOUNT field in Clock Generator PLL Register (CKGR_PLLR) is 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 bit in PMC_MCKR.

It is forbidden to change 4/8/12 Fast RC oscillator frequency or main selection in CKGR_MOR while Master clock source is PLL and PLL reference clock is Fast RC oscillator.

The user must:

- Switch on the Main RC oscillator by writing 1 in CSS field of PMC_MCKR.
- Change the frequency (MOSCRCF) or oscillator selection (MOSCSEL) in CKGR_MOR.
- Wait for MOSCRCS (if frequency changes) or MOSCSELS (if oscillator selection changes) in PMC_IER.
- Disable and then enable the PLL (LOCK in PMC_IDR and PMC_IER)
- Wait for PLLRDY.
- Switch back to PLL.

# 25. Power Management Controller (PMC)

## 25.1 Description

The Power Management Controller (PMC) optimizes power consumption by controlling all system and user peripheral clocks. The PMC enables/disables the clock inputs to many of the peripherals and the Cortex-M3 Processor.

The Supply Controller selects between the 32 kHz RC oscillator or the crystal oscillator. The unused oscillator is disabled automatically so that power consumption is optimized.

By default, at startup the chip runs out of the Master Clock using the fast RC oscillator running at 4 MHz.

The user can trim the 8 and 12 MHz RC Oscillator frequency by software.

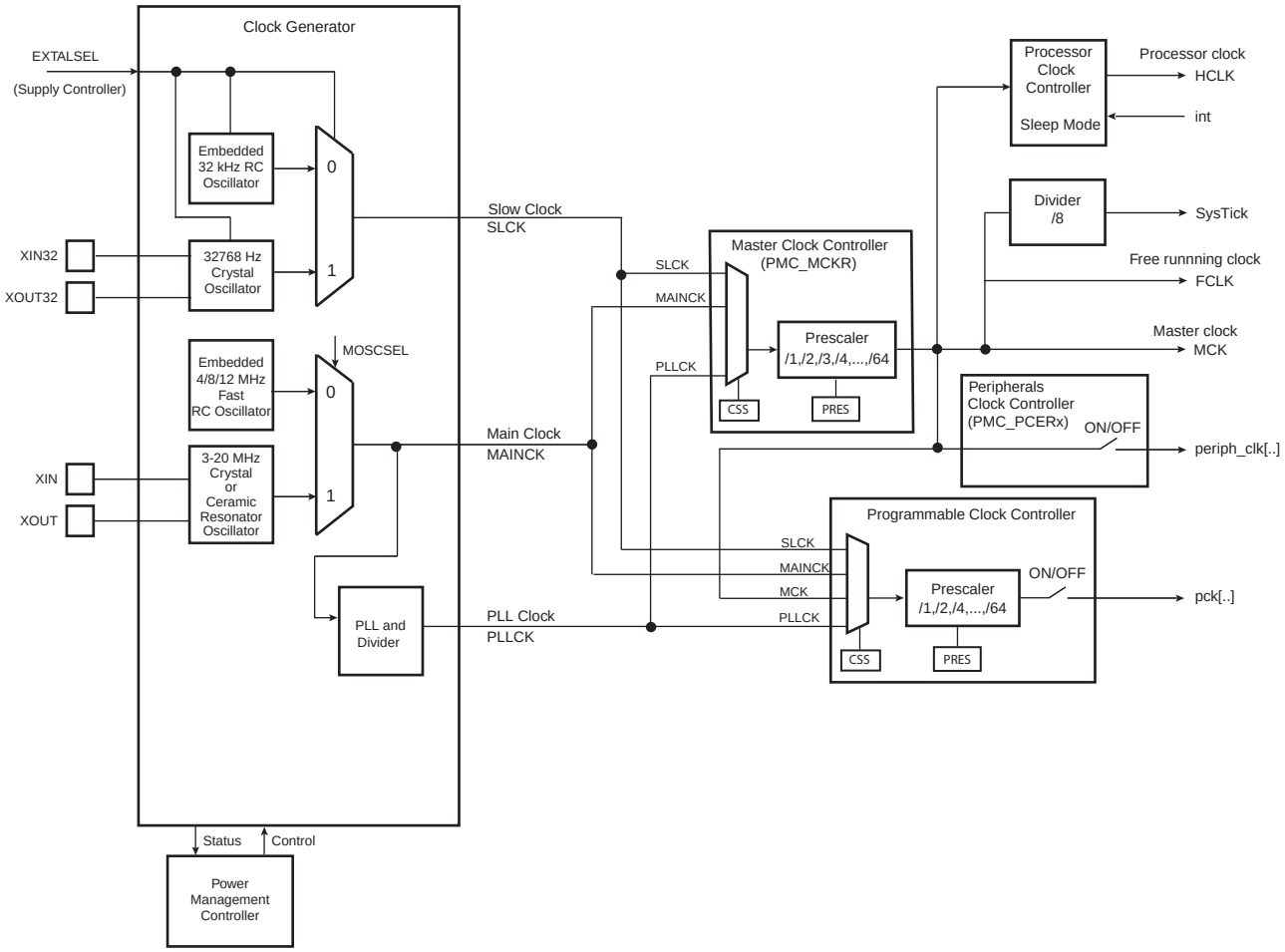
## 25.2 Embedded Characteristics

The Power Management Controller provides the following clocks:

- MCK, the Master Clock, programmable from a few hundred Hz to the maximum operating frequency of the device. It is available to the modules running permanently, such as the Enhanced Embedded Flash Controller.
- Processor Clock (HCLK) is automatically switched off when the processor enters Sleep Mode
- Free running processor Clock (FCLK)
- the Cortex-M3 SysTick external clock
- Peripheral Clocks, typically MCK, provided to the embedded peripherals (USART, SPI, TWI, TC, etc.) and are independently controllable. In order to reduce the number of clock names in a product, the Peripheral Clocks are named MCK in the product datasheet.
- Programmable Clock Outputs can be selected from the clocks provided by the clock generator and driven on the PCKx pins.

## 25.3 Block Diagram

**Figure 25-1. General Clock Block Diagram**




## 25.4 Master Clock Controller

The Master Clock Controller provides selection and division of the Master Clock (MCK). MCK is the clock provided to all the peripherals.

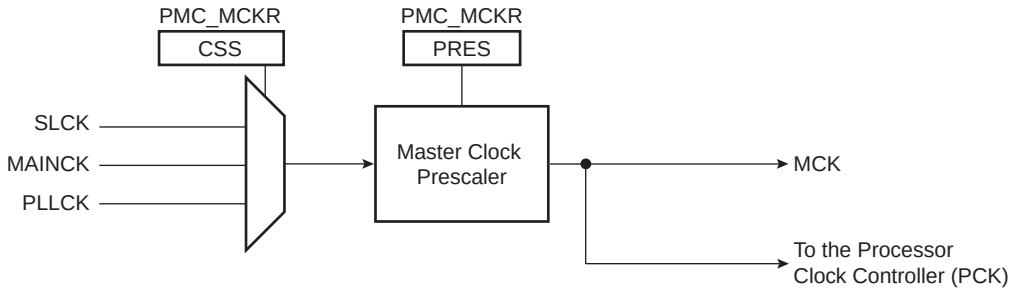
The Master Clock is selected from one of the clocks provided by the Clock Generator. Selecting the Slow Clock provides a Slow Clock signal to the whole device. Selecting the Main Clock saves power consumption of the PLL.

The Master Clock Controller is made up of a clock selector and a prescaler.

The Master Clock selection is made by writing the CSS field (Clock Source Selection) in PMC_MCKR (Master Clock Register). The prescaler supports the division by a power of 2 of the selected clock between 1 and 64, and the division by 3. The PRES field in PMC_MCKR programs the prescaler.

Each time PMC_MCKR is written to define a new Master Clock, the MCKRDY bit is cleared in PMC_SR. It reads 0 until the Master Clock is established. Then, the MCKRDY bit is set and can trigger an interrupt to the processor. This feature is useful when switching from a high-speed clock to a lower one to inform the software when the change is actually done.

**Figure 25-2. Master Clock Controller**




## 25.5 Processor Clock Controller

The PMC features a Processor Clock Controller (HCLK) that implements the Processor Sleep Mode. The Processor Clock can be disabled by executing the WFI (WaitForInterrupt) or the WFE (WaitForEvent) processor instruction once the LPM bit is at 0 in the PMC Fast Startup Mode Register (PMC_FSMR).

The Processor Clock HCLK is enabled after a reset and is automatically re-enabled by any enabled interrupt. The Processor Sleep Mode is achieved by disabling the Processor Clock, which is automatically re-enabled by any enabled fast or normal interrupt, or by the reset of the product.

When Processor Sleep Mode is entered, the current instruction is finished before the clock is stopped, but this does not prevent data transfers from other masters of the system bus.

## 25.6 SysTick Clock

The SysTick calibration value is fixed at 6000 which allows the generation of a time base of 1 ms with SysTick clock at 6 MHz (max MCK/8).

## 25.7 Peripheral Clock Controller

The Power Management Controller controls the clocks of each embedded peripheral by means of the Peripheral Clock Controller. The user can individually enable and disable the Clock on the peripherals.

The user can also enable and disable these clocks by writing Peripheral Clock Enable (PMC_PCER) and Peripheral Clock Disable (PMC_PCDR) registers. The status of the peripheral clock activity can be read in the Peripheral Clock Status Register (PMC_PCSR).

When a peripheral clock is disabled, the clock is immediately stopped. The peripheral clocks are automatically disabled after a reset.

In order to stop a peripheral, it is recommended that the system software wait until the peripheral has executed its last programmed operation before disabling the clock. This is to avoid data corruption or erroneous behavior of the system.

The bit number within the Peripheral Clock Control registers (PMC_PCER, PMC_PCDR and PMC_PCSR) is the Peripheral Identifier defined at the product level. The bit number corresponds to the interrupt source number assigned to the peripheral.

## 25.8 Free Running Processor Clock

The Free running processor clock (FCLK) used for sampling interrupts and clocking debug blocks ensures that interrupts can be sampled, and sleep events can be traced, while the processor is sleeping. It is connected to Master Clock (MCK).

## 25.9 Programmable Clock Output Controller

The PMC controls 3 signals to be output on external pins, PCKx. Each signal can be independently programmed via the PMC_PCKx registers.

PCKx can be independently selected between the Slow Clock (SLCK), the Main Clock (MAINCK), the PLL Clock (PLLCK) and the Master Clock (MCK) by writing the CSS field in PMC_PCKx. Each output signal can also be divided by a power of 2 between 1 and 64 by writing the PRES (Prescaler) field in PMC_PCKx.

Each output signal can be enabled and disabled by writing 1 in the corresponding bit, PCKx of PMC_SCER and PMC_SCDR, respectively. Status of the active programmable output clocks are given in the PCKx bits of PMC_SCSR (System Clock Status Register).

Moreover, like the PCK, a status bit in PMC_SR indicates that the Programmable Clock is actually what has been programmed in the Programmable Clock registers.

As the Programmable Clock Controller does not manage with glitch prevention when switching clocks, it is strongly recommended to disable the Programmable Clock before any configuration change and to re-enable it after the change is actually performed.

## 25.10 Fast Startup

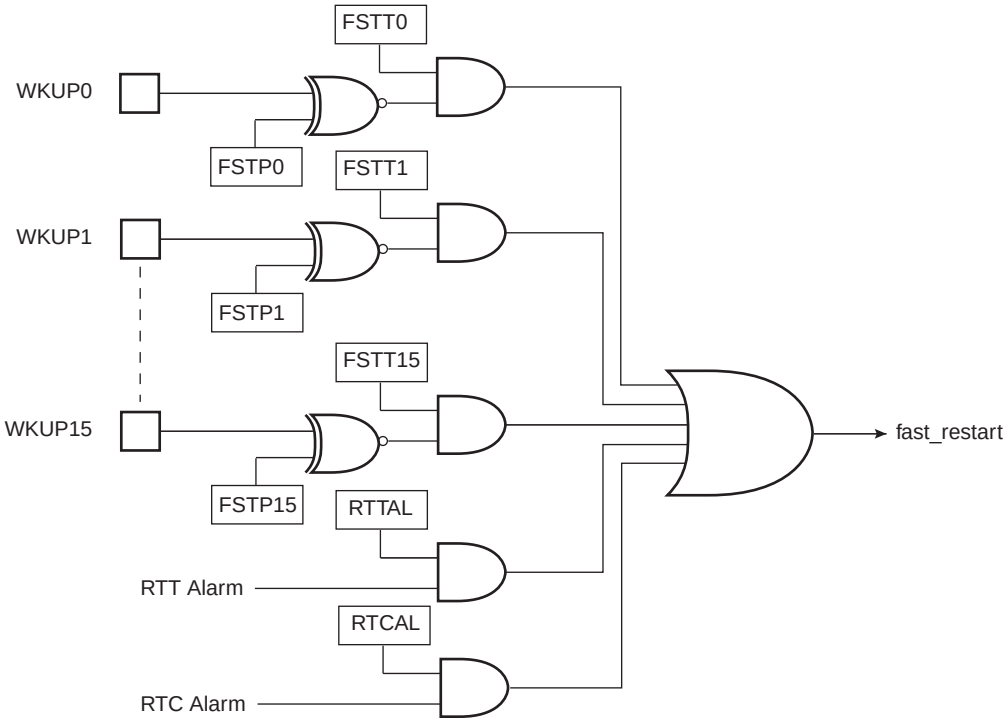
The SAM3N device allows the processor to restart in less than 10 µs while the device is in Wait mode. The system enters Wait mode either by writing the WAITMODE bit at 1 in the Clock Generator Main Oscillator Register (CKGR_MOR), or by executing the WaitForEvent (WFE) instruction of the processor once the LPM bit is at 1 in the PMC Fast Startup Mode Register (PMC_FSMR).

IMPORTANT: Prior to asserting any WFE instruction to the processor, the internal sources of wakeup provided by the RTT, RTC, and SM must be cleared and verified too that none of the enabled external wakeup inputs (WKUP) hold an active polarity.

A Fast Startup is enabled upon the detection of a programmed level on one of the 16 wake-up inputs (WKUP), SM or upon an active alarm from the RTC and RTT. The polarity of the 16 wake-up inputs is programmable by writing the PMC Fast Startup Polarity Register (PMC_FSPR).

The Fast Restart circuitry, as shown in Figure 25-3, is fully asynchronous and provides a fast startup signal to the Power Management Controller. As soon as the fast startup signal is asserted, this automatically restarts the embedded 4/8/12 MHz Fast RC oscillator.

**Figure 25-3. Fast Startup Circuitry**




Each wake-up input pin and alarm can be enabled to generate a Fast Startup event by writing at 1 the corresponding bit in the Fast Startup Mode Register PMC_FSMR.

The user interface does not provide any status for Fast Startup, but the user can easily recover this information by reading the PIO Controller, and the status registers of the RTC and RTT.

## 25.11 Clock Failure Detector

The clock failure detector allows to monitor the 3 to 20 MHz Crystal or Ceramic Resonator-based oscillator and to detect an eventual defect of this oscillator (for example if the crystal is unconnected).

The clock failure detector can be enabled or disabled by means of the CFDEN bit in the CKGR_MOR. After reset, the detector is disabled. However, if the 3 to 20 MHz Crystal or Ceramic Resonator-based Oscillator is disabled, the clock failure detector is disabled too.

A failure is detected by means of a counter incrementing on the 3 to 20 MHz Crystal oscillator or Ceramic Resonator-based oscillator clock edge and timing logic clocked on the slow clock RC oscillator controlling the counter. The counter is cleared when the slow clock RC oscillator signal is low and enabled when the slow clock RC oscillator is high. Thus the failure detection time is 1 slow clock RC oscillator clock period. If, during the high level period of slow clock RC oscillator, less than 8 fast crystal clock periods have been counted, then a failure is declared.

If a failure of the 3 to 20 MHz Crystal or Ceramic Resonator-based oscillator clock is detected, the CFDEV flag is set in the PMC Status Register (PMC_SR), and can generate an interrupt if it is not masked. The interrupt remains active until a read operation in the PMC_SR. The user can know the status of the clock failure detector at any time by reading the CFDS bit in the PMC_SR.

If the 3 to 20 MHz Crystal or Ceramic Resonator-based oscillator clock is selected as the source clock of MAINCK (MOSCSEL = 1), and if the Master Clock Source is PLLCK (CSS = 2), then a clock failure detection switches automatically the Master Clock on MAINCK. Then whatever the PMC configuration is, a clock failure detection switches automatically MAINCK on the 4/8/12 MHz Fast RC Oscillator clock. If the Fast RC oscillator is disabled when a clock failure detection occurs, it is automatically re-enabled by the clock failure detection mechanism.

It takes 2 slow clock RC oscillator cycles to detect and switch from the 3 to 20 MHz Crystal or Ceramic Resonator-based oscillator to the 4/8/12 MHz Fast RC Oscillator if the Master Clock Source is Main Clock or 3 slow clock RC oscillator cycles if the Master Clock Source is PLL.

The user can know the status of the fault output at any time by reading the FOS bit in the PMC_SR.

## 25.12 Programming Sequence

1. Enabling the Main Oscillator:
   The main oscillator is enabled by setting the MOSCXTEN field in the CKGR_MOR. The user can define a start-up time. This can be achieved by writing a value in the MOSCXTST field in the CKGR_MOR. Once this register has been correctly configured, the user must wait for MOSCXTS field in the PMC_SR to be set. This can be done either by polling the status register, or by waiting the interrupt line to be raised if the associated interrupt to MOSCXTS has been enabled in the PMC_IER.
   Start Up Time = 8 * MOSCXTST / SLCK = 56 Slow Clock Cycles.
   So, the main oscillator will be enabled (MOSCXTS bit set) after 56 Slow Clock Cycles.
2. Checking the Main Oscillator Frequency (Optional):

   In some situations the user may need an accurate measure of the main clock frequency. This measure can be accomplished via the Clock Generator Main Clock Frequency Register (CKGR_MCFR).

   Once the MAINFRDY field is set in CKGR_MCFR, the user may read the MAINF field in CKGR_MCFR. This provides the number of main clock cycles within sixteen slow clock cycles.
3. Setting PLL and Divider:

   All parameters needed to configure PLL and the divider are located in the Clock Generator PLL Register (CKGR_PLLR).

   The DIV field is used to control the divider itself. It must be set to 1 when PLL is used. By default, DIV parameter is set to 0 which means that the divider is turned off.

   The MUL field is the PLL multiplier factor. This parameter can be programmed between 0 and 2047. If MUL is set to 0, PLL will be turned off, otherwise the PLL output frequency is PLL input frequency multiplied by (MUL + 1).

   The PLLCOUNT field specifies the number of slow clock cycles before LOCK bit is set in the PMC_SR after CKGR_PLLR has been written.

   Once the PMC_PLL register has been written, the user must wait for the LOCK bit to be set in the PMC_SR. This can be done either by polling the status register or by waiting the interrupt line to be raised if the associated interrupt to LOCK has been enabled in the PMC_IER. All parameters in CKGR_PLLR can be programmed in a single write operation. If at some stage one of the following parameters, MUL, DIV is modified, LOCK bit will go low to indicate that PLL is not ready yet. When PLL is locked, LOCK will be set again. The user is constrained to wait for LOCK bit to be set before using the PLL output clock.
4. Selection of Master Clock and Processor Clock

   The Master Clock and the Processor Clock are configurable via the PMC_MCKR.

   The CSS field is used to select the Master Clock divider source. By default, the selected clock source is main clock.

   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 the status register or by waiting for the interrupt line to be raised if the associated interrupt to MCKRDY has been enabled in the PMC_IER.

   The PMC_MCKR must not be programmed in a single write operation. The preferred programming sequence for the PMC_MCKR is as follows:

- If a new value for CSS field corresponds to PLL Clock,
  - Program the PRES field in the PMC_MCKR.
  - Wait for the MCKRDY bit to be set in the PMC_SR.
  - Program the CSS field in the PMC_MCKR.
  - Wait for the MCKRDY bit to be set in the PMC_SR.
- If a new value for CSS field corresponds to Main Clock or Slow Clock,

- Program the CSS field in the PMC_MCKR.
- Wait for the MCKRDY bit to be set in the PMC_SR.
- Program the PRES field in the PMC_MCKR.
- Wait for the MCKRDY bit to be set in the PMC_SR.

If at some stage one of the following parameters, CSS or PRES, is modified, the MCKRDY bit will go low to indicate that the Master Clock and the Processor Clock are not ready yet. The user must wait for MCKRDY bit to be set again before using the Master and Processor Clocks.

Note: IF PLL clock was selected as the Master Clock and the user decides to modify it by writing in CKGR_PLLR, the MCKRDY flag will go low while PLL is unlocked. Once PLL is locked again, LOCK goes high and MCKRDY is set. While PLL is unlocked, the Master Clock selection is automatically changed to Slow Clock. For further information, see Section 25.13.2 “Clock Switching Waveforms” on page 341.

Code Example:

```
write_register(PMC_MCKR,0x00000001)
wait (MCKRDY=1)
write_register(PMC_MCKR,0x00000011)
wait (MCKRDY=1)
```

The Master Clock is main clock divided by 2.

The Processor Clock is the Master Clock.

5. Selection of Programmable Clocks

   Programmable clocks are controlled via registers PMC_SCER, PMC_SCDR, and PMC_SCSR.

   Programmable clocks can be enabled and/or disabled via the PMC_SCER and PMC_SCDR. Three Programmable clocks can be enabled or disabled. The PMC_SCSR provides a clear indication as to which Programmable clock is enabled. By default all Programmable clocks are disabled.

   PMC_PCKx registers are used to configure Programmable clocks.

   The CSS field is used to select the Programmable clock divider source. Three clock options are available: main clock, slow clock, PLL. By default, the clock source selected is slow clock.

   The PRES field is used to control the Programmable clock prescaler. It is possible to choose between different values (1, 2, 4, 8, 16, 32, 64). Programmable clock output is prescaler input divided by PRES parameter. By default, the PRES parameter is set to 0 which means that master clock is equal to slow clock.

   Once the PMC_PCKx register has been programmed, The corresponding Programmable clock must be enabled and the user is constrained to wait for the PCKRDYx bit to be set in the PMC_SR. This can be done either by polling the status register or by waiting the interrupt line to be raised if the associated interrupt to PCKRDYx has been enabled in the 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.

6. Enabling Peripheral Clocks

   Once all of the previous steps have been completed, the peripheral clocks can be enabled and/or disabled via registers PMC_PCER0, PMC_PCER1, PMC_PCDR0 and PMC_PCDR1.

## 25.13 Clock Switching Details

### 25.13.1 Master Clock Switching Timings

Table 25-1 gives the worst case timings required for the Master Clock to switch from one selected clock to another one. This is in the event that the prescaler is de-activated. When the prescaler is activated, an additional time of 64 clock cycles of the new selected clock has to be added.

**Table 25-1. Clock Switching Timings (Worst Case)**

| From<br>To | Main Clock | SLCK | PLL Clock |
|---|---|---|---|
| Main Clock | – | 4 x SLCK +<br>2.5 x Main Clock | 3 x PLL Clock +<br>4 x SLCK +<br>1 x Main Clock |
| SLCK | 0.5 x Main Clock +<br>4.5 x SLCK | – | 3 x PLL Clock +<br>5 x SLCK |
| PLL Clock | 0.5 x Main Clock +<br>4 x SLCK +<br>PLLCOUNT x SLCK +<br>2.5 x PLL Clock | 2.5 x PLL Clock +<br>5 x SLCK +<br>PLLCOUNT x SLCK | 2.5 x PLL Clock +<br>4 x SLCK +<br>PLLCOUNT x SLCK |

### 25.13.2 Clock Switching Waveforms

**Figure 25-4. Switch Master Clock from Slow Clock to PLL Clock**

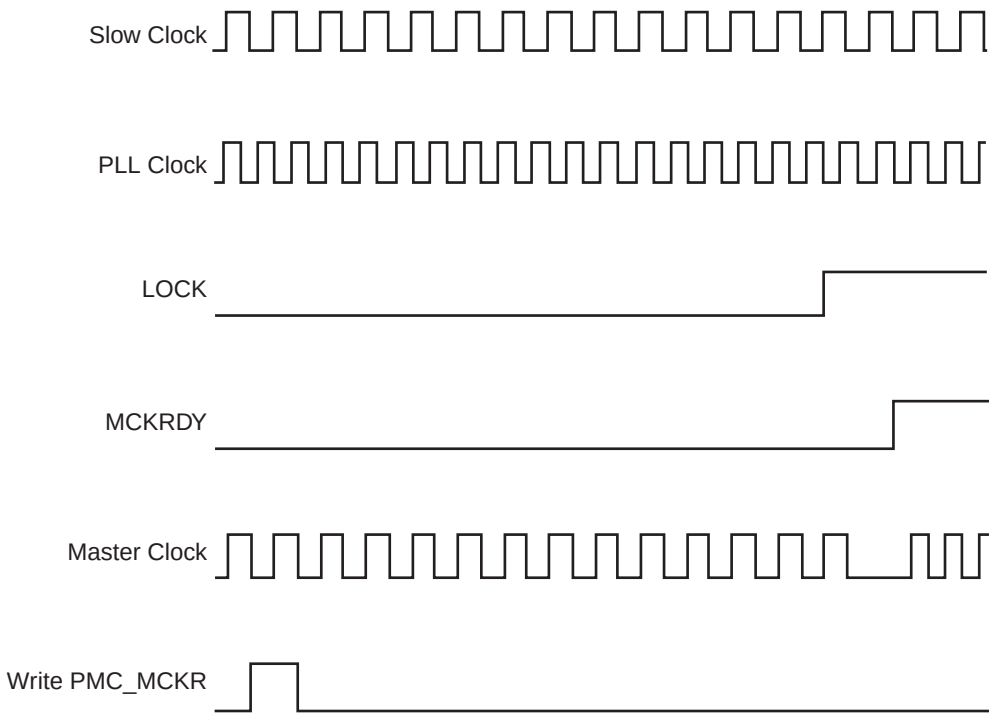


**Figure 25-5. Switch Master Clock from Main Clock to Slow Clock**

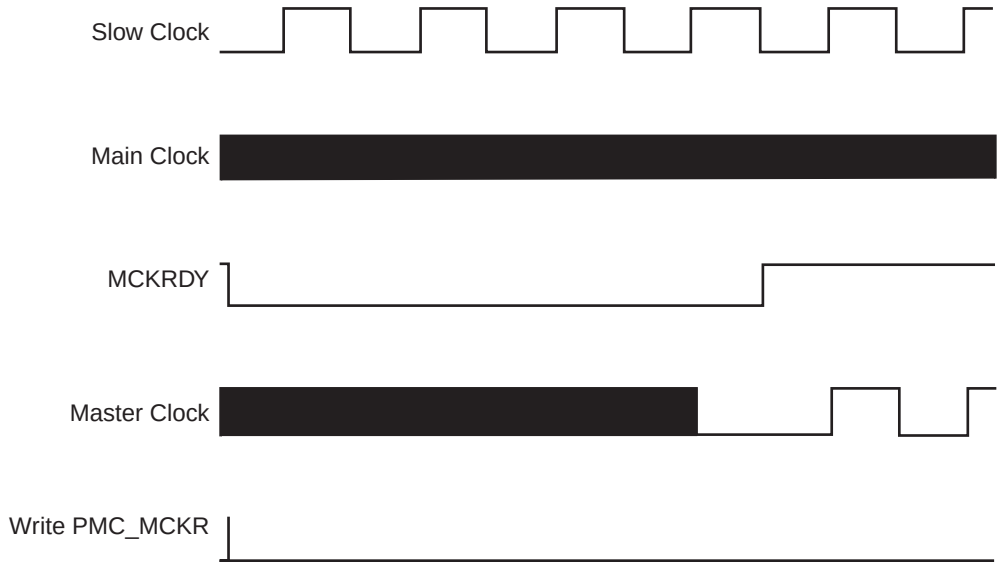



**Figure 25-6. Change PLL Programming**

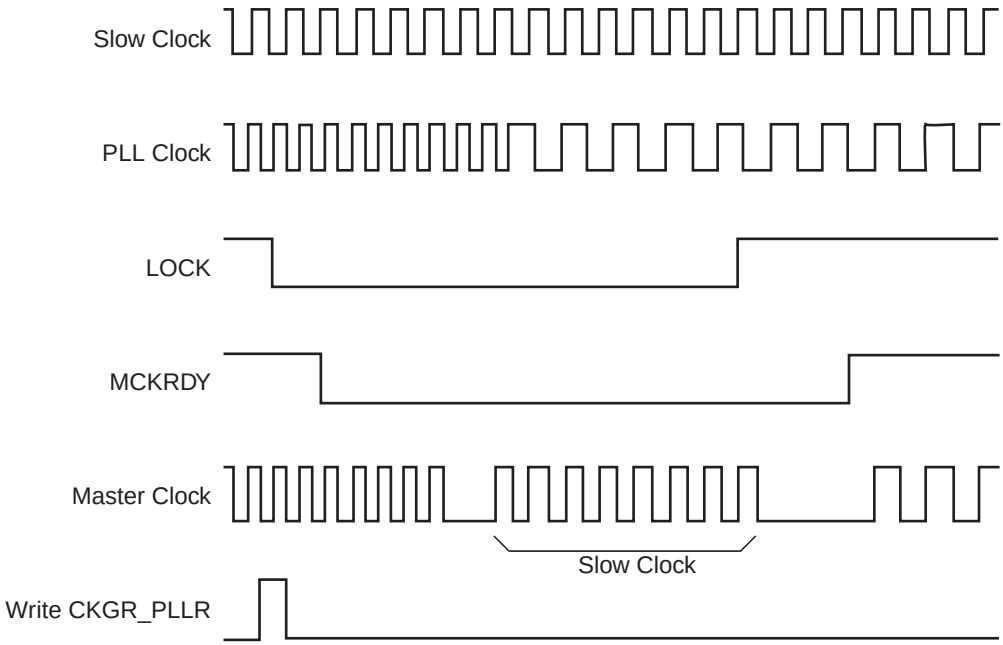
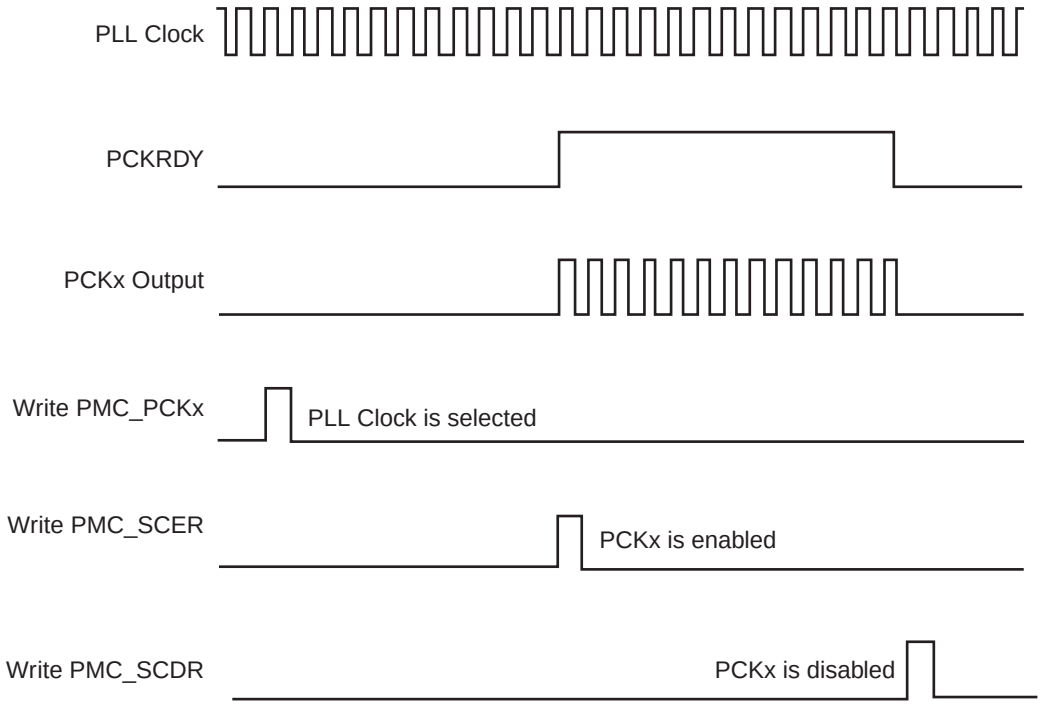


**Figure 25-7. Programmable Clock Output Programming**



## 25.14 Write Protection Registers

To prevent any single software error that may corrupt PMC behavior, certain address spaces can be write protected by setting the WPEN bit in the "PMC Write Protection Mode Register" (PMC_WPMR).

If a write access to the protected registers is detected, then the WPVS flag in the PMC Write Protect Status Register (PMC_WPSR) is set and the field WPVSRC indicates in which register the write access has been attempted.

The WPVS flag is reset by writing the PMC Write Protect Mode Register (PMC_WPMR) with the appropriate access key, WPKEY.

The protected registers are:

"PMC System Clock Enable Register" on page 346

"PMC System Clock Disable Register" on page 347

"PMC Peripheral Clock Enable Register" on page 349

"PMC Peripheral Clock Disable Register" on page 350

"PMC Clock Generator Main Oscillator Register" on page 352

"PMC Clock Generator PLL Register" on page 355

"PMC Master Clock Register" on page 356

"PMC Programmable Clock Register" on page 357

"PMC Fast Startup Mode Register" on page 363

"PMC Fast Startup Polarity Register" on page 364

"PMC Oscillator Calibration Register" on page 368

## 25.15 Power Management Controller (PMC) User Interface

**Table 25-2. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x0000 | System Clock Enable Register | PMC_SCER | Write-only | – |
| 0x0004 | System Clock Disable Register | PMC_SCDR | Write-only | – |
| 0x0008 | System Clock Status Register | PMC_SCSR | Read-only | 0x0000_0001 |
| 0x000C | Reserved | – | – | – |
| 0x0010 | Peripheral Clock Enable Register | PMC_PCER | Write-only | – |
| 0x0014 | Peripheral Clock Disable Register | PMC_PCDR | Write-only | – |
| 0x0018 | Peripheral Clock Status Register | PMC_PCSR | Read-only | 0x0000_0000 |
| 0x001C | Reserved | – | – | – |
| 0x0020 | Clock Generator Main Oscillator Register | CKGR_MOR | Read/Write | 0x0000_0001 |
| 0x0024 | Clock Generator Main Clock Frequency Register | CKGR_MCFR | Read-only | 0x0000_0000 |
| 0x0028 | Clock Generator PLL Register | CKGR_PLLR | Read/Write | 0x0000_3F00 |
| 0x002C | Reserved | – | – | – |
| 0x0030 | Master Clock Register | PMC_MCKR | Read/Write | 0x0000_0001 |
| 0x0034–0x003C | Reserved | – | – | – |
| 0x0040 | Programmable Clock 0 Register | PMC_PCK0 | Read/Write | 0x0000_0000 |
| 0x0044 | Programmable Clock 1 Register | PMC_PCK1 | Read/Write | 0x0000_0000 |
| 0x0048 | Programmable Clock 2 Register | PMC_PCK2 | Read/Write | 0x0000_0000 |
| 0x004C–0x005C | Reserved | – | – | – |
| 0x0060 | Interrupt Enable Register | PMC_IER | Write-only | – |
| 0x0064 | Interrupt Disable Register | PMC_IDR | Write-only | – |
| 0x0068 | Status Register | PMC_SR | Read-only | 0x0001_0008 |
| 0x006C | Interrupt Mask Register | PMC_IMR | Read-only | 0x0000_0000 |
| 0x0070 | Fast Startup Mode Register | PMC_FSMR | Read/Write | 0x0000_0000 |
| 0x0074 | Fast Startup Polarity Register | PMC_FSPR | Read/Write | 0x0000_0000 |
| 0x0078 | Fault Output Clear Register | PMC_FOCR | Write-only | – |
| 0x007C–0x00E0 | Reserved | – | – | – |
| 0x00E4 | Write Protection Mode Register | PMC_WPMR | Read/Write | 0x0000_0000 |
| 0x00E8 | Write Protection Status Register | PMC_WPSR | Read-only | 0x0000_0000 |
| 0x00EC–0x010C | Reserved | – | – | – |
| 0x0110 | Oscillator Calibration Register | PMC_OCR | Read/Write | 0x0040_4040 |

Note: If an offset is not listed in the table it must be considered as “reserved”.

### 25.15.1 PMC System Clock Enable Register

**Name:** PMC_SCER

**Address:** 0x400E0400

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | PCK2 | PCK1 | PCK0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | – |

This register can only be written if the WPEN bit is cleared in “PMC Write Protection Mode Register” on page 366.

- **PCKx: Programmable Clock x Output Enable**

0: No effect.

1: Enables the corresponding Programmable Clock output.

### 25.15.2 PMC System Clock Disable Register

**Name:** PMC_SCDR

**Address:** 0x400E0404

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | PCK2 | PCK1 | PCK0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | – |

This register can only be written if the WPEN bit is cleared in “PMC Write Protection Mode Register” on page 366.

- **PCKx: Programmable Clock x Output Disable**

0: No effect.

1: Disables the corresponding Programmable Clock output.

### 25.15.3 PMC System Clock Status Register

**Name:** PMC_SCSR

**Address:** 0x400E0408

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | PCK2 | PCK1 | PCK0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | – |

- **PCKx: Programmable Clock x Output Status**

0: The corresponding Programmable Clock output is disabled.

1: The corresponding Programmable Clock output is enabled.

### 25.15.4 PMC Peripheral Clock Enable Register

Name: PMC_PCER

**Address:** 0x400E0410

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| PID31 | PID30 | PID29 | PID28 | PID27 | PID26 | PID25 | PID24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| PID23 | PID22 | PID21 | PID20 | PID19 | PID18 | PID17 | PID16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| PID15 | PID14 | PID13 | PID12 | PID11 | PID10 | PID9 | PID8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PID7 | PID6 | PID5 | PID4 | PID3 | PID2 | – | – |

This register can only be written if the WPEN bit is cleared in “PMC Write Protection Mode Register” on page 366.

- **PIDx: Peripheral Clock x Enable**

0: No effect.

1: Enables the corresponding peripheral clock.

Note: PID2 to PID31 refer to identifiers as defined in the section “Peripheral Identifiers” in the product datasheet.

Note: Programming the control bits of the Peripheral ID that are not implemented has no effect on the behavior of the PMC.

### 25.15.5 PMC Peripheral Clock Disable Register

**Name:** PMC_PCDR

**Address:** 0x400E0414

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| PID31 | PID30 | PID29 | PID28 | PID27 | PID26 | PID25 | PID24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| PID23 | PID22 | PID21 | PID20 | PID19 | PID18 | PID17 | PID16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| PID15 | PID14 | PID13 | PID12 | PID11 | PID10 | PID9 | PID8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PID7 | PID6 | PID5 | PID4 | PID3 | PID2 | - | - |

This register can only be written if the WPEN bit is cleared in “PMC Write Protection Mode Register” on page 366.

- **PIDx: Peripheral Clock x Disable**

0: No effect.

1: Disables the corresponding peripheral clock.

Note: PID2 to PID31 refer to identifiers as defined in the section “Peripheral Identifiers” in the product datasheet.

### 25.15.6 PMC Peripheral Clock Status Register

**Name:** PMC_PCSR

**Address:** 0x400E0418

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| PID31 | PID30 | PID29 | PID28 | PID27 | PID26 | PID25 | PID24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| PID23 | PID22 | PID21 | PID20 | PID19 | PID18 | PID17 | PID16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| PID15 | PID14 | PID13 | PID12 | PID11 | PID10 | PID9 | PID8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PID7 | PID6 | PID5 | PID4 | PID3 | PID2 | – | – |

- **PIDx: Peripheral Clock x Status**

0: The corresponding peripheral clock is disabled.

1: The corresponding peripheral clock is enabled.

Note: PID2 to PID31 refer to identifiers as defined in the section “Peripheral Identifiers” in the product datasheet.

### 25.15.7 PMC Clock Generator Main Oscillator Register

**Name:** CKGR_MOR

**Address:** 0x400E0420

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | CFDEN | MOSCSEL |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| KEY | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| MOSCXTST | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | MOSCRCF | | | MOSCRCEN | WAITMODE | MOSCXTBY | MOSCXTEN |

This register can only be written if the WPEN bit is cleared in “PMC Write Protection Mode Register” on page 366.

- **KEY: Password**

Should be written at value 0x37. Writing any other value in this field aborts the write operation.

- **MOSCXTEN: Main Crystal Oscillator Enable**

A crystal must be connected between XIN and XOUT.

0: The Main Crystal Oscillator is disabled.

1: The Main Crystal Oscillator is enabled. MOSCXTBY must be set to 0.

When MOSCXTEN is set, the MOSCXTS flag is set once the Main Crystal Oscillator startup time is achieved.

- **MOSCXTBY: Main Crystal Oscillator Bypass**

0: No effect.

1: The Main Crystal Oscillator is bypassed. MOSCXTEN must be set to 0. An external clock must be connected on XIN.

When MOSCXTBY is set, the MOSCXTS flag in PMC_SR is automatically set.

Clearing MOSCXTEN and MOSCXTBY bits allows resetting the MOSCXTS flag.

- **WAITMODE: Wait Mode Command**

0: No effect.

1: Enters the device in Wait mode.

Note: The bit WAITMODE is write-only

- **MOSCRCEN: Main On-Chip RC Oscillator Enable**

0: The Main On-Chip RC Oscillator is disabled.

1: The Main On-Chip RC Oscillator is enabled.

When MOSCRCEN is set, the MOSCRCS flag is set once the Main On-Chip RC Oscillator startup time is achieved.

- **MOSCRCF: Main On-Chip RC Oscillator Frequency Selection**

| Value | Name | Description |
|---|---|---|
| 0x0 | 4MHZ | The Fast RC Oscillator Frequency is at 4 MHz (default) |
| 0x1 | 8MHZ | The Fast RC Oscillator Frequency is at 8 MHz |
| 0x2 | 12MHZ | The Fast RC Oscillator Frequency is at 12 MHz |

- **MOSCXTST: Main Crystal Oscillator Start-up Time**

Specifies the number of Slow Clock cycles multiplied by 8 for the Main Crystal Oscillator start-up time.

- **MOSCSEL: Main Oscillator Selection**

0: The Main On-Chip RC Oscillator is selected.

1: The Main Crystal Oscillator is selected.

- **CFDEN: Clock Failure Detector Enable**

0: The Clock Failure Detector is disabled.

1: The Clock Failure Detector is enabled.

### 25.15.8 PMC Clock Generator Main Clock Frequency Register

**Name:** CKGR_MCFR

**Address:** 0x400E0424

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | MAINFRDY |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| MAINF | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| MAINF | | | | | | | |

This register can only be written if the WPEN bit is cleared in “PMC Write Protection Mode Register” on page 366.

- **MAINF: Main Clock Frequency**

Gives the number of Main Clock cycles within 16 Slow Clock periods.

- **MAINFRDY: Main Clock Ready**

0: MAINF value is not valid or the Main Oscillator is disabled.

1: The Main Oscillator has been enabled previously and MAINF value is available.

### 25.15.9 PMC Clock Generator PLL Register

**Name:** CKGR_PLLR

**Address:** 0x400E0428

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | 1 | – | – | MUL | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| MUL | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | PLLCOUNT | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| DIV | | | | | | | |

Possible limitations on PLL input frequencies and multiplier factors should be checked before using the PMC.

**Warning:** Bit 29 must always be set to 1 when programming the CKGR_PLLR.

This register can only be written if the WPEN bit is cleared in “PMC Write Protection Mode Register” on page 366.

- **DIV: Divider**

| DIV | Divider Selected |
|---|---|
| 0 | Divider output is 0 |
| 1 | Divider is bypassed (DIV = 1) |
| 2–255 | Divider output is DIV |

- **PLLCOUNT: PLL Counter**

Specifies the number of Slow Clock cycles x8 before the LOCK bit is set in PMC_SR after CKGR_PLLR is written.

- **MUL: PLL Multiplier**

0: The PLL is deactivated.

1–2047: The PLL Clock frequency is the PLL input frequency multiplied by MUL + 1.

### 25.15.10 PMC Master Clock Register

**Name:** PMC_MCKR

**Address:** 0x400E0430

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | PLLDIV2 | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | PRES | | | – | – | CSS | |

This register can only be written if the WPEN bit is cleared in “PMC Write Protection Mode Register” on page 366.

- **CSS: Master Clock Source Selection**

| Value | Name | Description |
|---|---|---|
| 0 | SLOW_CLK | Slow Clock is selected |
| 1 | MAIN_CLK | Main Clock is selected |
| 2 | PLL_CLK | PLL Clock is selected |
| 3 | – | Reserved |

- **PRES: Processor Clock Prescaler**

| Value | Name | Description |
|---|---|---|
| 0 | CLK | Selected clock |
| 1 | CLK_2 | Selected clock divided by 2 |
| 2 | CLK_4 | Selected clock divided by 4 |
| 3 | CLK_28 | 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 |

- **PLLDIV2: PLL Divisor by 2**

| PLLDIV2 | PLL Clock Division |
|---|---|
| 0 | PLL clock frequency is divided by 1 |
| 1 | PLL clock frequency is divided by 2 |

### 25.15.11 PMC Programmable Clock Register

**Name:** PMC_PCKx

**Address:** 0x400E0440

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | PRES | | | – | CSS | | |

This register can only be written if the WPEN bit is cleared in “PMC Write Protection Mode Register” on page 366.

- **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: Processor Clock Prescaler**

| Value | Name | Description |
|---|---|---|
| 0 | CLK | Selected clock |
| 1 | CLK_2 | Selected clock divided by 2 |
| 2 | CLK_4 | Selected clock divided by 4 |
| 3 | CLK_28 | 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 |

### 25.15.12 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 |
| – | – | – | – | – | CFDEV | MOSCRCS | MOSCSELS |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | PCKRDY2 | PCKRDY1 | PCKRDY0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | MCKRDY | – | LOCK | MOSCXTS |

- **MOSCXTS: Main Crystal Oscillator Status Interrupt Enable**
- **LOCK: PLL Lock Interrupt Enable**
- **MCKRDY: Master Clock Ready Interrupt Enable**
- **PCKRDYx: Programmable Clock Ready x Interrupt Enable**
- **MOSCSELS: Main Oscillator Selection Status Interrupt Enable**
- **MOSCRCS: Main On-Chip RC Status Interrupt Enable**
- **CFDEV: Clock Failure Detector Event Interrupt Enable**

### 25.15.13 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 |
| – | – | – | – | – | CFDEV | MOSCRCS | MOSCSELS |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | PCKRDY2 | PCKRDY1 | PCKRDY0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | MCKRDY | – | LOCK | MOSCXTS |

- **MOSCXTS: Main Crystal Oscillator Status Interrupt Disable**
- **LOCK: PLL Lock Interrupt Disable**
- **MCKRDY: Master Clock Ready Interrupt Disable**
- **PCKRDYx: Programmable Clock Ready x Interrupt Disable**
- **MOSCSELS: Main Oscillator Selection Status Interrupt Disable**
- **MOSCRCS: Main On-Chip RC Status Interrupt Disable**
- **CFDEV: Clock Failure Detector Event Interrupt Disable**

### 25.15.14 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 |
| – | – | – | FOS | CFDS | CFDEV | MOSCRCS | MOSCSELS |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | PCKRDY2 | PCKRDY1 | PCKRDY0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| OSCSELS | – | – | – | MCKRDY | – | LOCK | MOSCXTS |

- **MOSCXTS: Main XTAL Oscillator Status**

0: Main XTAL oscillator is not stabilized.

1: Main XTAL oscillator is stabilized.

- **LOCK: PLL Lock Status**

0: PLL is not locked

1: PLL is locked.

- **MCKRDY: Master Clock Status**

0: Master Clock is not ready.

1: Master Clock is ready.

- **OSCSELS: Slow Clock Oscillator Selection**

0: Internal slow clock RC oscillator is selected.

1: External slow clock 32 kHz oscillator is selected.

- **PCKRDYx: Programmable Clock Ready Status**

0: Programmable Clock x is not ready.

1: Programmable Clock x is ready.

- **MOSCSELS: Main Oscillator Selection Status**

0: Selection is in progress

1: Selection is done

- **MOSCRCS: Main On-Chip RC Oscillator Status**

0: Main on-chip RC oscillator is not stabilized.

1: Main on-chip RC oscillator is stabilized.

- **CFDEV: Clock Failure Detector Event**

0: No clock failure detection of the main on-chip RC oscillator clock has occurred since the last read of PMC_SR.

1: At least one clock failure detection of the main on-chip RC oscillator clock has occurred since the last read of PMC_SR.

- **CFDS: Clock Failure Detector Status**

0: A clock failure of the main on-chip RC oscillator clock is not detected.

1: A clock failure of the main on-chip RC oscillator clock is detected.

- **FOS: Clock Failure Detector Fault Output Status**

0: The fault output of the clock failure detector is inactive.

1: The fault output of the clock failure detector is active.

### 25.15.15 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 |
| – | – | – | – | – | CFDEV | MOSCRCS | MOSCSELS |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | PCKRDY2 | PCKRDY1 | PCKRDY0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | MCKRDY | – | LOCK | MOSCXTS |

- **MOSCXTS: Main Crystal Oscillator Status Interrupt Mask**
- **LOCK: PLL Lock Interrupt Mask**
- **MCKRDY: Master Clock Ready Interrupt Mask**
- **PCKRDYx: Programmable Clock Ready x Interrupt Mask**
- **MOSCSELS: Main Oscillator Selection Status Interrupt Mask**
- **MOSCRCS: Main On-Chip RC Status Interrupt Mask**
- **CFDEV: Clock Failure Detector Event Interrupt Mask**

### 25.15.16 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 |
| – | – | – | LPM | – | – | RTCAL | RTTAL |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| FSTT15 | FSTT14 | FSTT13 | FSTT12 | FSTT11 | FSTT10 | FSTT9 | FSTT8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| FSTT7 | FSTT6 | FSTT5 | FSTT4 | FSTT3 | FSTT2 | FSTT1 | FSTT0 |

This register can only be written if the WPEN bit is cleared in “PMC Write Protection Mode Register” on page 366.

- **FSTT0–FSTT15: Fast Startup Input Enable 0 to 15**

0: The corresponding wake up input has no effect on the Power Management Controller.

1: The corresponding wake up input enables a fast restart signal to the Power Management Controller.

- **RTTAL: RTT Alarm Enable**

0: The RTT alarm has no effect on the Power Management Controller.

1: The RTT alarm enables a fast restart signal to the Power Management Controller.

- **RTCAL: RTC Alarm Enable**

0: The RTC alarm has no effect on the Power Management Controller.

1: The RTC alarm enables a fast restart signal to the Power Management Controller.

- **LPM: Low Power Mode**

0: The WaitForInterrupt (WFI) or WaitForEvent (WFE) instruction of the processor causes the processor to enter Idle Mode.

1: The WaitForEvent (WFE) instruction of the processor causes the system to enter Wait Mode.

### 25.15.17 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 “PMC Write Protection Mode Register” on page 366.

- **FSTPx: Fast Startup Input Polarityx**

Defines the active polarity of the corresponding wake up input. If the corresponding wake up input is enabled and at the FSTP level, it enables a fast restart signal.

### 25.15.18 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.

### 25.15.19 PMC Write Protection Mode Register

**Name:** PMC_WPMR

**Address:** 0x400E04E4

**Access:** Read/Write

**Reset:** See Table 25-2

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| WPKEY | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| WPKEY | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | WPEN |

- **WPEN: Write Protect Enable**

0: Disables the Write Protect if WPKEY corresponds to 0x504D43 (“PMC” in ASCII).

1: Enables the Write Protect if WPKEY corresponds to 0x504D43 (“PMC” in ASCII).

Protects the registers:

- “PMC System Clock Enable Register” on page 346
- “PMC System Clock Disable Register” on page 347
- “PMC Peripheral Clock Enable Register” on page 349
- “PMC Peripheral Clock Disable Register” on page 350
- “PMC Clock Generator Main Oscillator Register” on page 352
- “PMC Clock Generator PLL Register” on page 355
- “PMC Master Clock Register” on page 356
- “PMC Programmable Clock Register” on page 357
- “PMC Fast Startup Mode Register” on page 363
- “PMC Fast Startup Polarity Register” on page 364
- “PMC Oscillator Calibration Register” on page 368

- **WPKEY: Write Protect Key**

Should be written at value 0x504D43 (“PMC” in ASCII). Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

### 25.15.20 PMC Write Protection Status Register

**Name:** PMC_WPSR

**Address:** 0x400E04E8

**Access:** Read-only

**Reset:** See Table 25-2

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| WPVSRC | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| WPVSRC | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | WPVS |

- **WPVS: Write Protect Violation Status**

0: No Write Protect Violation has occurred since the last read of the PMC_WPSR.

1: A Write Protect 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 Protect Violation Source**

When WPVS is active, this field indicates the write-protected register (through address offset or code) in which a write access has been attempted.

Reading PMC_WPSR automatically clears all fields.

### 25.15.21 PMC Oscillator Calibration Register

**Name:** PMC_OCR

**Address:** 0x400E0510

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| SEL12 | CAL12 | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| SEL8 | CAL8 | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SEL4 | CAL4 | | | | | | |

This register can only be written if the WPEN bit is cleared in “PMC Write Protection Mode Register” on page 366.

- **CAL4: RC Oscillator Calibration bits for 4 MHz**

Calibration bits applied to the RC Oscillator when SEL4 is set.

- **SEL4: Selection of RC Oscillator Calibration bits for 4 MHz**

0: Default value stored in Flash memory.

1: Value written by user in CAL4 field of this register.

- **CAL8: RC Oscillator Calibration bits for 8 MHz**

Calibration bits applied to the RC Oscillator when SEL8 is set.

- **SEL8: Selection of RC Oscillator Calibration bits for 8 MHz**

0: Factory determined value stored in Flash memory.

1: Value written by user in CAL8 field of this register.

- **CAL12: RC Oscillator Calibration bits for 12 MHz**

Calibration bits applied to the RC Oscillator when SEL12 is set.

- **SEL12: Selection of RC Oscillator Calibration bits for 12 MHz**

0: Factory determined value stored in Flash memory.

1: Value written by user in CAL12 field of this register.

# 26. Chip Identifier (CHIPID)

## 26.1 Description

Chip Identifier registers permit recognition of the device and its revision. These registers provide the sizes and types of the on-chip memories, as well as the set of embedded peripherals.

Two chip identifier registers are embedded: CHIPID_CIDR (Chip ID Register) and CHIPID_EXID (Extension ID). Both registers contain a hard-wired value that is read-only. The first register contains the following fields:

- EXT - shows the use of the extension identifier register
- NVPTYP and NVPSIZ - identifies the type of embedded non-volatile memory and its size
- ARCH - identifies the set of embedded peripherals
- SRAMSIZ - indicates the size of the embedded SRAM
- EPROC - indicates the embedded ARM processor
- VERSION - gives the revision of the silicon

The second register is device-dependent and reads 0 if the bit EXT is 0.

**Table 26-1. ATSAM3N Chip IDs Register**

| Chip Name | CHIPID_CIDR | CHIPID_EXID |
|---|---|---|
| ATSAM3N4C (Rev A) | 0x29540960 | 0x0 |
| ATSAM3N2C (Rev A) | 0x29590760 | 0x0 |
| ATSAM3N1C (Rev A) | 0x29580560 | 0x0 |
| ATSAM3N4B (Rev A) | 0x29440960 | 0x0 |
| ATSAM3N2B (Rev A) | 0x29490760 | 0x0 |
| ATSAM3N1B (Rev A) | 0x29480560 | 0x0 |
| ATSAM3N4A (Rev A) | 0x29340960 | 0x0 |
| ATSAM3N2A (Rev A) | 0x29390760 | 0x0 |
| ATSAM3N1A (Rev A) | 0x29380560 | 0x0 |
| ATSAM3N1C (Rev B) | 0x29580561 | 0x0 |
| ATSAM3N1B (Rev B) | 0x29480561 | 0x0 |
| ATSAM3N1A (Rev B) | 0x29380561 | 0x0 |
| ATSAM3N0C (Rev A) | 0x295 80361 | 0x0 |
| ATSAM3N0B (Rev A) | 0x294 80361 | 0x0 |
| ATSAM3N0A (Rev A) | 0x293 80361 | 0x0 |
| ATSAM3N00B (Rev A) | 0x294 50261 | 0x0 |
| ATSAM3N00A (Rev A) | 0x293 50261 | 0x0 |

## 26.2 Chip Identifier (CHIPID) User Interface

**Table 26-2. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x0 | Chip ID Register | CHIPID_CIDR | Read-only | – |
| 0x4 | Chip ID Extension Register | CHIPID_EXID | Read-only | – |

### 26.2.1 Chip ID Register

**Name:** CHIPID_CIDR

**Address:** 0x400E0740

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| EXT | NVPTYP | | | ARCH | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| ARCH | | | | SRAMSIZ | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| NVPSIZ2 | | | | NVPSIZ | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| EPROC | | | VERSION | | | | |

- **VERSION: Version of the Device**

Current version of the device.

- **EPROC: Embedded Processor**

| Value | Name | Description |
|---|---|---|
| 1 | ARM946ES | ARM946ES |
| 2 | ARM7TDMI | ARM7TDMI |
| 3 | CM3 | Cortex-M3 |
| 4 | ARM920T | ARM920T |
| 5 | ARM926EJS | ARM926EJS |
| 6 | CA5 | Cortex-A5 |

- **NVPSIZ: Nonvolatile Program Memory Size**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | None |
| 1 | 8K | 8K bytes |
| 2 | 16K | 16K bytes |
| 3 | 32K | 32K bytes |
| 4 | | Reserved |
| 5 | 64K | 64K bytes |
| 6 | | Reserved |
| 7 | 128K | 128K bytes |
| 8 | | Reserved |
| 9 | 256K | 256K bytes |
| 10 | 512K | 512K bytes |
| 11 | | Reserved |
| 12 | 1024K | 1024K bytes |
| 13 | | Reserved |

| Value | Name | Description |
|---|---|---|
| 14 | 2048K | 2048K bytes |
| 15 | | Reserved |

- **NVPSIZ2 Second Nonvolatile Program Memory Size**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | None |
| 1 | 8K | 8K bytes |
| 2 | 16K | 16K bytes |
| 3 | 32K | 32K bytes |
| 4 | | Reserved |
| 5 | 64K | 64K bytes |
| 6 | | Reserved |
| 7 | 128K | 128K bytes |
| 8 | | Reserved |
| 9 | 256K | 256K bytes |
| 10 | 512K | 512K bytes |
| 11 | | Reserved |
| 12 | 1024K | 1024K bytes |
| 13 | | Reserved |
| 14 | 2048K | 2048K bytes |
| 15 | | Reserved |

- **SRAMSIZ: Internal SRAM Size**

| Value | Name | Description |
|---|---|---|
| 0 | 48K | 48K bytes |
| 1 | 1K | 1K bytes |
| 2 | 2K | 2K bytes |
| 3 | 6K | 6K bytes |
| 4 | 112K | 112K bytes |
| 5 | 4K | 4K bytes |
| 6 | 80K | 80K bytes |
| 7 | 160K | 160K bytes |
| 8 | 8K | 8K bytes |
| 9 | 16K | 16K bytes |
| 10 | 32K | 32K bytes |
| 11 | 64K | 64K bytes |
| 12 | 128K | 128K bytes |
| 13 | 256K | 256K bytes |

| Value | Name | Description |
|---|---|---|
| 14 | 96K | 96K bytes |
| 15 | 512K | 512K bytes |

- **ARCH: Architecture Identifier**

| Value | Name | Description |
|---|---|---|
| 0x19 | AT91SAM9xx | AT91SAM9xx Series |
| 0x29 | AT91SAM9XExx | AT91SAM9XExx Series |
| 0x34 | AT91x34 | AT91x34 Series |
| 0x37 | CAP7 | CAP7 Series |
| 0x39 | CAP9 | CAP9 Series |
| 0x3B | CAP11 | CAP11 Series |
| 0x40 | AT91x40 | AT91x40 Series |
| 0x42 | AT91x42 | AT91x42 Series |
| 0x55 | AT91x55 | AT91x55 Series |
| 0x60 | AT91SAM7Axx | AT91SAM7Axx Series |
| 0x61 | AT91SAM7AQxx | AT91SAM7AQxx Series |
| 0x63 | AT91x63 | AT91x63 Series |
| 0x70 | AT91SAM7Sxx | AT91SAM7Sxx Series |
| 0x71 | AT91SAM7XCxx | AT91SAM7XCxx Series |
| 0x72 | AT91SAM7SExx | AT91SAM7SExx Series |
| 0x73 | AT91SAM7Lxx | AT91SAM7Lxx Series |
| 0x75 | AT91SAM7Xxx | AT91SAM7Xxx Series |
| 0x76 | AT91SAM7SLxx | AT91SAM7SLxx Series |
| 0x80 | ATSAM3UxC | ATSAM3UxC Series (100-pin version) |
| 0x81 | ATSAM3UxE | ATSAM3UxE Series (144-pin version) |
| 0x83 | ATSAM3AxC | ATSAM3AxC Series (100-pin version) |
| 0x84 | ATSAM3XxC | ATSAM3XxC Series (100-pin version) |
| 0x85 | ATSAM3XxE | ATSAM3XxE Series (144-pin version) |
| 0x86 | ATSAM3XxG | ATSAM3XxG Series (208/217-pin version) |
| 0x88 | ATSAM3SxA | ATSAM3SxA Series (48-pin version) |
| 0x89 | ATSAM3SxB | ATSAM3SxB Series (64-pin version) |
| 0x8A | ATSAM3SxC | ATSAM3SxC Series (100-pin version) |
| 0x92 | AT91x92 | AT91x92 Series |
| 0x93 | ATSAM3NxA | ATSAM3NxA Series (48-pin version) |
| 0x94 | ATSAM3NxB | ATSAM3NxB Series (64-pin version) |
| 0x95 | ATSAM3NxC | ATSAM3NxC Series (100-pin version) |
| 0x98 | ATSAM3SDxA | ATSAM3SDxA Series (48-pin version) |
| 0x99 | ATSAM3SDxB | ATSAM3SDxB Series (64-pin version) |

| Value | Name | Description |
|---|---|---|
| 0x9A | ATSAM3SDxC | ATSAM3SDxC Series (100-pin version) |
| 0xA5 | ATSAM5A | ATSAM5A |
| 0xF0 | AT75Cxx | AT75Cxx Series |

- **NVPTYP: Nonvolatile Program Memory Type**

| Value | Name | Description |
|---|---|---|
| 0 | ROM | ROM |
| 1 | ROMLESS | ROMless or on-chip Flash |
| 4 | SRAM | SRAM emulating ROM |
| 2 | FLASH | Embedded Flash Memory |
| 3 | ROM_FLASH | ROM and Embedded Flash Memory<br>NVPSIZ is ROM size<br>NVPSIZ2 is Flash size |

- **EXT: Extension Flag**

0 = Chip ID has a single register definition without extension

1 = An extended Chip ID exists.

### 26.2.2 Chip ID Extension Register

**Name:** CHIPID_EXID

**Address:** 0x400E0744

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| EXID | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| EXID | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| EXID | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| EXID | | | | | | | |

- **EXID: Chip ID Extension**

Reads 0 if the bit EXT in CHIPID_CIDR is 0.

# 27. Parallel Input/Output (PIO) Controller

## 27.1 Description

The Parallel Input/Output Controller (PIO) manages up to 32 fully programmable input/output lines. Each I/O line may be dedicated as a general-purpose I/O or be assigned to a function of an embedded peripheral. This assures effective optimization of the pins of a product.

Each I/O line is associated with a bit number in all of the 32-bit registers of the 32-bit wide User Interface.

Each I/O line of the PIO Controller features:

- An input change interrupt enabling level change detection on any I/O line.
- Additional Interrupt modes enabling rising edge, falling edge, low level or high level detection on any I/O line.
- A glitch filter providing rejection of glitches lower than one-half of PIO clock cycle.
- A debouncing filter providing rejection of unwanted pulses from key or push button operations.
- Multi-drive capability similar to an open drain I/O line.
- Control of the pull-up and pull-down of the I/O line.
- Input visibility and output control.

The PIO Controller also features a synchronous output providing up to 32 bits of data output in a single write operation.

## 27.2 Embedded Characteristics

- Up to 32 Programmable I/O Lines
- Fully Programmable through Set/Clear Registers
- Multiplexing of Four Peripheral Functions per I/O Line
- For each I/O Line (Whether Assigned to a Peripheral or Used as General Purpose I/O)
  - Input Change Interrupt
  - Programmable Glitch Filter
  - Programmable Debouncing Filter
  - Multi-drive Option Enables Driving in Open Drain
  - Programmable Pull Up on Each I/O Line
  - Pin Data Status Register, Supplies Visibility of the Level on the Pin at Any Time
  - Additional Interrupt Modes on a Programmable Event: Rising Edge, Falling Edge, Low Level or High Level
  - Lock of the Configuration by the Connected Peripheral
- Synchronous Output, Provides Set and Clear of Several I/O lines in a Single Write
- Write Protect Registers
- Programmable Schmitt Trigger Inputs

## 27.3 Block Diagram

**Figure 27-1. Block Diagram**

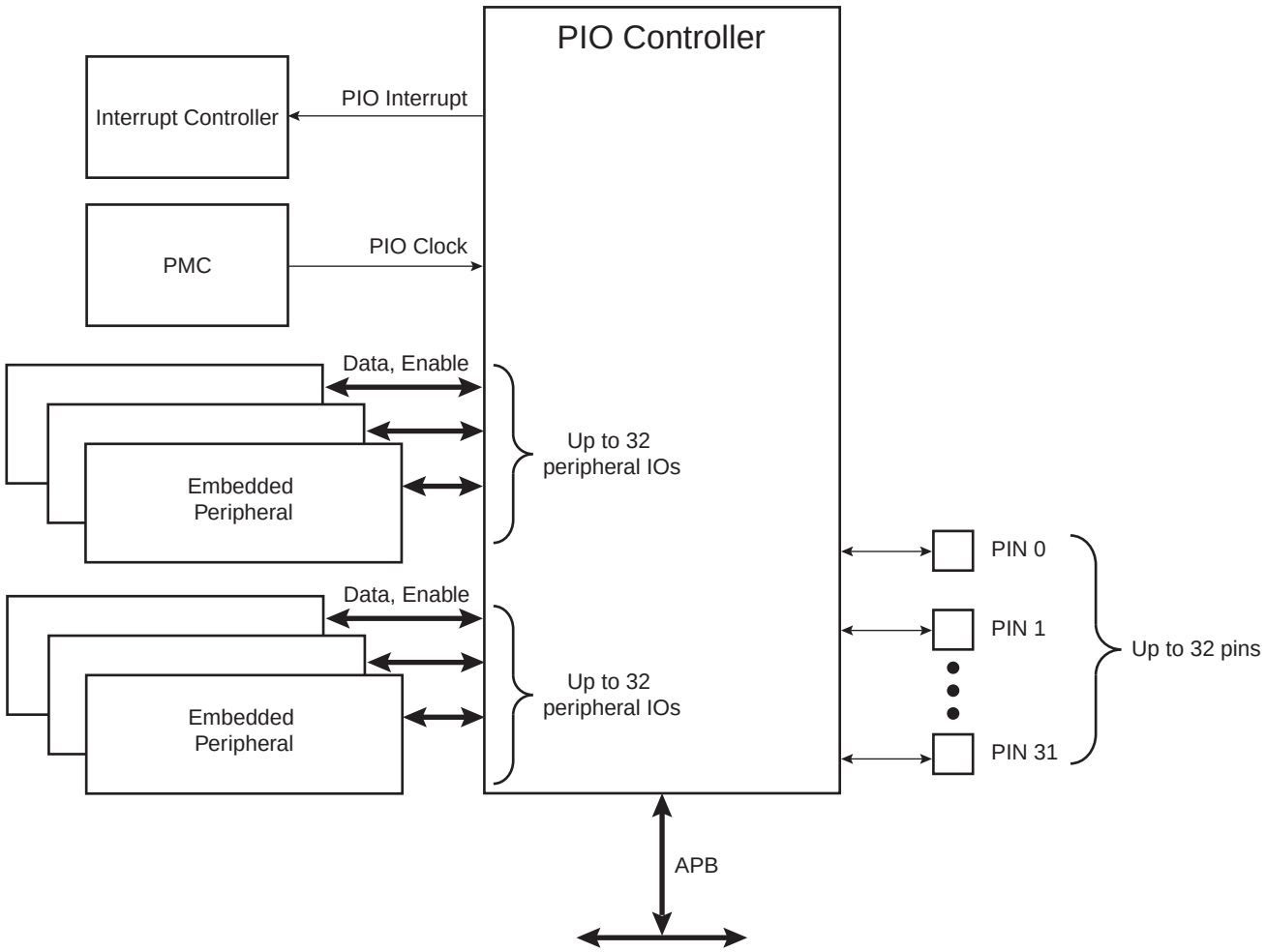
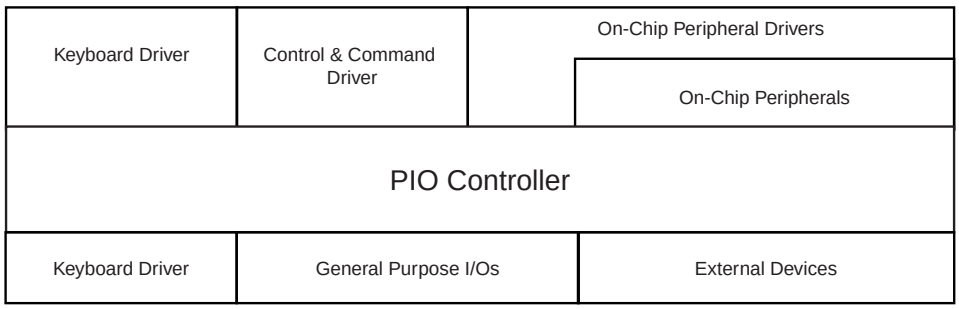



**Figure 27-2. Application Block Diagram**



## 27.4 Product Dependencies

### 27.4.1 Pin Multiplexing

Each pin is configurable, according to product definition as either a general-purpose I/O line only, or as an I/O line multiplexed with one or two peripheral I/Os. As the multiplexing is hardware defined and thus product-dependent, the hardware designer and programmer must carefully determine the configuration of the PIO controllers required by their application. When an I/O line is general-purpose only, i.e. not multiplexed with any peripheral I/O, programming of the PIO Controller regarding the assignment to a peripheral has no effect and only the PIO Controller can control how the pin is driven by the product.

### 27.4.2 Power Management

The Power Management Controller controls the PIO Controller clock in order to save power. Writing any of the registers of the user interface does not require the PIO Controller clock to be enabled. This means that the configuration of the I/O lines does not require the PIO Controller clock to be enabled.

However, when the clock is disabled, not all of the features of the PIO Controller are available, including glitch filtering. Note that the Input Change Interrupt, Interrupt Modes on a programmable event and the read of the pin level require the clock to be validated.

After a hardware reset, the PIO clock is disabled by default.

The user must configure the Power Management Controller before any access to the input line information.

### 27.4.3 Interrupt Generation

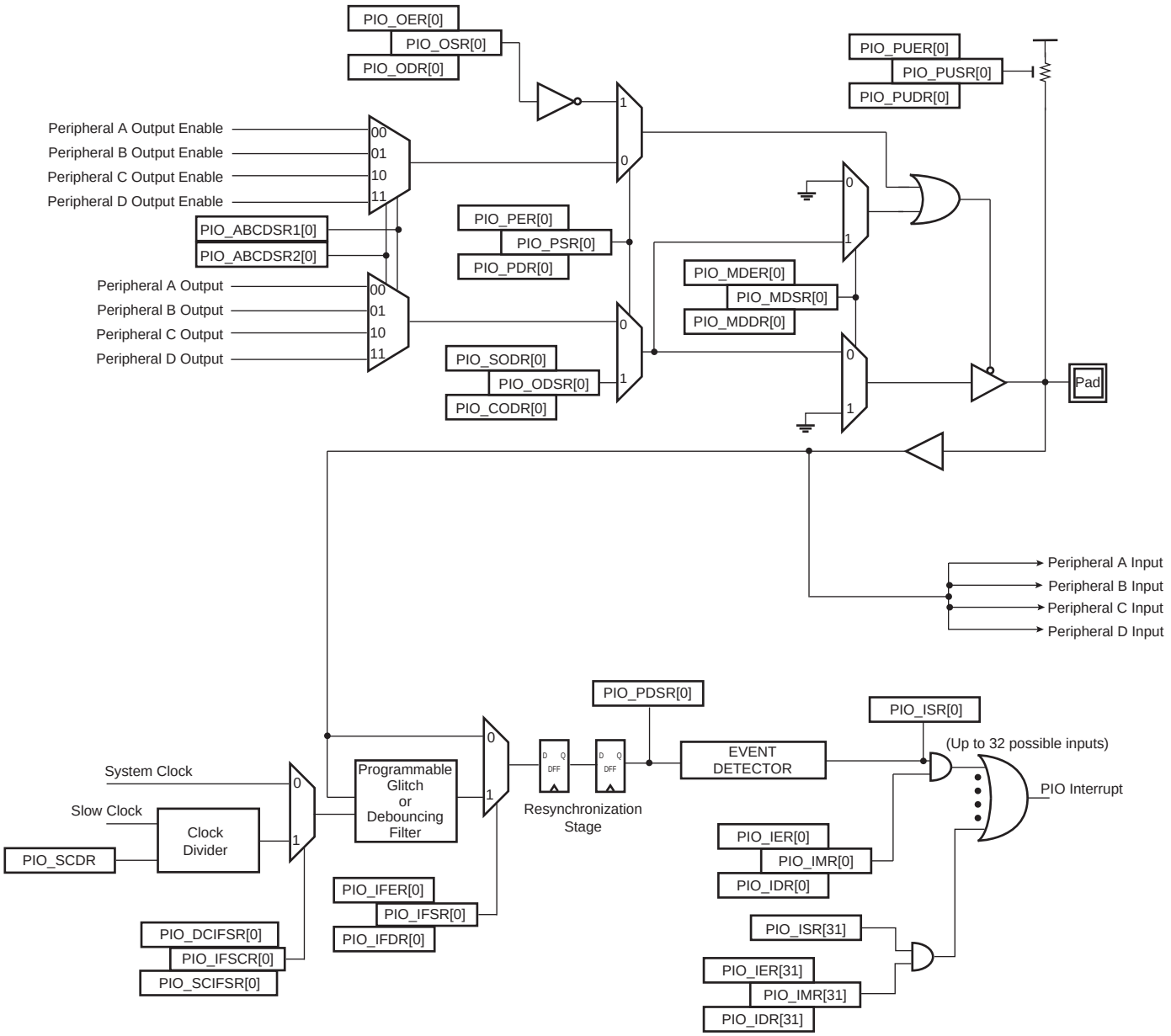
The PIO Controller is connected on one of the sources of the Nested Vectored Interrupt Controller (NVIC). Using the PIO Controller requires the NVIC to be programmed first.

The PIO Controller interrupt can be generated only if the PIO Controller clock is enabled.

## 27.5 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 27-3. In this description each signal shown represents but one of up to 32 possible indexes.

**Figure 27-3. I/O Line Control Logic**



### 27.5.1 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 respectively PIO_PUER (Pull-up Enable Register) and PIO_PUDR (Pull-up Disable Resistor). Writing in these registers results in setting or clearing the corresponding bit in PIO_PUSR (Pull-up Status Register). Reading a 1 in PIO_PUSR means the pull-up is disabled and reading a 0 means the pull-up is enabled. The pull-down resistor can be enabled or disabled by writing respectively PIO_PPDER (Pull-down Enable Register) and PIO_PPDDR (Pull-down Disable Resistor). Writing in these registers results in setting or clearing the corresponding bit in PIO_PPDSR (Pull-down Status Register). Reading a 1 in PIO_PPDSR means the pull-up is disabled and reading a 0 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 concerned 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 concerned I/O line is discarded.

Control of the pull-up resistor is possible regardless of the configuration of the I/O line.

After reset, all of the pull-ups are enabled, i.e. PIO_PUSR resets at the value 0x0, and all the pull-downs are disabled, i.e. PIO_PPDSR resets at the value 0xFFFFFFFF.

### 27.5.2 I/O Line or Peripheral Function Selection

When a pin is multiplexed with one or two peripheral functions, the selection is controlled with the registers PIO_PER (PIO Enable Register) and PIO_PDR (PIO Disable Register). The register PIO_PSR (PIO Status Register) is the result of the set and clear registers and indicates whether the pin is controlled by the corresponding peripheral or by the PIO Controller. A value of 0 indicates that the pin is controlled by the corresponding on-chip peripheral selected in the PIO_ABCDSR1 and PIO_ABCDSR2 (ABCD Select Registers). A value of 1 indicates the pin is controlled by the PIO controller.

If a pin is used as a general purpose I/O line (not multiplexed with an on-chip peripheral), PIO_PER and PIO_PDR have no effect and PIO_PSR returns 1 for the corresponding bit.

After reset, most generally, the I/O lines are controlled by the PIO controller, i.e. PIO_PSR resets at 1. However, in some events, it is important that PIO lines are controlled by the peripheral (as in the case of memory chip select lines that must be driven inactive after reset or for address lines that must be driven low for booting out of an external memory). Thus, the reset value of PIO_PSR is defined at the product level, depending on the multiplexing of the device.

### 27.5.3 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 (ABCD Select Registers).

For each pin:

- the corresponding bit at level 0 in PIO_ABCDSR1 and the corresponding bit at level 0 in PIO_ABCDSR2 means peripheral A is selected.
- the corresponding bit at level 1 in PIO_ABCDSR1 and the corresponding bit at level 0 in PIO_ABCDSR2 means peripheral B is selected.
- the corresponding bit at level 0 in PIO_ABCDSR1 and the corresponding bit at level 1 in PIO_ABCDSR2 means peripheral C is selected.
- the corresponding bit at level 1 in PIO_ABCDSR1 and the corresponding bit at level 1 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.

After reset, PIO_ABCDSR1 and PIO_ABCDSR2 are 0, thus indicating that all the PIO lines are configured on peripheral A. However, peripheral A generally does not drive the pin as the PIO Controller resets in I/O line mode.

Writing in PIO_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 the peripheral selection registers (PIO_ABCDSR1 and PIO_ABCDSR2) in addition to a write in PIO_PDR.

### 27.5.4 Output Control

When the I/0 line is assigned to a peripheral function, i.e. the corresponding bit in PIO_PSR is at 0, the drive of the I/O line is controlled by the peripheral. Peripheral A or B or C or D depending on the value in PIO_ABCDSR1 and PIO_ABCDSR2 (ABCD Select Registers) determines whether the pin is driven or not.

When the I/O line is controlled by the PIO controller, the pin can be configured to be driven. This is done by writing PIO_OER (Output Enable Register) and PIO_ODR (Output Disable Register). The results of these write operations are detected in PIO_OSR (Output Status Register). When a bit in this register is at 0, the corresponding I/O line is used as an input only. When the bit is at 1, the corresponding I/O line is driven by the PIO controller.

The level driven on an I/O line can be determined by writing in PIO_SODR (Set Output Data Register) and PIO_CODR (Clear Output Data Register). These write operations respectively set and clear PIO_ODSR (Output Data Status Register), which represents the data driven on the I/O lines. Writing in PIO_OER and PIO_ODR manages PIO_OSR whether the pin is configured to be controlled by the PIO controller or assigned to a peripheral function. This enables configuration of the I/O line prior to setting it to be managed by the PIO Controller.

Similarly, writing in PIO_SODR and PIO_CODR effects PIO_ODSR. This is important as it defines the first level driven on the I/O line.

### 27.5.5 Synchronous Data Output

Clearing one (or more) PIO line(s) and setting another one (or more) PIO line(s) synchronously cannot be done by using PIO_SODR and PIO_CODR registers. It requires two successive write operations into two different registers. To overcome this, the PIO Controller offers a direct control of PIO outputs by single write access to PIO_ODSR (Output Data Status Register).Only bits unmasked by PIO_OWSR (Output Write Status Register) are written. The mask bits in PIO_OWSR are set by writing to PIO_OWER (Output Write Enable Register) and cleared by writing to PIO_OWDR (Output Write Disable Register).

After reset, the synchronous data output is disabled on all the I/O lines as PIO_OWSR resets at 0x0.

### 27.5.6 Multi Drive Control (Open Drain)

Each I/O can be independently programmed in Open Drain by using the Multi Drive feature. This feature permits several drivers to be connected on the I/O line which is driven low only by each device. An external pull-up resistor (or enabling of the internal one) is generally required to guarantee a high level on the line.

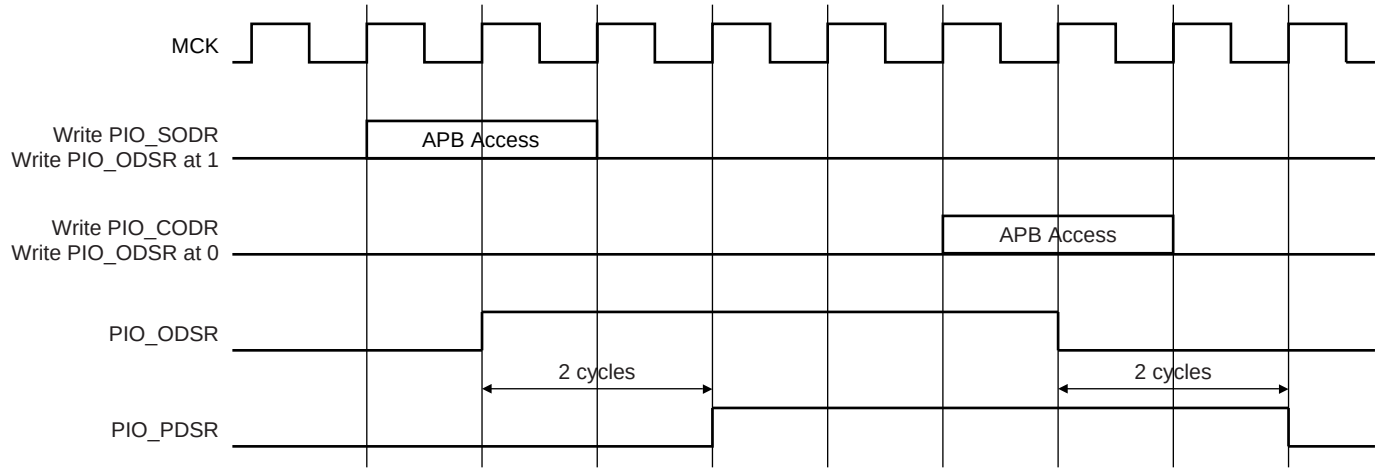
The Multi Drive feature is controlled by PIO_MDER (Multi-driver Enable Register) and PIO_MDDR (Multi-driver Disable Register). The Multi Drive can be selected whether the I/O line is controlled by the PIO controller or assigned to a peripheral function. PIO_MDSR (Multi-driver Status Register) indicates the pins that are configured to support external drivers.

After reset, the Multi Drive feature is disabled on all pins, i.e. PIO_MDSR resets at value 0x0.

### 27.5.7 Output Line Timings

Figure 27-4 shows how the outputs are driven either by writing PIO_SODR or PIO_CODR, or by directly writing PIO_ODSR. This last case is valid only if the corresponding bit in PIO_OWSR is set. Figure 27-4 also shows when the feedback in PIO_PDSR is available.

**Figure 27-4. Output Line Timings**




### 27.5.8 Inputs

The level on each I/O line can be read through PIO_PDSR (Pin Data Status Register). This register indicates the level of the I/O lines regardless of their configuration, whether uniquely as an input or driven by the PIO controller or driven by a peripheral.

Reading the I/O line levels requires the clock of the PIO controller to be enabled, otherwise PIO_PDSR reads the levels present on the I/O line at the time the clock was disabled.

### 27.5.9 Input Glitch and Debouncing Filters

Optional input glitch and debouncing filters are independently programmable on each I/O line.

The glitch filter can filter a glitch with a duration of less than 1/2 Master Clock (MCK) and the debouncing filter can filter a pulse of less than 1/2 Period of a Programmable Divided Slow Clock.

The selection between glitch filtering or debounce filtering is done by writing in the registers PIO_IFSCDR (PIO Input Filter Slow Clock Disable Register) and PIO_IFSCER (PIO Input Filter Slow Clock Enable Register). Writing PIO_IFSCDR and PIO_IFSCER respectively, sets and clears bits in PIO_IFSCSR.

The current selection status can be checked by reading the register PIO_IFSCSR (Input Filter Slow Clock Status Register).

- If PIO_IFSCSR[i] = 0: The glitch filter can filter a glitch with a duration of less than 1/2 Period of Master Clock.
- If PIO_IFSCSR[i] = 1: The debouncing filter can filter a pulse with a duration of less than 1/2 Period of the Programmable Divided Slow Clock.

For the debouncing filter, the Period of the Divided Slow Clock is performed by writing in the DIV field of the PIO_SCDR (Slow Clock Divider Register)

Tdiv_slclk = ((DIV+1)*2).Tslow_clock

When the glitch or debouncing filter is enabled, a glitch or pulse with a duration of less than 1/2 Selected Clock Cycle (Selected Clock represents MCK or Divided Slow Clock depending on PIO_IFSCDR and PIO_IFSCER programming) is automatically rejected, while a pulse with a duration of 1 Selected Clock (MCK or Divided Slow Clock) cycle or more is accepted. For pulse durations between 1/2 Selected Clock cycle and 1 Selected Clock cycle the pulse may or may not be taken into account, depending on the precise timing of its occurrence. Thus for a pulse to be visible it must exceed 1 Selected Clock cycle, whereas for a glitch to be reliably filtered out, its duration must not exceed 1/2 Selected Clock cycle.

The filters also introduce some latencies, this is illustrated in Figure 27-5 and Figure 27-6.

The glitch filters are controlled by the register set: PIO_IFER (Input Filter Enable Register), PIO_IFDR (Input Filter Disable Register) and PIO_IFSR (Input Filter Status Register). Writing PIO_IFER and PIO_IFDR respectively sets and clears bits in PIO_IFSR. This last register enables the glitch filter on the I/O lines.

When the glitch and/or debouncing filter is enabled, it does not modify the behavior of the inputs on the peripherals. It acts only on the value read in PIO_PDSR and on the input change interrupt detection. The glitch and debouncing filters require that the PIO Controller clock is enabled.

**Figure 27-5. Input Glitch Filter Timing**

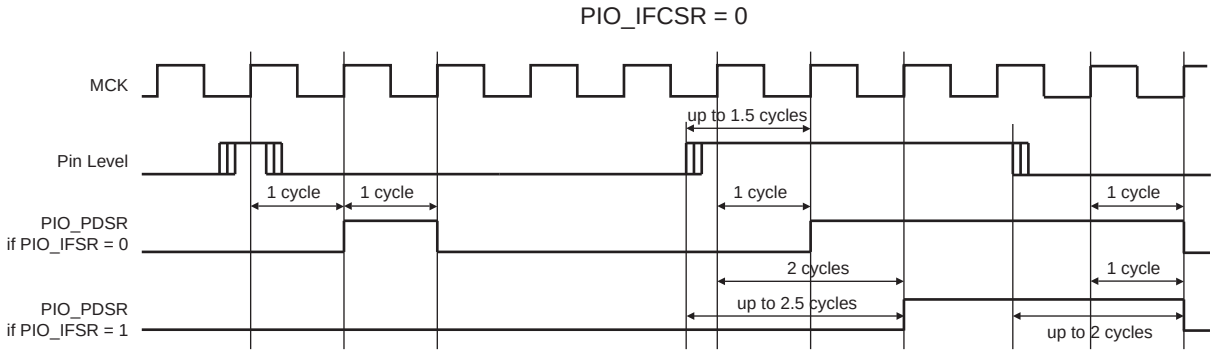
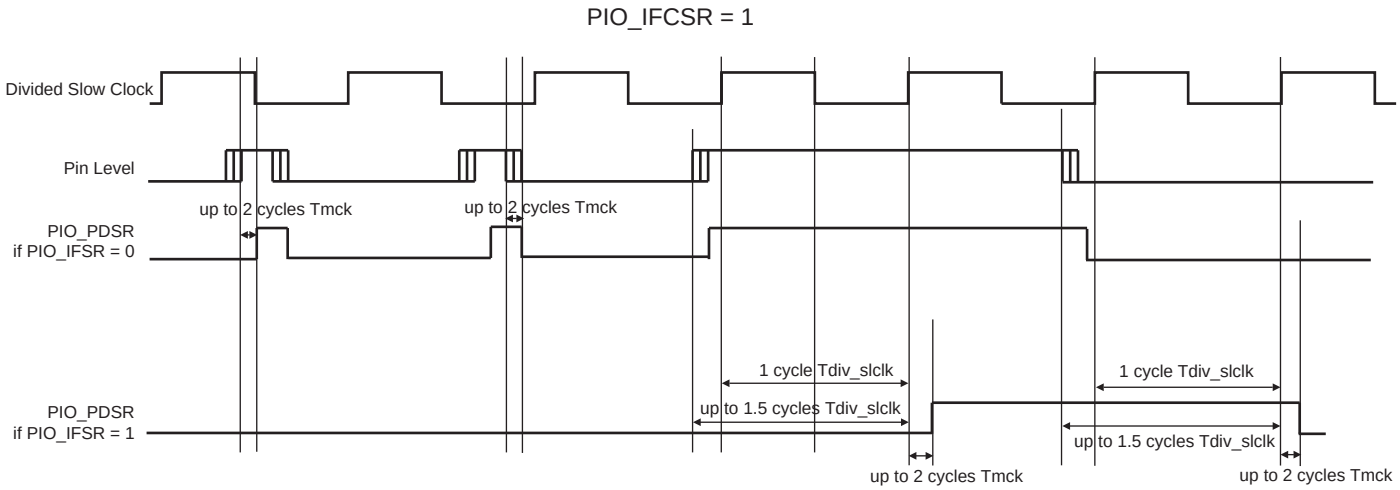



**Figure 27-6. Input Debouncing Filter Timing**




### 27.5.10 Input Edge/Level Interrupt

The PIO Controller can be programmed to generate an interrupt when it detects an edge or a level on an I/O line. The Input Edge/Level Interrupt is controlled by writing PIO_IER (Interrupt Enable Register) and PIO_IDR (Interrupt Disable Register), which respectively enable and disable the input change interrupt by setting and clearing the corresponding bit in PIO_IMR (Interrupt Mask Register). As Input change detection is possible only by comparing two successive samplings of the input of the I/O line, the PIO Controller clock must be enabled. The Input Change Interrupt is available, regardless of the configuration of the I/O line, i.e. configured as an input only, controlled by the PIO Controller or assigned to a peripheral function.

By default, the interrupt can be generated at any time an edge is detected on the input.

Some additional Interrupt modes can be enabled/disabled by writing in the PIO_AIMER (Additional Interrupt Modes Enable Register) and PIO_AIMDR (Additional Interrupt Modes Disable Register). The current state of this selection can be read through the PIO_AIMMR (Additional Interrupt Modes Mask Register)

These Additional Modes are:

- Rising Edge Detection
- Falling Edge Detection
- Low Level Detection
- High Level Detection

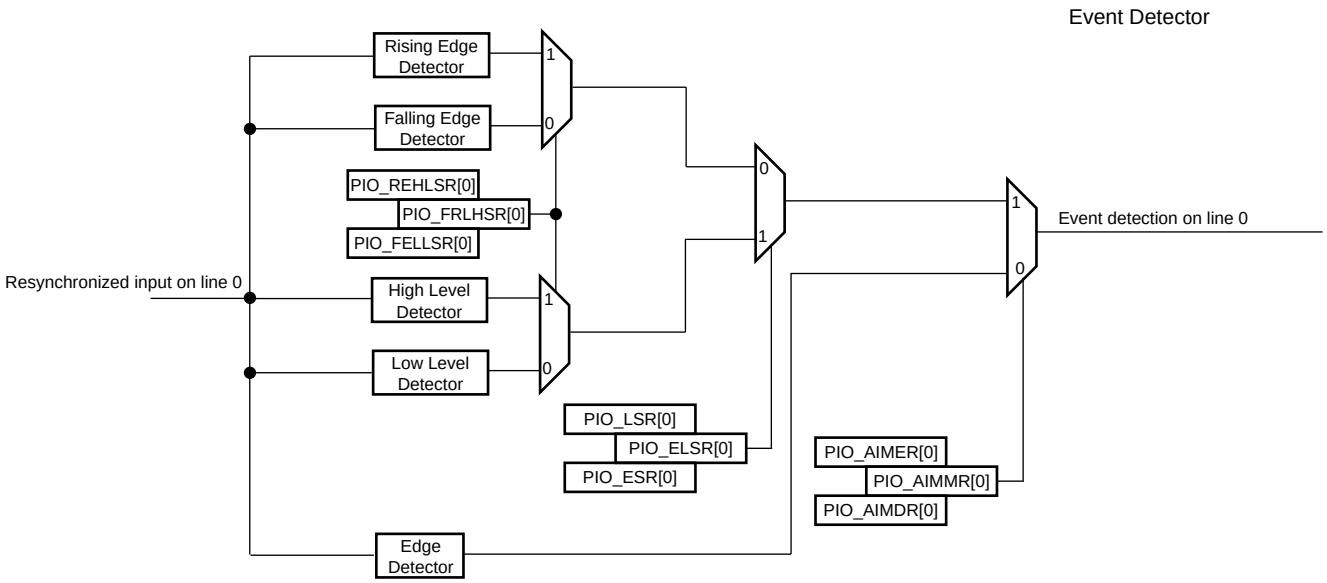
In order to select an Additional Interrupt Mode:

- The type of event detection (Edge or Level) must be selected by writing in the set of registers; PIO_ESR (Edge Select Register) and PIO_LSR (Level Select Register) which enable respectively, the Edge and Level Detection. The current status of this selection is accessible through the PIO_ELSR (Edge/Level Status Register).
- The Polarity of the event detection (Rising/Falling Edge or High/Low Level) must be selected by writing in the set of registers; PIO_FELLSR (Falling Edge /Low Level Select Register) and PIO_REHLSR (Rising Edge/High Level Select Register) which allow to select Falling or Rising Edge (if Edge is selected in the PIO_ELSR) Edge or High or Low Level Detection (if Level is selected in the PIO_ELSR). The current status of this selection is accessible through the PIO_FRLHSR (Fall/Rise - Low/High Status Register).

When an input Edge or Level is detected on an I/O line, the corresponding bit in PIO_ISR (Interrupt Status Register) is set. If the corresponding bit in PIO_IMR is set, the PIO Controller interrupt line is asserted. The interrupt signals of the thirty-two channels are ORed-wired together to generate a single interrupt signal to the . Nested Vector Interrupt Controller (NVIC).

When the software reads PIO_ISR, all the interrupts are automatically cleared. This signifies that all the interrupts that are pending when PIO_ISR is read must be handled. When an Interrupt is enabled on a “Level”, the interrupt is generated as long as the interrupt source is not cleared, even if some read accesses in PIO_ISR are performed.

**Figure 27-7. Event Detector on Input Lines (Figure represents line 0)**



#### 27.5.10.1 Example

If generating an interrupt is required on the following:

- Rising edge on PIO line 0
- Falling edge on PIO line 1
- Rising edge on PIO line 2
- Low Level on PIO line 3
- High Level on PIO line 4
- High Level on PIO line 5
- Falling edge on PIO line 6
- Rising edge on PIO line 7
- Any edge on the other lines

The configuration required is described below.

#### 27.5.10.2 Interrupt Mode Configuration

All the interrupt sources are enabled by writing 32'hFFFF_FFFF in PIO_IER.

Then the Additional Interrupt Mode is enabled for line 0 to 7 by writing 32'h0000_00FF in PIO_AIMER.

#### 27.5.10.3 Edge or Level Detection Configuration

Lines 3, 4 and 5 are configured in Level detection by writing 32'h0000_0038 in PIO_LSR.

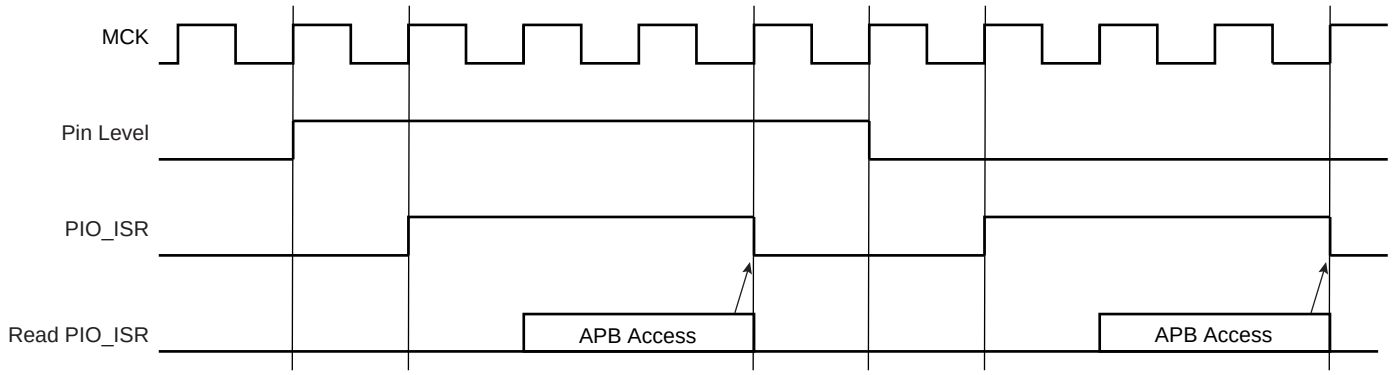
The other lines are configured in Edge detection by default, if they have not been previously configured. Otherwise, lines 0, 1, 2, 6 and 7 must be configured in Edge detection by writing 32'h0000_00C7 in PIO_ESR.

#### 27.5.10.4 Falling/Rising Edge or Low/High Level Detection Configuration.

Lines 0, 2, 4, 5 and 7 are configured in Rising Edge or High Level detection by writing 32'h0000_00B5 in PIO_REHLSR.

The other lines are configured in Falling Edge or Low Level detection by default, if they have not been previously configured. Otherwise, lines 1, 3 and 6 must be configured in Falling Edge/Low Level detection by writing 32'h0000_004A in PIO_FELLSR.

**Figure 27-8. Input Change Interrupt Timings if there are no Additional Interrupt Modes**




### 27.5.11 I/O Lines Lock

When an I/O line is controlled by a peripheral (particularly the Pulse Width Modulation Controller PWM), it can become locked by the action of this peripheral via an input of the PIO controller. When an I/O line is locked, the write of the corresponding bit in the registers PIO_PER, PIO_PDR, PIO_MDER, PIO_MDDR, PIO_PUDR, PIO_PUER, PIO_ABCDSR1 and PIO_ABCDSR2 is discarded in order to lock its configuration. The user can know

at anytime which I/O line is locked by reading the PIO Lock Status register PIO_LOCKSR. Once an I/O line is locked, the only way to unlock it is to apply a hardware reset to the PIO Controller.

### 27.5.12 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.

### 27.5.13 Write Protection Registers

To prevent any single software error that may corrupt PIO behavior, certain address spaces can be write-protected by setting the WPEN bit in the "PIO Write Protect Mode Register" (PIO_WPMR).

If a write access to the protected registers is detected, then the WPVS flag in the PIO Write Protect Status Register (PIO_WPSR) is set and the field WPVSRC indicates in which register the write access has been attempted.

The WPVS flag is reset by writing the PIO Write Protect Mode Register (PIO_WPMR) with the appropriate access key, WPKEY.

The protected registers are:

- "PIO Enable Register" on page 390
- "PIO Disable Register" on page 391
- "PIO Output Enable Register" on page 393
- "PIO Output Disable Register" on page 394
- "PIO Input Filter Enable Register" on page 396
- "PIO Input Filter Disable Register" on page 397
- "PIO Multi-driver Enable Register" on page 407
- "PIO Multi-driver Disable Register" on page 408
- "PIO Pull Up Disable Register" on page 410
- "PIO Pull Up Enable Register" on page 411
- "PIO Peripheral ABCD Select Register 1" on page 413
- "PIO Peripheral ABCD Select Register 2" on page 414
- "PIO Output Write Enable Register" on page 422
- "PIO Output Write Disable Register" on page 423
- "PIO Pad Pull Down Disable Register" on page 419
- "PIO Pad Pull Down Status Register" on page 421

## 27.6 I/O Lines Programming Example

The programing example as shown in Table 27-1 below is used to obtain the following configuration.

- 4-bit output port on I/O lines 0 to 3, (should be written in a single write operation), open-drain, with pull-up resistor
- Four output signals on I/O lines 4 to 7 (to drive LEDs for example), driven high and low, no pull-up resistor, no pull-down resistor
- Four input signals on I/O lines 8 to 11 (to read push-button states for example), with pull-up resistors, glitch filters and input change interrupts
- Four input signals on I/O line 12 to 15 to read an external device status (polled, thus no input change interrupt), no pull-up resistor, no glitch filter
- I/O lines 16 to 19 assigned to peripheral A functions with pull-up resistor
- I/O lines 20 to 23 assigned to peripheral B functions with pull-down resistor
- I/O line 24 to 27 assigned to peripheral C with Input Change Interrupt, no pull-up resistor and no pull-down resistor
- I/O line 28 to 31 assigned to peripheral D, no pull-up resistor and no pull-down resistor

**Table 27-1. Programming Example**

| Register | Value to be Written |
|---|---|
| PIO_PER | 0x0000_FFFF |
| PIO_PDR | 0xFFFF_0000 |
| PIO_OER | 0x0000_00FF |
| PIO_ODR | 0xFFFF_FF00 |
| PIO_IFER | 0x0000_0F00 |
| PIO_IFDR | 0xFFFF_F0FF |
| PIO_SODR | 0x0000_0000 |
| PIO_CODR | 0x0FFF_FFFF |
| PIO_IER | 0x0F00_0F00 |
| PIO_IDR | 0xF0FF_F0FF |
| PIO_MDER | 0x0000_000F |
| PIO_MDDR | 0xFFFF_FFF0 |
| PIO_PUDR | 0xFFF0_00F0 |
| PIO_PUER | 0x000F_FF0F |
| PIO_PPDDR | 0xFF0F_FFFF |
| PIO_PPDER | 0x00F0_0000 |
| PIO_ABCDSR1 | 0xF0F0_0000 |
| PIO_ABCDSR2 | 0xFF00_0000 |
| PIO_OWER | 0x0000_000F |
| PIO_OWDR | 0x0FFF_ FFF0 |

## 27.7 Parallel Input/Output Controller (PIO) User Interface

Each I/O line controlled by the PIO Controller is associated with a bit in each of the PIO Controller User Interface registers. Each register is 32 bits wide. If a parallel I/O line is not defined, writing to the corresponding bits has no effect. Undefined bits read zero. If the I/O line is not multiplexed with any peripheral, the I/O line is controlled by the PIO Controller and PIO_PSR returns 1 systematically.

**Table 27-2. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x0000 | PIO Enable Register | PIO_PER | Write-only | – |
| 0x0004 | PIO Disable Register | PIO_PDR | Write-only | – |
| 0x0008 | PIO Status Register | PIO_PSR | Read-only | (1) |
| 0x000C | Reserved | | | |
| 0x0010 | Output Enable Register | PIO_OER | Write-only | – |
| 0x0014 | Output Disable Register | PIO_ODR | Write-only | – |
| 0x0018 | Output Status Register | PIO_OSR | Read-only | 0x0000 0000 |
| 0x001C | Reserved | | | |
| 0x0020 | Glitch Input Filter Enable Register | PIO_IFER | Write-only | – |
| 0x0024 | Glitch Input Filter Disable Register | PIO_IFDR | Write-only | – |
| 0x0028 | Glitch Input Filter Status Register | PIO_IFSR | Read-only | 0x0000 0000 |
| 0x002C | Reserved | | | |
| 0x0030 | Set Output Data Register | PIO_SODR | Write-only | – |
| 0x0034 | Clear Output Data Register | PIO_CODR | Write-only | |
| 0x0038 | Output Data Status Register | PIO_ODSR | Read-only or(2) Read-write | – |
| 0x003C | Pin Data Status Register | PIO_PDSR | Read-only | (3) |
| 0x0040 | Interrupt Enable Register | PIO_IER | Write-only | – |
| 0x0044 | Interrupt Disable Register | PIO_IDR | Write-only | – |
| 0x0048 | Interrupt Mask Register | PIO_IMR | Read-only | 0x00000000 |
| 0x004C | Interrupt Status Register(4) | PIO_ISR | Read-only | 0x00000000 |
| 0x0050 | Multi-driver Enable Register | PIO_MDER | Write-only | – |
| 0x0054 | Multi-driver Disable Register | PIO_MDDR | Write-only | – |
| 0x0058 | Multi-driver Status Register | PIO_MDSR | Read-only | 0x00000000 |
| 0x005C | Reserved | – | – | – |
| 0x0060 | Pull-up Disable Register | PIO_PUDR | Write-only | – |
| 0x0064 | Pull-up Enable Register | PIO_PUER | Write-only | – |
| 0x0068 | Pad Pull-up Status Register | PIO_PUSR | Read-only | 0x00000000 |
| 0x006C | Reserved | – | – | – |
| 0x0070 | Peripheral Select Register 1 | PIO_ABCDSR1 | Read-write | 0x00000000 |
| 0x0074 | Peripheral Select Register 2 | PIO_ABCDSR2 | Read-write | 0x00000000 |
| 0x0078 to 0x007C | Reserved | – | – | – |

**Table 27-2. 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 | 0xFFFFFFFF |
| 0x009C | Reserved | – | – | |
| 0x00A0 | Output Write Enable | PIO_OWER | Write-only | – |
| 0x00A4 | Output Write Disable | PIO_OWDR | Write-only | – |
| 0x00A8 | Output Write Status Register | PIO_OWSR | Read-only | 0x00000000 |
| 0x00AC | Reserved | – | – | |
| 0x00B0 | Additional Interrupt Modes Enable Register | PIO_AIMER | Write-only | – |
| 0x00B4 | Additional Interrupt Modes Disables Register | PIO_AIMDR | Write-only | – |
| 0x00B8 | Additional Interrupt Modes Mask Register | PIO_AIMMR | Read-only | 0x00000000 |
| 0x00BC | Reserved | – | – | – |
| 0x00C0 | Edge Select Register | PIO_ESR | Write-only | – |
| 0x00C4 | Level Select Register | PIO_LSR | Write-only | – |
| 0x00C8 | Edge/Level Status Register | PIO_ELSR | Read-only | 0x00000000 |
| 0x00CC | Reserved | – | – | – |
| 0x00D0 | Falling Edge/Low Level Select Register | PIO_FELLSR | Write-only | – |
| 0x00D4 | Rising Edge/ High Level Select Register | PIO_REHLSR | Write-only | – |
| 0x00D8 | Fall/Rise - Low/High Status Register | PIO_FRLHSR | Read-only | 0x00000000 |
| 0x00DC | Reserved | – | – | – |
| 0x00E0 | Lock Status | PIO_LOCKSR | Read-only | 0x00000000 |
| 0x00E4 | Write Protect Mode Register | PIO_WPMR | Read-write | 0x0 |
| 0x00E8 | Write Protect Status Register | PIO_WPSR | Read-only | 0x0 |
| 0x00EC to 0x00F8 | Reserved | – | – | – |
| 0x0100 | Schmitt Trigger Register | PIO_SCHMITT | Read-write | 0x00000000 |
| 0x0104-0x010C | Reserved | – | – | – |
| 0x0110 | Reserved | – | – | – |
| 0x0114-0x011C | Reserved | – | – | – |

Notes: 1. Reset value of PIO_PSR depends on the product implementation.
2. PIO_ODSR is Read-only or Read/Write depending on PIO_OWSR I/O lines.
3. Reset value of PIO_PDSR depends on the level of the I/O lines. Reading the I/O line levels requires the clock of the PIO Controller to be enabled, otherwise PIO_PDSR reads the levels present on the I/O line at the time the clock was disabled.
4. PIO_ISR is reset at 0x0. However, the first read of the register may read a different value as input changes may have occurred.

Note: If an offset is not listed in the table it must be considered as reserved.

### 27.7.1 PIO Enable Register

**Name:** PIO_PER

**Addresses:** 0x400E0E00 (PIOA), 0x400E1000 (PIOB), 0x400E1200 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

This register can only be written if the WPEN bit is cleared in “PIO Write Protect Mode Register” .

- **P0-P31: PIO Enable**

0 = No effect.

1 = Enables the PIO to control the corresponding pin (disables peripheral control of the pin).

### 27.7.2 PIO Disable Register

**Name:** PIO_PDR

**Addresses:** 0x400E0E04 (PIOA), 0x400E1004 (PIOB), 0x400E1204 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

This register can only be written if the WPEN bit is cleared in “PIO Write Protect Mode Register” .

- **P0-P31: PIO Disable**

0 = No effect.

1 = Disables the PIO from controlling the corresponding pin (enables peripheral control of the pin).

### 27.7.3 PIO Status Register

**Name:** PIO_PSR

**Addresses:** 0x400E0E08 (PIOA), 0x400E1008 (PIOB), 0x400E1208 (PIOC)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: PIO Status**

0 = PIO is inactive on the corresponding I/O line (peripheral is active).

1 = PIO is active on the corresponding I/O line (peripheral is inactive).

### 27.7.4 PIO Output Enable Register

**Name:** PIO_OER

**Addresses:** 0x400E0E10 (PIOA), 0x400E1010 (PIOB), 0x400E1210 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

This register can only be written if the WPEN bit is cleared in “PIO Write Protect Mode Register” .

- **P0-P31: Output Enable**

0 = No effect.

1 = Enables the output on the I/O line.

### 27.7.5 PIO Output Disable Register

**Name:** PIO_ODR

**Addresses:** 0x400E0E14 (PIOA), 0x400E1014 (PIOB), 0x400E1214 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

This register can only be written if the WPEN bit is cleared in “PIO Write Protect Mode Register” .

- **P0-P31: Output Disable**

0 = No effect.

1 = Disables the output on the I/O line.

### 27.7.6 PIO Output Status Register

**Name:** PIO_OSR

**Addresses:** 0x400E0E18 (PIOA), 0x400E1018 (PIOB), 0x400E1218 (PIOC)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Output Status**

0 = The I/O line is a pure input.

1 = The I/O line is enabled in output.

### 27.7.7 PIO Input Filter Enable Register

**Name:** PIO_IFER

**Addresses:** 0x400E0E20 (PIOA), 0x400E1020 (PIOB), 0x400E1220 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

This register can only be written if the WPEN bit is cleared in “PIO Write Protect Mode Register” .

- **P0-P31: Input Filter Enable**

0 = No effect.

1 = Enables the input glitch filter on the I/O line.

### 27.7.8 PIO Input Filter Disable Register

**Name:** PIO_IFDR

**Addresses:** 0x400E0E24 (PIOA), 0x400E1024 (PIOB), 0x400E1224 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

This register can only be written if the WPEN bit is cleared in “PIO Write Protect Mode Register” .

- **P0-P31: Input Filter Disable**

0 = No effect.

1 = Disables the input glitch filter on the I/O line.

### 27.7.9 PIO Input Filter Status Register

**Name:** PIO_IFSR

**Addresses:** 0x400E0E28 (PIOA), 0x400E1028 (PIOB), 0x400E1228 (PIOC)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Input Filer Status**

0 = The input glitch filter is disabled on the I/O line.

1 = The input glitch filter is enabled on the I/O line.

### 27.7.10 PIO Set Output Data Register

**Name:** PIO_SODR

**Addresses:** 0x400E0E30 (PIOA), 0x400E1030 (PIOB), 0x400E1230 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Set Output Data**

0 = No effect.

1 = Sets the data to be driven on the I/O line.

### 27.7.11 PIO Clear Output Data Register

**Name:** PIO_CODR

**Addresses:** 0x400E0E34 (PIOA), 0x400E1034 (PIOB), 0x400E1234 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Clear Output Data**

0 = No effect.

1 = Clears the data to be driven on the I/O line.

### 27.7.12 PIO Output Data Status Register

**Name:** PIO_ODSR

**Addresses:** 0x400E0E38 (PIOA), 0x400E1038 (PIOB), 0x400E1238 (PIOC)

**Access:** Read-only or Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Output Data Status**

0 = The data to be driven on the I/O line is 0.

1 = The data to be driven on the I/O line is 1.

### 27.7.13 PIO Pin Data Status Register

**Name:** PIO_PDSR

**Addresses:** 0x400E0E3C (PIOA), 0x400E103C (PIOB), 0x400E123C (PIOC)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Output Data Status**

0 = The I/O line is at level 0.

1 = The I/O line is at level 1.

### 27.7.14 PIO Interrupt Enable Register

**Name:** PIO_IER

**Addresses:** 0x400E0E40 (PIOA), 0x400E1040 (PIOB), 0x400E1240 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Input Change Interrupt Enable**

0 = No effect.

1 = Enables the Input Change Interrupt on the I/O line.

### 27.7.15 PIO Interrupt Disable Register

**Name:** PIO_IDR

**Addresses:** 0x400E0E44 (PIOA), 0x400E1044 (PIOB), 0x400E1244 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Input Change Interrupt Disable**

0 = No effect.

1 = Disables the Input Change Interrupt on the I/O line.

### 27.7.16 PIO Interrupt Mask Register

**Name:** PIO_IMR

**Addresses:** 0x400E0E48 (PIOA), 0x400E1048 (PIOB), 0x400E1248 (PIOC)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Input Change Interrupt Mask**

0 = Input Change Interrupt is disabled on the I/O line.

1 = Input Change Interrupt is enabled on the I/O line.

### 27.7.17 PIO Interrupt Status Register

**Name:** PIO_ISR

**Addresses:** 0x400E0E4C (PIOA), 0x400E104C (PIOB), 0x400E124C (PIOC)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Input Change Interrupt Status**

0 = No Input Change has been detected on the I/O line since PIO_ISR was last read or since reset.

1 = At least one Input Change has been detected on the I/O line since PIO_ISR was last read or since reset.

### 27.7.18 PIO Multi-driver Enable Register

**Name:** PIO_MDER

**Addresses:** 0x400E0E50 (PIOA), 0x400E1050 (PIOB), 0x400E1250 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

This register can only be written if the WPEN bit is cleared in “PIO Write Protect Mode Register” .

- **P0-P31: Multi Drive Enable.**

0 = No effect.

1 = Enables Multi Drive on the I/O line.

### 27.7.19 PIO Multi-driver Disable Register

**Name:** PIO_MDDR

**Addresses:** 0x400E0E54 (PIOA), 0x400E1054 (PIOB), 0x400E1254 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

This register can only be written if the WPEN bit is cleared in “PIO Write Protect Mode Register” .

- **P0-P31: Multi Drive Disable.**

0 = No effect.

1 = Disables Multi Drive on the I/O line.

### 27.7.20 PIO Multi-driver Status Register

**Name:** PIO_MDSR

**Addresses:** 0x400E0E58 (PIOA), 0x400E1058 (PIOB), 0x400E1258 (PIOC)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Multi Drive Status.**

0 = The Multi Drive is disabled on the I/O line. The pin is driven at high and low level.

1 = The Multi Drive is enabled on the I/O line. The pin is driven at low level only.

### 27.7.21 PIO Pull Up Disable Register

**Name:** PIO_PUDR

**Addresses:** 0x400E0E60 (PIOA), 0x400E1060 (PIOB), 0x400E1260 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

This register can only be written if the WPEN bit is cleared in “PIO Write Protect Mode Register” .

- **P0-P31: Pull Up Disable.**

0 = No effect.

1 = Disables the pull up resistor on the I/O line.

### 27.7.22 PIO Pull Up Enable Register

**Name:** PIO_PUER

**Addresses:** 0x400E0E64 (PIOA), 0x400E1064 (PIOB), 0x400E1264 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

This register can only be written if the WPEN bit is cleared in “PIO Write Protect Mode Register” .

- **P0-P31: Pull Up Enable.**

0 = No effect.

1 = Enables the pull up resistor on the I/O line.

### 27.7.23 PIO Pull Up Status Register

**Name:** PIO_PUSR

**Addresses:** 0x400E0E68 (PIOA), 0x400E1068 (PIOB), 0x400E1268 (PIOC)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Pull Up Status.**

0 = Pull Up resistor is enabled on the I/O line.

1 = Pull Up resistor is disabled on the I/O line.

### 27.7.24 PIO Peripheral ABCD Select Register 1

**Name:** PIO_ABCDSR1

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

This register can only be written if the WPEN bit is cleared in “PIO Write Protect 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.

### 27.7.25 PIO Peripheral ABCD Select Register 2

**Name:** PIO_ABCDSR2

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

This register can only be written if the WPEN bit is cleared in “PIO Write Protect 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.

### 27.7.26 PIO Input Filter Slow Clock Disable Register

**Name:** PIO_IFSCDR

**Addresses:** 0x400E0E80 (PIOA), 0x400E1080 (PIOB), 0x400E1280 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: PIO Clock Glitch Filtering Select.**

0 = No Effect.

1 = The Glitch Filter is able to filter glitches with a duration < Tmck/2.

### 27.7.27 PIO Input Filter Slow Clock Enable Register

**Name:** PIO_IFSCER

**Addresses:** 0x400E0E84 (PIOA), 0x400E1084 (PIOB), 0x400E1284 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Debouncing Filtering Select.**

0 = No Effect.

1 = The Debouncing Filter is able to filter pulses with a duration < Tdiv_slclk/2.

### 27.7.28 PIO Input Filter Slow Clock Status Register

**Name:** PIO_IFSCSR

**Addresses:** 0x400E0E88 (PIOA), 0x400E1088 (PIOB), 0x400E1288 (PIOC)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Glitch or Debouncing Filter Selection Status**

0 = The Glitch Filter is able to filter glitches with a duration < Tmck2.

1 = The Debouncing Filter is able to filter pulses with a duration < Tdiv_slclk/2.

### 27.7.29 PIO Slow Clock Divider Debouncing Register

**Name:** PIO_SCDR

**Addresses:** 0x400E0E8C (PIOA), 0x400E108C (PIOB), 0x400E128C (PIOC)

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | - | - |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| - | - | - | - | - | - | - | - |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| - | - | DIV13 | DIV12 | DIV11 | DIV10 | DIV9 | DIV8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| DIV7 | DIV6 | DIV5 | DIV4 | DIV3 | DIV2 | DIV1 | DIV0 |

- **DIVx: Slow Clock Divider Selection for Debouncing**

Tdiv_slclk = 2*(DIV+1)*Tslow_clock.

### 27.7.30 PIO Pad Pull Down Disable Register

**Name:** PIO_PPDDR

**Addresses:** 0x400E0E90 (PIOA), 0x400E1090 (PIOB), 0x400E1290 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

This register can only be written if the WPEN bit is cleared in “PIO Write Protect Mode Register” .

- **P0-P31: Pull Down Disable.**

0 = No effect.

1 = Disables the pull down resistor on the I/O line.

### 27.7.31 PIO Pad Pull Down Enable Register

**Name:** PIO_PPDER

**Addresses:** 0x400E0E94 (PIOA), 0x400E1094 (PIOB), 0x400E1294 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

This register can only be written if the WPEN bit is cleared in “PIO Write Protect Mode Register” .

- **P0-P31: Pull Down Enable.**

0 = No effect.

1 = Enables the pull down resistor on the I/O line.

### 27.7.32 PIO Pad Pull Down Status Register

**Name:** PIO_PPDSR

**Addresses:** 0x400E0E98 (PIOA), 0x400E1098 (PIOB), 0x400E1298 (PIOC)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

This register can only be written if the WPEN bit is cleared in “PIO Write Protect Mode Register” .

- **P0-P31: Pull Down Status.**

0 = Pull Down resistor is enabled on the I/O line.

1 = Pull Down resistor is disabled on the I/O line.

### 27.7.33 PIO Output Write Enable Register

**Name:** PIO_OWER

**Addresses:** 0x400E0EA0 (PIOA), 0x400E10A0 (PIOB), 0x400E12A0 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

This register can only be written if the WPEN bit is cleared in “PIO Write Protect Mode Register” .

- **P0-P31: Output Write Enable.**

0 = No effect.

1 = Enables writing PIO_ODSR for the I/O line.

### 27.7.34 PIO Output Write Disable Register

**Name:** PIO_OWDR

**Addresses:** 0x400E0EA4 (PIOA), 0x400E10A4 (PIOB), 0x400E12A4 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

This register can only be written if the WPEN bit is cleared in “PIO Write Protect Mode Register” .

- **P0-P31: Output Write Disable.**

0 = No effect.

1 = Disables writing PIO_ODSR for the I/O line.

### 27.7.35 PIO Output Write Status Register

**Name:** PIO_OWSR

**Addresses:** 0x400E0EA8 (PIOA), 0x400E10A8 (PIOB), 0x400E12A8 (PIOC)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Output Write Status.**

0 = Writing PIO_ODSR does not affect the I/O line.

1 = Writing PIO_ODSR affects the I/O line.

### 27.7.36 PIO Additional Interrupt Modes Enable Register

**Name:** PIO_AIMER

**Addresses:** 0x400E0EB0 (PIOA), 0x400E10B0 (PIOB), 0x400E12B0 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Additional Interrupt Modes Enable.**

0 = No effect.

1 = The interrupt source is the event described in PIO_ELSR and PIO_FRLHSR.

### 27.7.37 PIO Additional Interrupt Modes Disable Register

**Name:** PIO_AIMDR

**Addresses:** 0x400E0EB4 (PIOA), 0x400E10B4 (PIOB), 0x400E12B4 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Additional Interrupt Modes Disable.**

0 = No effect.

1 = The interrupt mode is set to the default interrupt mode (Both Edge detection).

### 27.7.38 PIO Additional Interrupt Modes Mask Register

**Name:** PIO_AIMMR

**Addresses:** 0x400E0EB8 (PIOA), 0x400E10B8 (PIOB), 0x400E12B8 (PIOC)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Peripheral CD Status.**

0 = The interrupt source is a Both Edge detection event

1 = The interrupt source is described by the registers PIO_ELSR and PIO_FRLHSR

### 27.7.39 PIO Edge Select Register

**Name:** PIO_ESR

**Addresses:** 0x400E0EC0 (PIOA), 0x400E10C0 (PIOB), 0x400E12C0 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Edge Interrupt Selection.**

0 = No effect.

1 = The interrupt source is an Edge detection event.

### 27.7.40 PIO Level Select Register

**Name:** PIO_LSR

**Addresses:** 0x400E0EC4 (PIOA), 0x400E10C4 (PIOB), 0x400E12C4 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Level Interrupt Selection.**

0 = No effect.

1 = The interrupt source is a Level detection event.

### 27.7.41 PIO Edge/Level Status Register

**Name:** PIO_ELSR

**Addresses:** 0x400E0EC8 (PIOA), 0x400E10C8 (PIOB), 0x400E12C8 (PIOC)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Edge/Level Interrupt source selection.**

0 = The interrupt source is an Edge detection event.

1 = The interrupt source is a Level detection event.

### 27.7.42 PIO Falling Edge/Low Level Select Register

**Name:** PIO_FELLSR

**Addresses:** 0x400E0ED0 (PIOA), 0x400E10D0 (PIOB), 0x400E12D0 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Falling Edge/Low Level Interrupt Selection.**

0 = No effect.

1 = The interrupt source is set to a Falling Edge detection or Low Level detection event, depending on PIO_ELSR.

### 27.7.43 PIO Rising Edge/High Level Select Register

**Name:** PIO_REHLSR

**Addresses:** 0x400E0ED4 (PIOA), 0x400E10D4 (PIOB), 0x400E12D4 (PIOC)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Rising Edge /High Level Interrupt Selection.**

0 = No effect.

1 = The interrupt source is set to a Rising Edge detection or High Level detection event, depending on PIO_ELSR.

### 27.7.44 PIO Fall/Rise - Low/High Status Register

**Name:** PIO_FRLHSR

**Addresses:** 0x400E0ED8 (PIOA), 0x400E10D8 (PIOB), 0x400E12D8 (PIOC)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Edge /Level Interrupt Source Selection.**

0 = The interrupt source is a Falling Edge detection (if PIO_ELSR = 0) or Low Level detection event (if PIO_ELSR = 1).

1 = The interrupt source is a Rising Edge detection (if PIO_ELSR = 0) or High Level detection event (if PIO_ELSR = 1).

### 27.7.45 PIO Lock Status Register

**Name:** PIO_LOCKSR

**Addresses:** 0x400E0EE0 (PIOA), 0x400E10E0 (PIOB), 0x400E12E0 (PIOC)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

- **P0-P31: Lock Status.**

0 = The I/O line is not locked.

1 = The I/O line is locked.

### 27.7.46 PIO Write Protect Mode Register

**Name:** PIO_WPMR

**Addresses:** 0x400E0EE4 (PIOA), 0x400E10E4 (PIOB), 0x400E12E4 (PIOC)

**Access:** Read-write

**Reset:** See Table 27-2

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | WPEN |

For more information on Write Protection Registers, refer to Section 27.7 ”Parallel Input/Output Controller (PIO) User Interface”.

- **WPEN: Write Protect Enable**

0 = Disables the Write Protect if WPKEY corresponds to 0x50494F (“PIO” in ASCII).

1 = Enables the Write Protect if WPKEY corresponds to 0x50494F (“PIO” in ASCII).

Protects the registers:

“PIO Enable Register” on page 390

“PIO Disable Register” on page 391

“PIO Output Enable Register” on page 393

“PIO Output Disable Register” on page 394

“PIO Input Filter Enable Register” on page 396

“PIO Input Filter Disable Register” on page 397

“PIO Multi-driver Enable Register” on page 407

“PIO Multi-driver Disable Register” on page 408

“PIO Pull Up Disable Register” on page 410

“PIO Pull Up Enable Register” on page 411

“PIO Peripheral ABCD Select Register 1” on page 413

“PIO Peripheral ABCD Select Register 2” on page 414

“PIO Output Write Enable Register” on page 422

“PIO Output Write Disable Register” on page 423

“PIO Pad Pull Down Disable Register” on page 419

“PIO Pad Pull Down Status Register” on page 421

- **WPKEY: Write Protect KEY**

Should be written at value 0x50494F (“PIO” in ASCII). Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

### 27.7.47 PIO Write Protect Status Register

**Name:** PIO_WPSR

**Addresses:** 0x400E0EE8 (PIOA), 0x400E10E8 (PIOB), 0x400E12E8 (PIOC)

**Access:** Read-only

**Reset:** See Table 27-2

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| WPVSRC | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| WPVSRC | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| — | — | — | — | — | — | — | WPVS |

- **WPVS: Write Protect Violation Status**

0 = No Write Protect Violation has occurred since the last read of the PIO_WPSR register.

1 = A Write Protect Violation has occurred since the last read of the PIO_WPSR register. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protect Violation Source**

When WPVS is active, this field indicates the write-protected register (through address offset or code) in which a write access has been attempted.

Note: Reading PIO_WPSR automatically clears all fields.

### 27.7.48 PIO Schmitt Trigger Register

**Name:** PIO_SCHMITT

**Addresses:** 0x400E0F00 (PIOA), 0x400E1100 (PIOB), 0x400E1300 (PIOC)

**Access:** Read-write

**Reset:** See Figure 27-2

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| SCHMITT31 | SCHMITT30 | SCHMITT29 | SCHMITT28 | SCHMITT27 | SCHMITT26 | SCHMITT25 | SCHMITT24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| SCHMITT23 | SCHMITT22 | SCHMITT21 | SCHMITT20 | SCHMITT19 | SCHMITT18 | SCHMITT17 | SCHMITT16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| SCHMITT15 | SCHMITT14 | SCHMITT13 | SCHMITT12 | SCHMITT11 | SCHMITT10 | SCHMITT9 | SCHMITT8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SCHMITT7 | SCHMITT6 | SCHMITT5 | SCHMITT4 | SCHMITT3 | SCHMITT2 | SCHMITT1 | SCHMITT0 |

- **SCHMITTx [x=0..31]:**

0 = Schmitt Trigger is enabled.

1= Schmitt Trigger is disabled.

# 28. Serial Peripheral Interface (SPI)

## 28.1 Description

The Serial Peripheral Interface (SPI) circuit is a synchronous serial data link that provides communication with external devices in Master or Slave Mode. It also enables communication between processors if an external processor is connected to the system.

The Serial Peripheral Interface is essentially a shift register that serially transmits data bits to other SPIs. During a data transfer, one SPI system acts as the “master”' which controls the data flow, while the other devices act as “slaves'' which have data shifted into and out by the master. Different CPUs can take turn being masters (Multiple Master Protocol opposite to Single Master Protocol where one CPU is always the master while all of the others are always slaves) and one master may simultaneously shift data into multiple slaves. However, only one slave may drive its output to write data back to the master at any given time.

A slave device is selected when the master asserts its NSS signal. If multiple slave devices exist, the master generates a separate slave select signal for each slave (NPCS).

The SPI system consists of two data lines and two control lines:

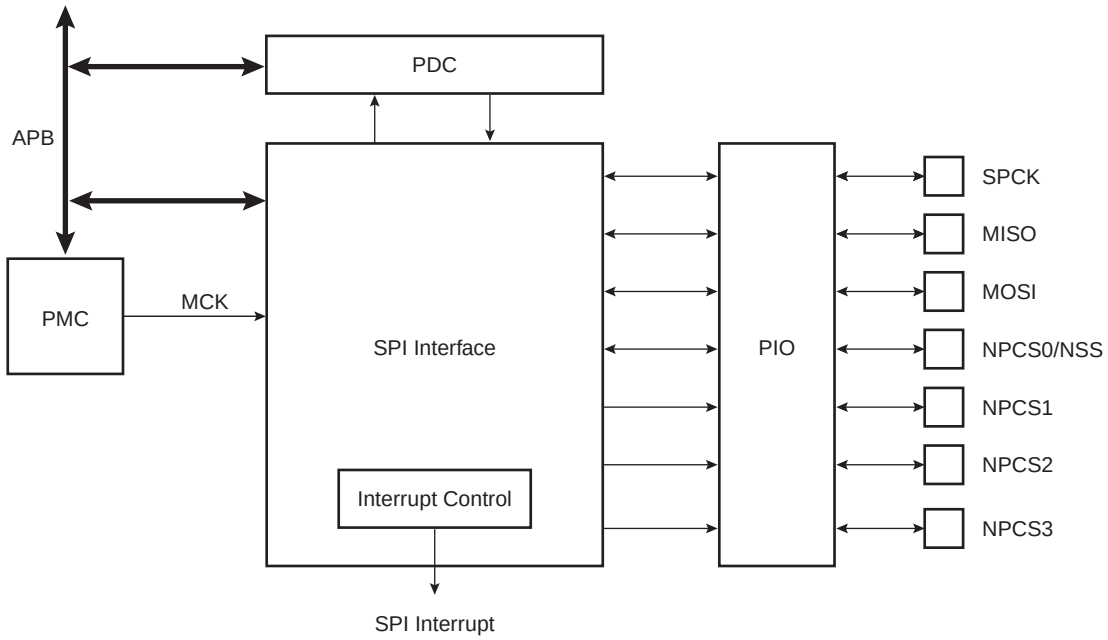
- Master Out Slave In (MOSI): This data line supplies the output data from the master shifted into the input(s) of the slave(s).
- Master In Slave Out (MISO): This data line supplies the output data from a slave to the input of the master. There may be no more than one slave transmitting data during any particular transfer.
- Serial Clock (SPCK): This control line is driven by the master and regulates the flow of the data bits. The master may transmit data at a variety of baud rates; the SPCK line cycles once for each bit that is transmitted.
- Slave Select (NSS): This control line allows slaves to be turned on and off by hardware.

## 28.2 Embedded Characteristics

- Compatible with an Embedded 32-bit Microcontroller
- Supports Communication with Serial External Devices
  - Four Chip Selects with External Decoder Support Allow Communication with Up to 15 Peripherals
  - Serial Memories, such as DataFlash and 3-wire EEPROMs
  - Serial Peripherals, such as ADCs, DACs, LCD Controllers, CAN Controllers and Sensors
  - External Co-processors
- Master or Slave Serial Peripheral Bus Interface
  - 8- to 16-bit Programmable Data Length Per Chip Select
  - Programmable Phase and Polarity Per Chip Select
  - Programmable Transfer Delays Between Consecutive Transfers and Between Clock and Data Per Chip Select
  - Programmable Delay Between Consecutive Transfers
  - Selectable Mode Fault Detection
- Connection to PDC Channel Capabilities Optimizes Data Transfers
  - One Channel for the Receiver, One Channel for the Transmitter
  - Next Buffer Support

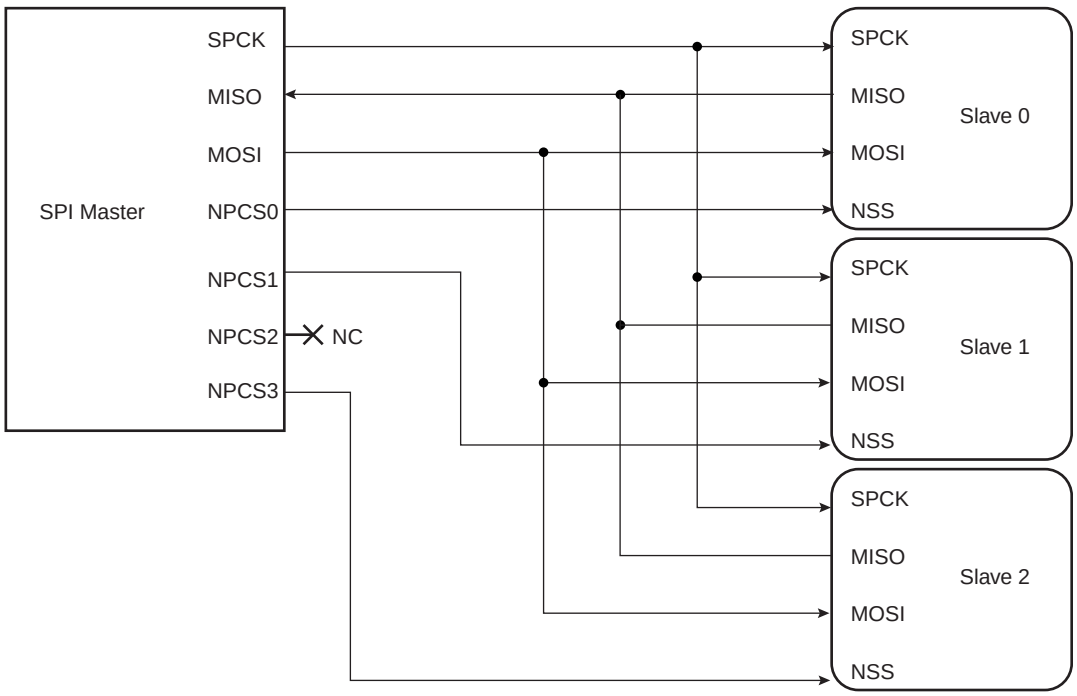
## 28.3 Block Diagram

**Figure 28-1. Block Diagram**




## 28.4 Application Block Diagram

**Figure 28-2. Application Block Diagram: Single Master/Multiple Slave Implementation**



## 28.5 Signal Description

**Table 28-1. Signal Description**

| Pin Name | Pin Description | Type | |
|---|---|---|---|
| | | Master | Slave |
| MISO | Master In Slave Out | Input | Output |
| MOSI | Master Out Slave In | Output | Input |
| SPCK | Serial Clock | Output | Input |
| NPCS1-NPCS3 | Peripheral Chip Selects | Output | Unused |
| NPCS0/NSS | Peripheral Chip Select/Slave Select | Output | Input |

## 28.6 Product Dependencies

### 28.6.1 I/O Lines

The pins used for interfacing the compliant external devices may be multiplexed with PIO lines. The programmer must first program the PIO controllers to assign the SPI pins to their peripheral functions.

**Table 28-2. I/O Lines**

| Instance | Signal | I/O Line | Peripheral |
|---|---|---|---|
| SPI | MISO | PA12 | A |
| SPI | MOSI | PA13 | A |
| SPI | NPCS0 | PA11 | A |
| SPI | NPCS1 | PA9 | B |
| SPI | NPCS1 | PA31 | A |
| SPI | NPCS1 | PB14 | A |
| SPI | NPCS1 | PC4 | B |
| SPI | NPCS2 | PA10 | B |
| SPI | NPCS2 | PA30 | B |
| SPI | NPCS2 | PB2 | B |
| SPI | NPCS2 | PC7 | B |
| SPI | NPCS3 | PA3 | B |
| SPI | NPCS3 | PA5 | B |
| SPI | NPCS3 | PA22 | B |
| SPI | SPCK | PA14 | A |

### 28.6.2 Power Management

The SPI may be clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the SPI clock.

### 28.6.3 Interrupt

The SPI interface has an interrupt line connected to the Nested Vector Interrupt Controller (NVIC).Handling the SPI interrupt requires programming the NVIC before configuring the SPI.

**Table 28-3. Peripheral IDs**

| Instance | ID |
|---|---|
| SPI | 21 |

## 28.7 Functional Description

### 28.7.1 Modes of Operation

The SPI operates in Master Mode or in Slave Mode.

Operation in Master Mode is programmed by writing at 1 the MSTR bit in the Mode Register. The pins NPCS0 to NPCS3 are all configured as outputs, the SPCK pin is driven, the MISO line is wired on the receiver input and the MOSI line driven as an output by the transmitter.

If the MSTR bit is written at 0, the SPI operates in Slave Mode. The MISO line is driven by the transmitter output, the MOSI line is wired on the receiver input, the SPCK pin is driven by the transmitter to synchronize the receiver. The NPCS0 pin becomes an input, and is used as a Slave Select signal (NSS). The pins NPCS1 to NPCS3 are not driven and can be used for other purposes.

The data transfers are identically programmable for both modes of operations. The baud rate generator is activated only in Master Mode.

### 28.7.2 Data Transfer

Four combinations of polarity and phase are available for data transfers. The clock polarity is programmed with the CPOL bit in the Chip Select Register. The clock phase is programmed with the NCPHA bit. These two parameters determine the edges of the clock signal on which data is driven and sampled. Each of the two parameters has two possible states, resulting in four possible combinations that are incompatible with one another. Thus, a master/slave pair must use the same parameter pair values to communicate. If multiple slaves are used and fixed in different configurations, the master must reconfigure itself each time it needs to communicate with a different slave.

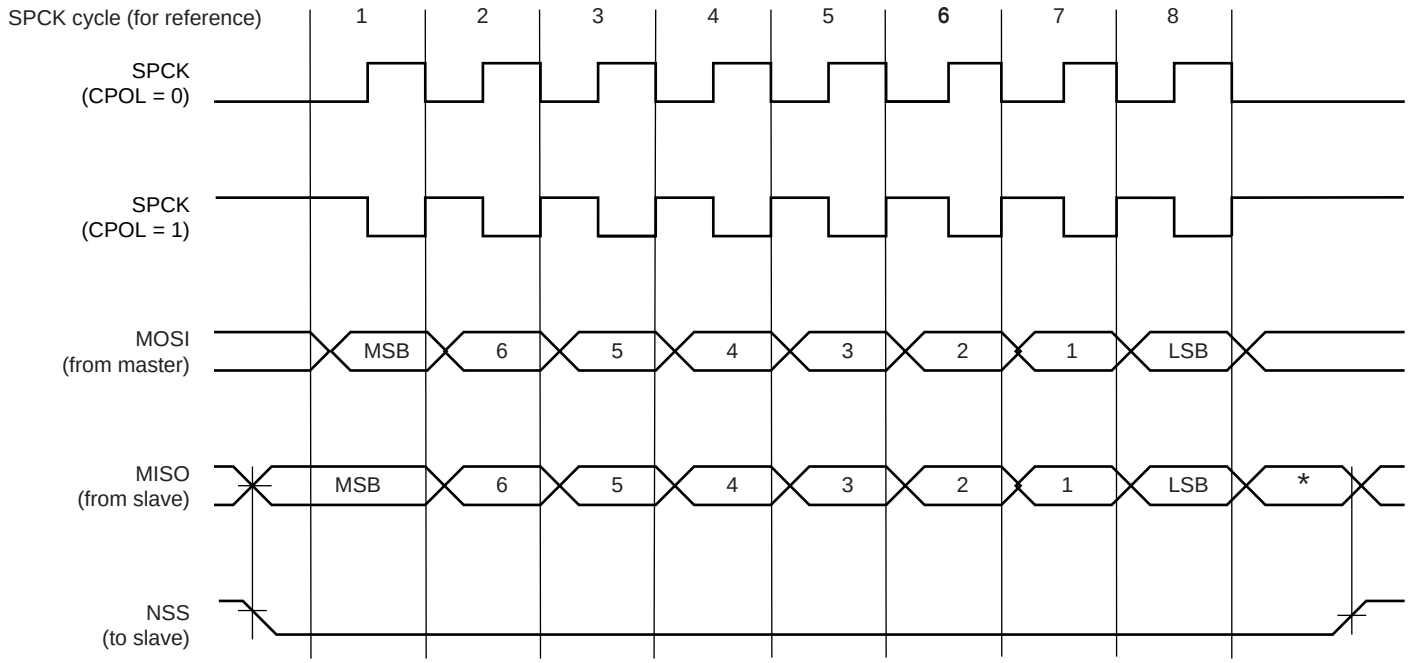
Table 28-4 shows the four modes and corresponding parameter settings.

**Table 28-4. 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 |

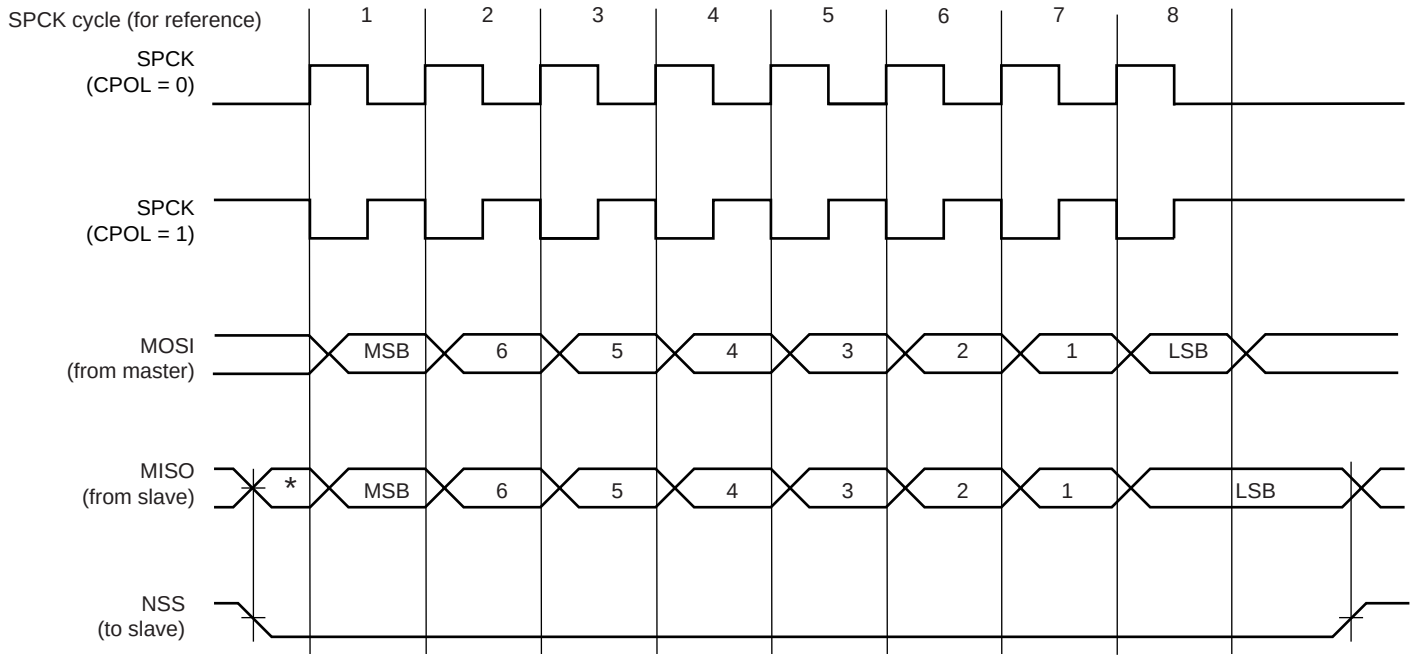
Figure 28-3 and Figure 28-4 show examples of data transfers.

**Figure 28-3. SPI Transfer Format (NCPHA = 1, 8 bits per transfer)**




* Not defined, but normally MSB of previous character received.

**Figure 28-4. SPI Transfer Format (NCPHA = 0, 8 bits per transfer)**




* Not defined but normally LSB of previous character transmitted.

### 28.7.3 Master Mode Operations

When configured in Master Mode, the SPI operates on the clock generated by the internal programmable baud rate generator. It fully controls the data transfers to and from the slave(s) connected to the SPI bus. The SPI drives the chip select line to the slave and the serial clock signal (SPCK).

The SPI features two holding registers, the Transmit Data Register and the Receive Data Register, and a single Shift Register. The holding registers maintain the data flow at a constant rate.

After enabling the SPI, a data transfer begins when the processor writes to the SPI_TDR (Transmit Data Register). The written data is immediately transferred in the Shift Register and transfer on the SPI bus starts. While the data in the Shift Register is shifted on the MOSI line, the MISO line is sampled and shifted in the Shift Register. Receiving data cannot occur without transmitting data. If receiving mode is not needed, for example when communicating with a slave receiver only (such as an LCD), the receive status flags in the status register can be discarded.

Before writing the TDR, the PCS field in the SPI_MR register must be set in order to select a slave.

After enabling the SPI, a data transfer begins when the processor writes to the SPI_TDR (Transmit Data Register). The written data is immediately transferred in the Shift Register and transfer on the SPI bus starts. While the data in the Shift Register is shifted on the MOSI line, the MISO line is sampled and shifted in the Shift Register. Transmission cannot occur without reception.

Before writing the TDR, the PCS field must be set in order to select a slave.

If new data is written in SPI_TDR during the transfer, it stays in it until the current transfer is completed. Then, the received data is transferred from the Shift Register to SPI_RDR, the data in SPI_TDR is loaded in the Shift Register and a new transfer starts.

The transfer of a data written in SPI_TDR in the Shift Register is indicated by the TDRE bit (Transmit Data Register Empty) in the Status Register (SPI_SR). When new data is written in SPI_TDR, this bit is cleared. The TDRE bit is used to trigger the Transmit PDC channel.

The end of transfer is indicated by the TXEMPTY flag in the SPI_SR register. If a transfer delay (DLYBCT) is greater than 0 for the last transfer, TXEMPTY is set after the completion of said delay. The master clock (MCK) can be switched off at this time.

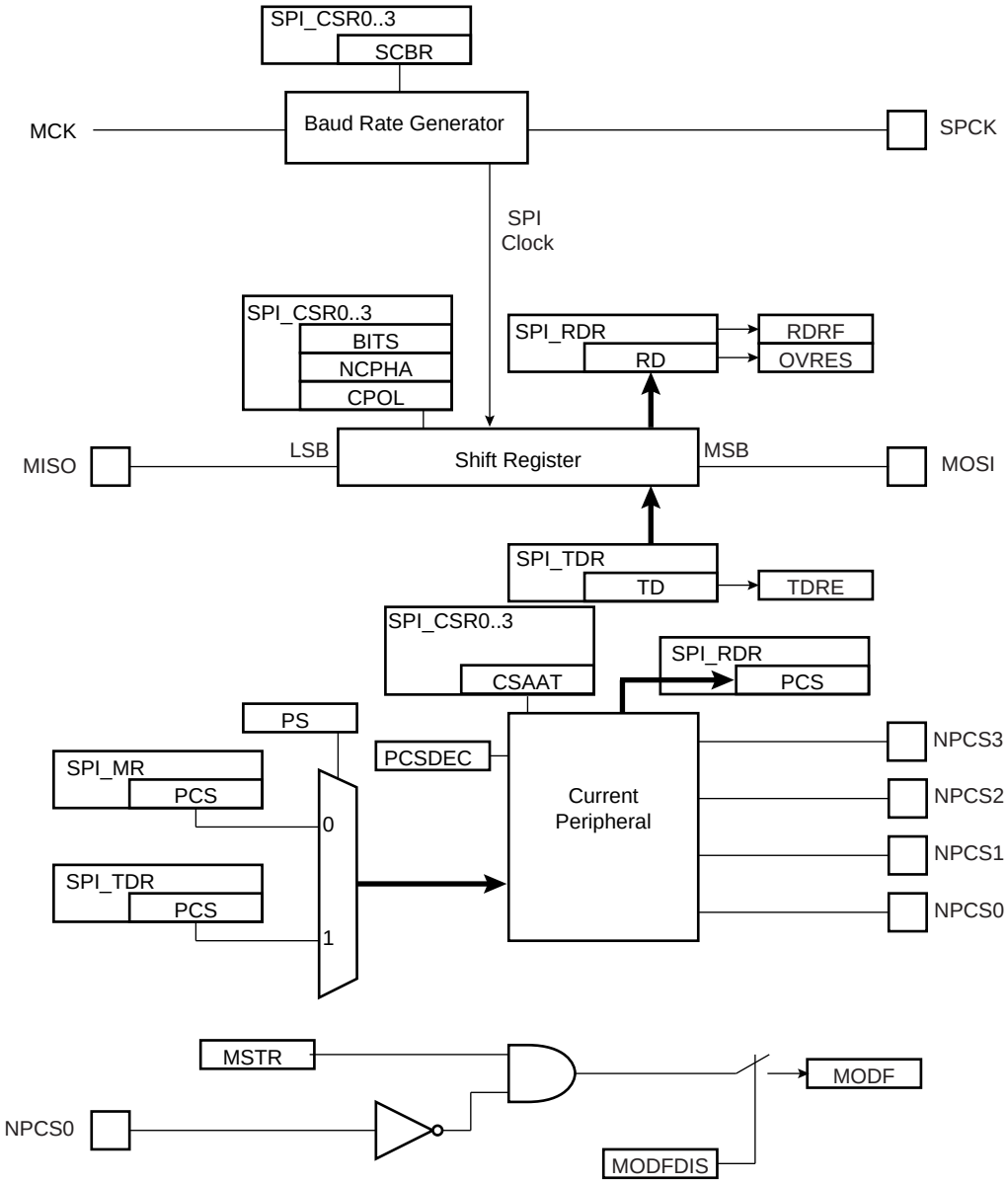
The transfer of received data from the Shift Register in SPI_RDR is indicated by the RDRF bit (Receive Data Register Full) in the Status Register (SPI_SR). When the received data is read, the RDRF bit is cleared.

If the SPI_RDR (Receive Data Register) has not been read before new data is received, the Overrun Error bit (OVRES) in SPI_SR is set. As long as this flag is set, data is loaded in SPI_RDR. The user has to read the status register to clear the OVRES bit.

Figure 28-5, shows a block diagram of the SPI when operating in Master Mode. Figure 28-6 on page 446 shows a flow chart describing how transfers are handled.

### 28.7.3.1 Master Mode Block Diagram

**Figure 28-5. Master Mode Block Diagram**



### 28.7.3.2 Master Mode Flow Diagram

**Figure 28-6. Master Mode Flow Diagram**

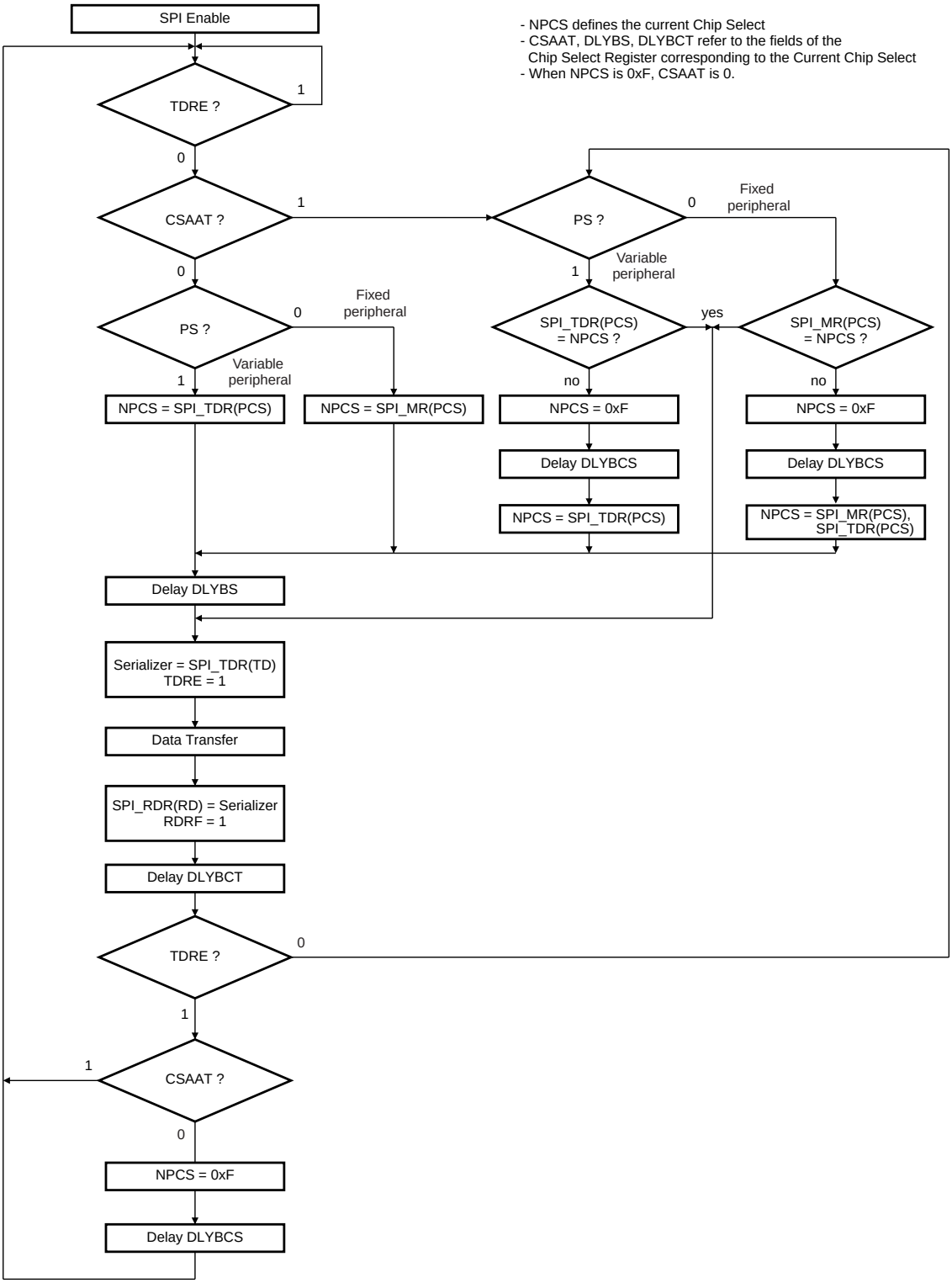


Figure 28-7 shows Transmit Data Register Empty (TDRE), Receive Data Register (RDRF) and Transmission Register Empty (TXEMPTY) status flags behavior within the SPI_SR (Status Register) during an 8-bit data transfer in fixed mode and no Peripheral Data Controller involved.

**Figure 28-7. Status Register Flags Behavior**

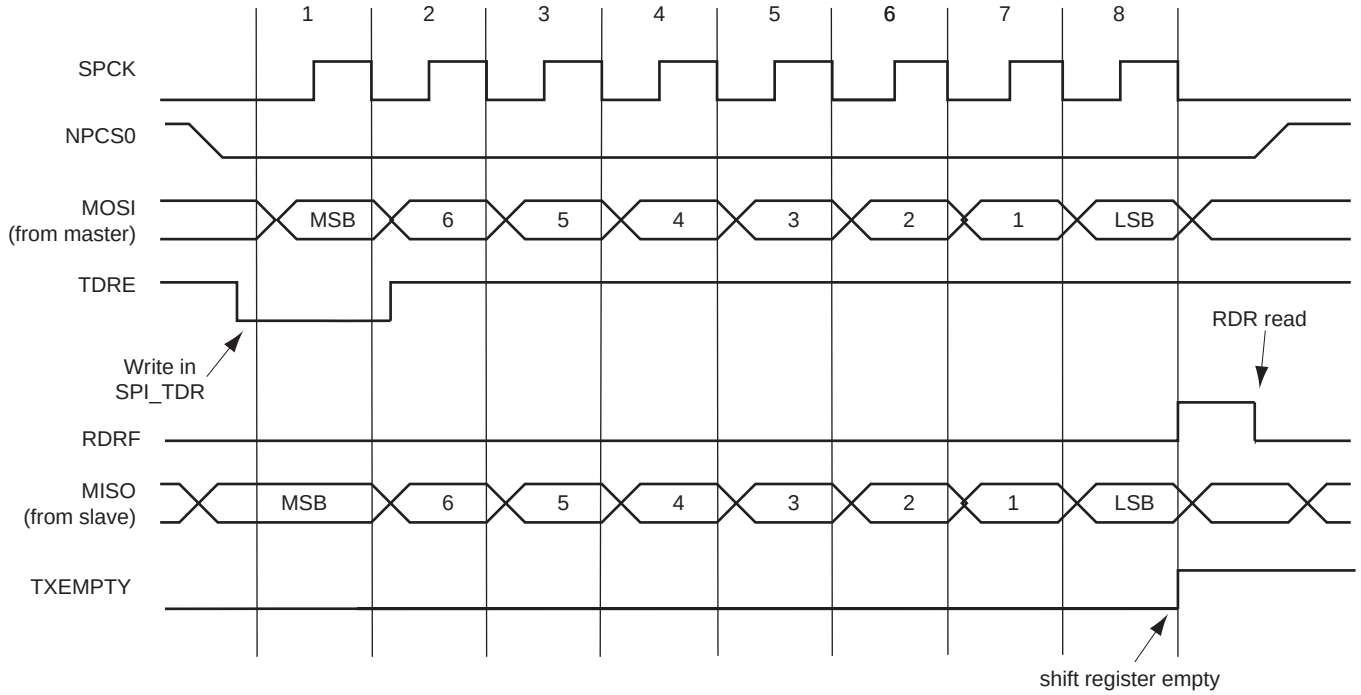
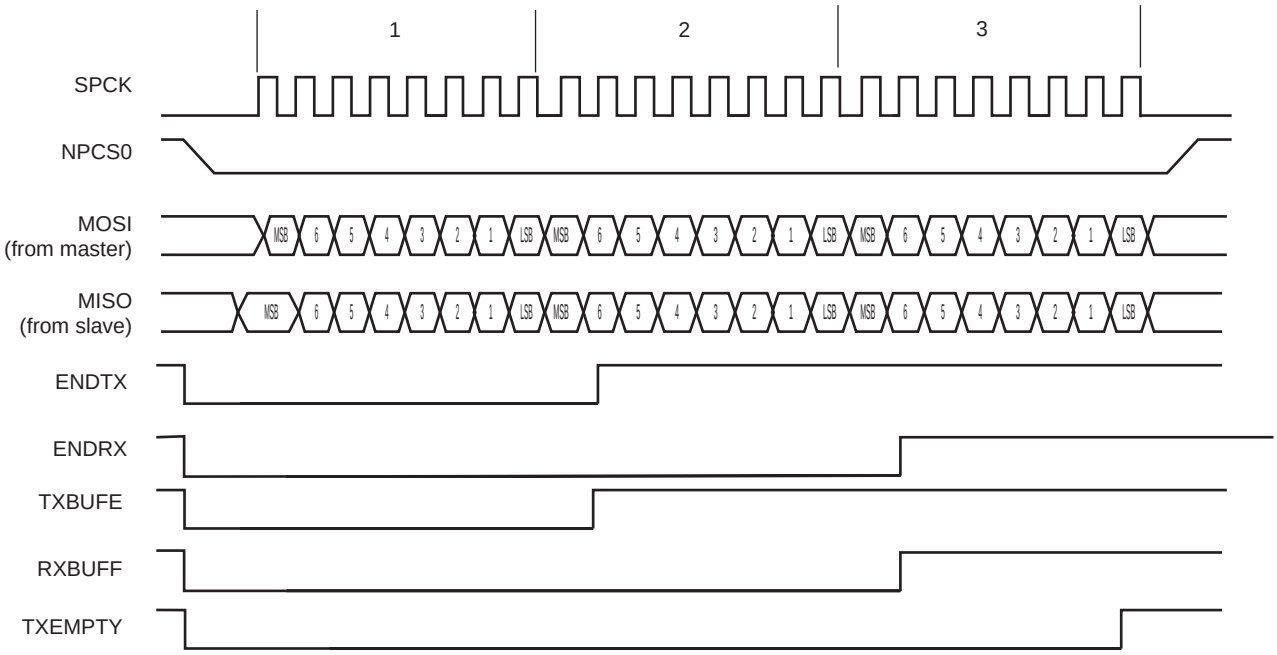



Figure 28-8 shows Transmission Register Empty (TXEMPTY), End of RX buffer (ENDRX), End of TX buffer (ENDTX), RX Buffer Full (RXBUFF) and TX Buffer Empty (TXBUFE) status flags behavior within the SPI_SR (Status Register) during an 8-bit data transfer in fixed mode with the Peripheral Data Controller involved. The PDC is programmed to transfer and receive three data. The next pointer and counter are not used. The RDRF and TDRE are not shown because these flags are managed by the PDC when using the PDC.

**Figure 28-8. PDC Status Register Flags Behavior**




### 28.7.3.3 Clock Generation

The SPI Baud rate clock is generated by dividing the Master Clock (MCK), by a value between 1 and 255.

This allows a maximum operating baud rate at up to Master Clock and a minimum operating baud rate of MCK divided by 255.

Programming the SCBR field at 0 is forbidden. Triggering a transfer while SCBR is at 0 can lead to unpredictable results.

At reset, SCBR is 0 and the user has to program it at a valid value before performing the first transfer.

The divisor can be defined independently for each chip select, as it has to be programmed in the SCBR field of the Chip Select Registers. This allows the SPI to automatically adapt the baud rate for each interfaced peripheral without reprogramming.

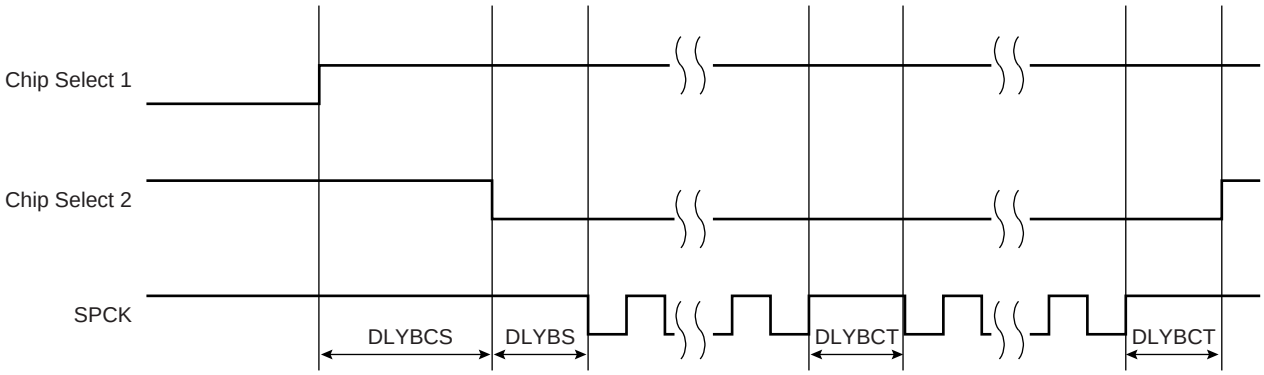
### 28.7.3.4 Transfer Delays

Figure 28-9 shows a chip select transfer change and consecutive transfers on the same chip select. Three delays can be programmed to modify the transfer waveforms:

- The delay between chip selects, programmable only once for all the chip selects by writing the DLYBCS field in the Mode Register. Allows insertion of a delay between release of one chip select and before assertion of a new one.
- The delay before SPCK, independently programmable for each chip select by writing the field DLYBS. Allows the start of SPCK to be delayed after the chip select has been asserted.
- The delay between consecutive transfers, independently programmable for each chip select by writing the DLYBCT field. Allows insertion of a delay between two transfers occurring on the same chip select

These delays allow the SPI to be adapted to the interfaced peripherals and their speed and bus release time.

**Figure 28-9. Programmable Delays**




### 28.7.3.5 Peripheral Selection

The serial peripherals are selected through the assertion of the NPCS0 to NPCS3 signals. By default, all the NPCS signals are high before and after each transfer.

- Fixed Peripheral Select: SPI exchanges data with only one peripheral

Fixed Peripheral Select is activated by writing the PS bit to zero in SPI_MR (Mode Register). In this case, the current peripheral is defined by the PCS field in SPI_MR and the PCS field in the SPI_TDR has no effect.

- Variable Peripheral Select: Data can be exchanged with more than one peripheral without having to reprogram the NPCS field in the SPI_MR register.

Variable Peripheral Select is activated by setting PS bit to one. The PCS field in SPI_TDR is used to select the current peripheral. This means that the peripheral selection can be defined for each new data. The value to write in the SPI_TDR register as the following format.

[xxxxxxx(7-bit) + LASTXFER(1-bit)[0]+ xxxx(4-bit) + PCS (4-bit) + DATA (8 to 16-bit)] with PCS equals to the chip select to assert as defined in Section 28.8.4 (SPI Transmit Data Register) and LASTXFER bit at 0 or 1 depending on CSAAT bit.

Note: 1. Optional.

CSAAT, LASTXFER and CSNAAT bits are discussed in Section 28.7.3.9 ”Peripheral Deselection with PDC” .

If LASTXFER is used, the command must be issued before writing the last character. Instead of LASTXFER, the user can use the SPIDIS command. After the end of the PDC transfer, wait for the TXEMPTY flag, then write SPIDIS into the SPI_CR register (this will not change the configuration register values); the NPCS will be deactivated after the last character transfer. Then, another PDC transfer can be started if the SPIEN was previously written in the SPI_CR register.

### 28.7.3.6 SPI Peripheral DMA Controller (PDC)

In both fixed and variable mode the Peripheral DMA Controller (PDC) can be used to reduce processor overhead.

The Fixed Peripheral Selection allows buffer transfers with a single peripheral. Using the PDC is an optimal means, as the size of the data transfer between the memory and the SPI is either 8 bits or 16 bits. However, changing the peripheral selection requires the Mode Register to be reprogrammed.

The Variable Peripheral Selection allows buffer transfers with multiple peripherals without reprogramming the Mode Register. Data written in SPI_TDR is 32 bits wide and defines the real data to be transmitted and the peripheral it is destined to. Using the PDC in this mode requires 32-bit wide buffers, with the data in the LSBs and the PCS and LASTXFER fields in the MSBs, however the SPI still controls the number of bits (8 to16) to be transferred through MISO and MOSI lines with the chip select configuration registers. This is not the optimal means in term of memory size for the buffers, but it provides a very effective means to exchange data with several peripherals without any intervention of the processor.

*Transfer Size*

Depending on the data size to transmit, from 8 to 16 bits, the PDC manages automatically the type of pointer's size it has to point to. The PDC will perform the following transfer size depending on the mode and number of bits per data.

Fixed Mode:

- 8-bit Data:
  Byte transfer,
  PDC Pointer Address = Address + 1 byte,
  PDC Counter = Counter - 1
- 8-bit to 16-bit Data:
  2 bytes transfer. n-bit data transfer with don't care data (MSB) filled with 0's,
  PDC Pointer Address = Address + 2 bytes,
  PDC Counter = Counter - 1

Variable Mode:

In variable Mode, PDC Pointer Address = Address +4 bytes and PDC Counter = Counter - 1 for 8 to 16-bit transfer size. When using the PDC, the TDRE and RDRF flags are handled by the PDC, thus the user's application does not have to check those bits. Only End of RX Buffer (ENDRX), End of TX Buffer (ENDTX), Buffer Full (RXBUFF), TX Buffer Empty (TXBUFE) are significant. For further details about the Peripheral DMA Controller and user interface, refer to the PDC section of the product datasheet.

#### 28.7.3.7 Peripheral Chip Select Decoding

The user can program the SPI to operate with up to 15 peripherals by decoding the four Chip Select lines, NPCS0 to NPCS3 with 1 of up to 16 decoder/demultiplexer. This can be enabled by writing the PCSDEC bit at 1 in the Mode Register (SPI_MR).

When operating without decoding, the SPI makes sure that in any case only one chip select line is activated, i.e., one NPCS line driven low at a time. If two bits are defined low in a PCS field, only the lowest numbered chip select is driven low.

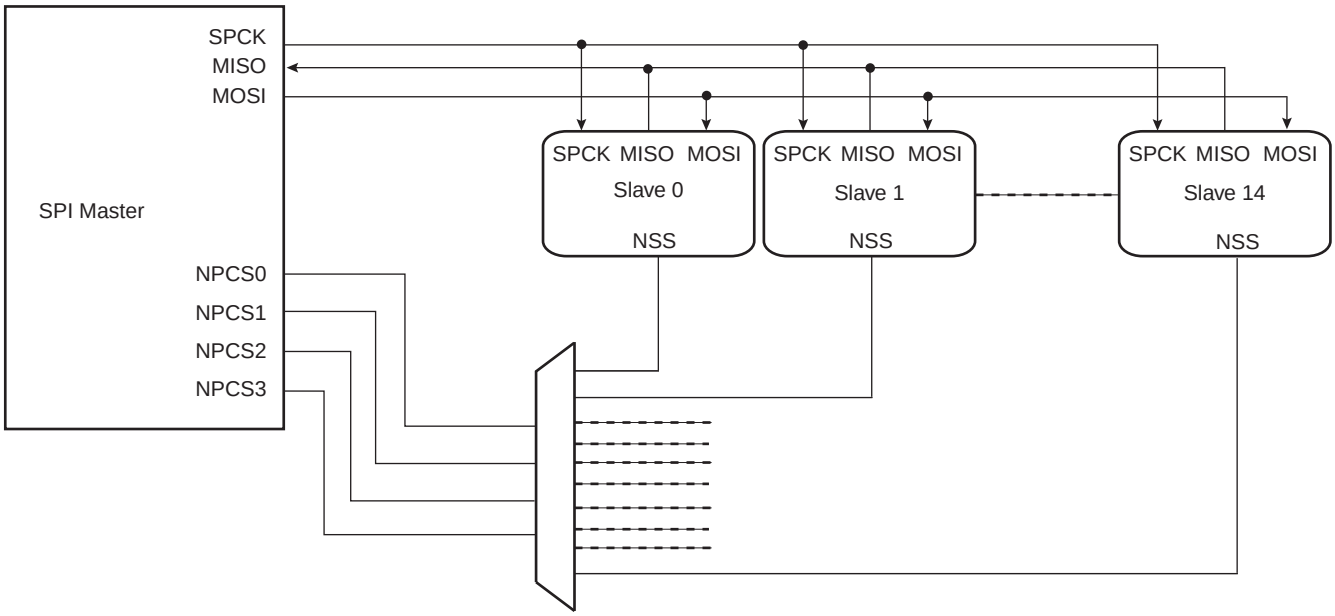
When operating with decoding, the SPI directly outputs the value defined by the PCS field on NPCS lines of either the Mode Register or the Transmit Data Register (depending on PS).

As the SPI sets a default value of 0xF on the chip select lines (i.e. all chip select lines at 1) when not processing any transfer, only 15 peripherals can be decoded.

The SPI has only four Chip Select Registers, not 15. As a result, when decoding is activated, each chip select defines the characteristics of up to four peripherals. As an example, SPI_CRS0 defines the characteristics of the externally decoded peripherals 0 to 3, corresponding to the PCS values 0x0 to 0x3. Thus, the user has to make sure to connect compatible peripherals on the decoded chip select lines 0 to 3, 4 to 7, 8 to 11 and 12 to 14. Figure 28-10 below shows such an implementation.

If the CSAAT bit is used, with or without the PDC, the Mode Fault detection for NPCS0 line must be disabled. This is not needed for all other chip select lines since Mode Fault Detection is only on NPCS0.

**Figure 28-10. Chip Select Decoding Application Block Diagram: Single Master/Multiple Slave Implementation**




### 28.7.3.8 Peripheral Deselection without PDC

During a transfer of more than one data on a Chip Select without the PDC, the SPI_TDR is loaded by the processor, the flag TDRE rises as soon as the content of the SPI_TDR is transferred into the internal shift register. When this flag is detected high, the SPI_TDR can be reloaded. If this reload by the processor occurs before the end of the current transfer and if the next transfer is performed on the same chip select as the current transfer, the Chip Select is not de-asserted between the two transfers. But depending on the application software handling the SPI status register flags (by interrupt or polling method) or servicing other interrupts or other tasks, the processor may not reload the SPI_TDR in time to keep the chip select active (low). A null Delay Between Consecutive Transfer (DLYBCT) value in the SPI_CSR register, will give even less time for the processor to reload the SPI_TDR. With some SPI slave peripherals, requiring the chip select line to remain active (low) during a full set of transfers might lead to communication errors.

To facilitate interfacing with such devices, the Chip Select Register [CSR0...CSR3] can be programmed with the CSAAT bit (Chip Select Active After Transfer) at 1. This allows the chip select lines to remain in their current state (low = active) until transfer to another chip select is required. Even if the SPI_TDR is not reloaded the chip select will remain active. To have the chip select line to raise at the end of the transfer the Last transfer Bit (LASTXFER) in the SPI_MR register must be set at 1 before writing the last data to transmit into the SPI_TDR.

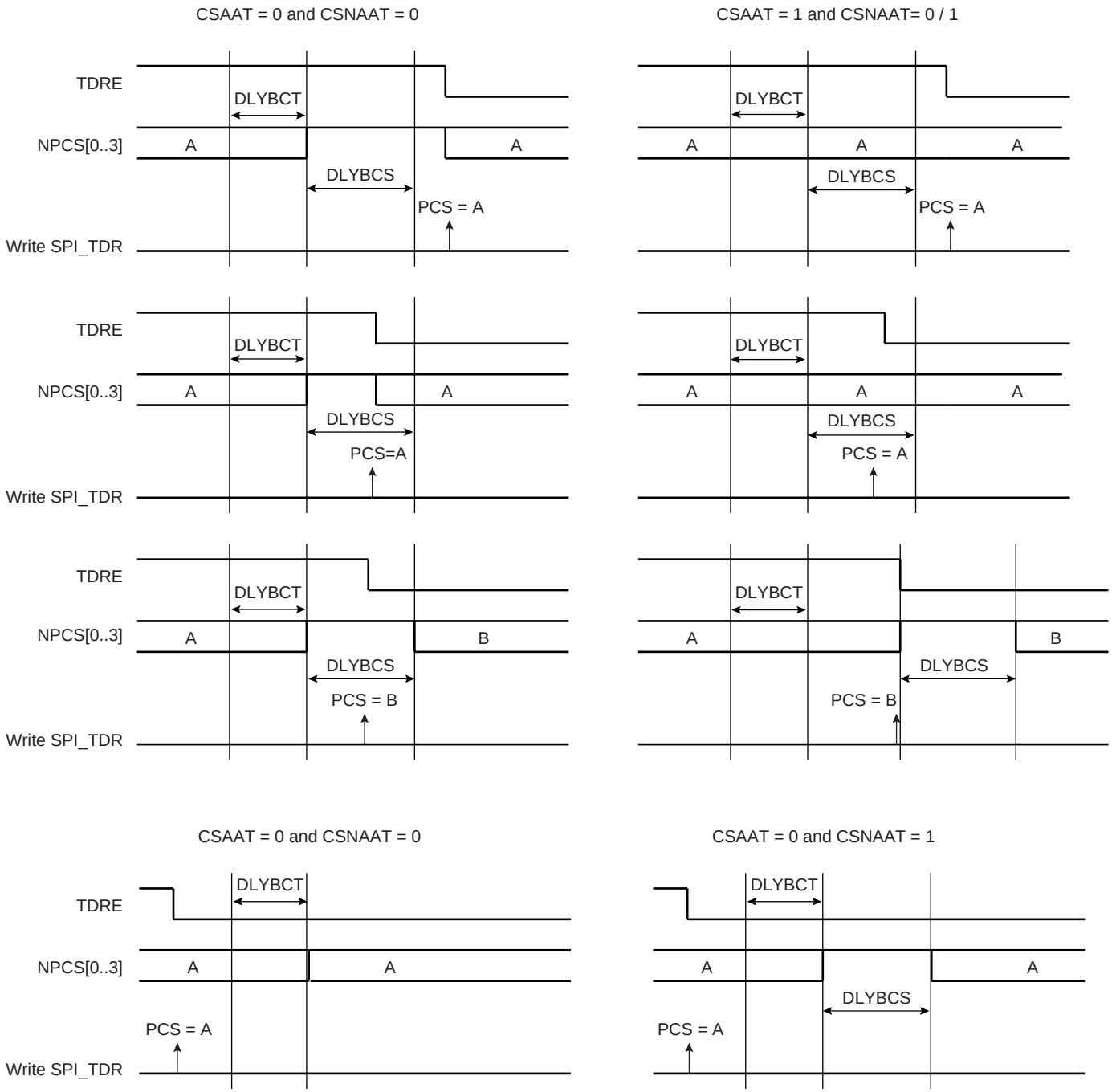
### 28.7.3.9 Peripheral Deselection with PDC

When the Peripheral DMA Controller is used, the chip select line will remain low during the whole transfer since the TDRE flag is managed by the PDC itself. The reloading of the SPI_TDR by the PDC is done as soon as TDRE flag is set to one. In this case the use of CSAAT bit might not be needed. However, it may happen that when other PDC channels connected to other peripherals are in use as well, the SPI PDC might be delayed by another (PDC with a higher priority on the bus). Having PDC buffers in slower memories like flash memory or SDRAM compared to fast internal SRAM, may lengthen the reload time of the SPI_TDR by the PDC as well. This means that the SPI_TDR might not be reloaded in time to keep the chip select line low. In this case the chip select line may toggle between data transfer and according to some SPI Slave devices, the communication might get lost. The use of the CSAAT bit might be needed.

When the CSAAT bit is set at 0, the NPCS does not rise in all cases between two transfers on the same peripheral. During a transfer on a Chip Select, the flag TDRE rises as soon as the content of the SPI_TDR is transferred into the internal shifter. When this flag is detected the SPI_TDR can be reloaded. If this reload occurs before the end of the current transfer and if the next transfer is performed on the same chip select as the current transfer, the Chip Select is not de-asserted between the two transfers. This might lead to difficulties for interfacing with some serial peripherals requiring the chip select to be de-asserted after each transfer. To facilitate interfacing with such devices, the Chip Select Register can be programmed with the CSNAAT bit (Chip Select Not Active After Transfer) at 1. This allows to de-assert systematically the chip select lines during a time DLYBCS. (The value of the CSNAAT bit is taken into account only if the CSAAT bit is set at 0 for the same Chip Select).

Figure 28-11 shows different peripheral deselection cases and the effect of the CSAAT and CSNAAT bits.

**Figure 28-11. Peripheral Deselection**




### 28.7.3.10Mode Fault Detection

A mode fault is detected when the SPI is programmed in Master Mode and a low level is driven by an external master on the NPCS0/NSS signal. In this case, multi-master configuration, NPCS0, MOSI, MISO and SPCK pins must be configured in open drain (through the PIO controller). When a mode fault is detected, the MODF bit in the SPI_SR is set until the SPI_SR is read and the SPI is automatically disabled until re-enabled by writing the SPIEN bit in the SPI_CR (Control Register) at 1.

By default, the Mode Fault detection circuitry is enabled. The user can disable Mode Fault detection by setting the MODFDIS bit in the SPI Mode Register (SPI_MR).

### 28.7.4 SPI Slave Mode

When operating in Slave Mode, the SPI processes data bits on the clock provided on the SPI clock pin (SPCK).

The SPI waits for NSS to go active before receiving the serial clock from an external master. When NSS falls, the clock is validated on the serializer, which processes the number of bits defined by the BITS field of the Chip Select Register 0 (SPI_CSR0). These bits are processed following a phase and a polarity defined respectively by the NCPHA and CPOL bits of the SPI_CSR0. Note that BITS, CPOL and NCPHA of the other Chip Select Registers have no effect when the SPI is programmed in Slave Mode.

The bits are shifted out on the MISO line and sampled on the MOSI line.

(For more information on BITS field, see also, the (Note:) below the register table; Section 28.8.9 “SPI Chip Select Register” on page 467.)

When all the bits are processed, the received data is transferred in the Receive Data Register and the RDRF bit rises. If the SPI_RDR (Receive Data Register) has not been read before new data is received, the Overrun Error bit (OVRES) in SPI_SR is set. As long as this flag is set, data is loaded in SPI_RDR. The user has to read the status register to clear the OVRES bit.

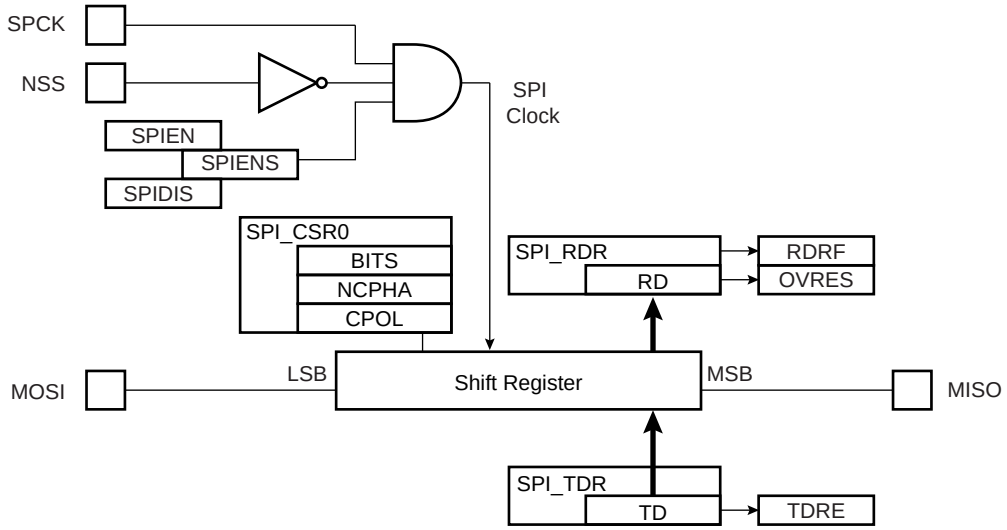
When a transfer starts, the data shifted out is the data present in the Shift Register. If no data has been written in the Transmit Data Register (SPI_TDR), the last data received is transferred. If no data has been received since the last reset, all bits are transmitted low, as the Shift Register resets at 0.

When a first data is written in SPI_TDR, it is transferred immediately in the Shift Register and the TDRE bit rises. If new data is written, it remains in SPI_TDR until a transfer occurs, i.e. NSS falls and there is a valid clock on the SPCK pin. When the transfer occurs, the last data written in SPI_TDR is transferred in the Shift Register and the TDRE bit rises. This enables frequent updates of critical variables with single transfers.

Then, a new data is loaded in the Shift Register from the Transmit Data Register. In case no character is ready to be transmitted, i.e. no character has been written in SPI_TDR since the last load from SPI_TDR to the Shift Register, the Shift Register is not modified and the last received character is retransmitted. In this case the Underrun Error Status Flag (UNDES) is set in the SPI_SR.

Figure 28-12 shows a block diagram of the SPI when operating in Slave Mode.

**Figure 28-12. Slave Mode Functional Bloc Diagram**



### 28.7.5 Write Protected Registers

To prevent any single software error that may corrupt SPI behavior, the registers listed below can be write-protected by setting the SPIWPEN bit in the SPI Write Protection Mode Register (SPI_WPMR).

If a write access in a write-protected register is detected, then the SPIWPVS flag in the SPI Write Protection Status Register (SPI_WPSR) is set and the field SPIWPVSRC indicates in which register the write access has been attempted.

The SPIWPVS flag is automatically reset after reading the SPI Write Protection Status Register (SPI_WPSR).

List of the write-protected registers:

Section 28.8.2 ”SPI Mode Register”

Section 28.8.9 ”SPI Chip Select Register”

## 28.8 Serial Peripheral Interface (SPI) User Interface

**Table 28-5. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x00 | Control Register | SPI_CR | Write-only | --- |
| 0x04 | Mode Register | SPI_MR | Read-write | 0x0 |
| 0x08 | Receive Data Register | SPI_RDR | Read-only | 0x0 |
| 0x0C | Transmit Data Register | SPI_TDR | Write-only | --- |
| 0x10 | Status Register | SPI_SR | Read-only | 0x000000F0 |
| 0x14 | Interrupt Enable Register | SPI_IER | Write-only | --- |
| 0x18 | Interrupt Disable Register | SPI_IDR | Write-only | --- |
| 0x1C | Interrupt Mask Register | SPI_IMR | Read-only | 0x0 |
| 0x20 - 0x2C | Reserved | | | |
| 0x30 | Chip Select Register 0 | SPI_CSR0 | Read-write | 0x0 |
| 0x34 | Chip Select Register 1 | SPI_CSR1 | Read-write | 0x0 |
| 0x38 | Chip Select Register 2 | SPI_CSR2 | Read-write | 0x0 |
| 0x3C | Chip Select Register 3 | SPI_CSR3 | Read-write | 0x0 |
| 0x4C - 0xE0 | Reserved | – | – | – |
| 0xE4 | Write Protection Control Register | SPI_WPMR | Read-write | 0x0 |
| 0xE8 | Write Protection Status Register | SPI_WPSR | Read-only | 0x0 |
| 0x00E8 - 0x00F8 | Reserved | – | – | – |
| 0x00FC | Reserved | – | – | – |
| 0x100 - 0x124 | Reserved for the PDC | – | – | – |

### 28.8.1 SPI Control Register

**Name:** SPI_CR

**Address:** 0x40008000

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | LASTXFER |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SWRST | – | – | – | – | – | SPIDIS | SPIEN |

- **SPIEN: SPI Enable**

0 = No effect.

1 = Enables the SPI to transfer and receive data.

- **SPIDIS: SPI Disable**

0 = No effect.

1 = Disables the SPI.

As soon as SPIDIS is set, SPI finishes its transfer.

All pins are set in input mode and no data is received or transmitted.

If a transfer is in progress, the transfer is finished before the SPI is disabled.

If both SPIEN and SPIDIS are equal to one when the control register is written, the SPI is disabled.

- **SWRST: SPI Software Reset**

0 = No effect.

1 = Reset the SPI. A software-triggered hardware reset of the SPI interface is performed.

The SPI is in slave mode after software reset.

PDC channels are not affected by software reset.

- **LASTXFER: Last Transfer**

0 = No effect.

1 = The current NPCS will be deasserted after the character written in TD has been transferred. When CSAAT is set, this allows to close the communication with the current serial peripheral by raising the corresponding NPCS line as soon as TD transfer has completed.

Refer to Section 28.7.3.5 ”Peripheral Selection” for more details.

### 28.8.2 SPI Mode Register

**Name:** SPI_MR

**Address:** 0x40008004

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| DLYBCS | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | PCS | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| LLB | – | WDRBT | MODFDIS | – | PCSDEC | PS | MSTR |

- **MSTR: Master/Slave Mode**

0 = SPI is in Slave mode.

1 = SPI is in Master mode.

- **PS: Peripheral Select**

0 = Fixed Peripheral Select.

1 = Variable Peripheral Select.

- **PCSDEC: Chip Select Decode**

0 = The chip selects are directly connected to a peripheral device.

1 = The four chip select lines are connected to a 4- to 16-bit decoder.

When PCSDEC equals one, up to 15 Chip Select signals can be generated with the four lines using an external 4- to 16-bit decoder. The Chip Select Registers define the characteristics of the 15 chip selects according to the following rules:

SPI_CSR0 defines peripheral chip select signals 0 to 3.

SPI_CSR1 defines peripheral chip select signals 4 to 7.

SPI_CSR2 defines peripheral chip select signals 8 to 11.

SPI_CSR3 defines peripheral chip select signals 12 to 14.

- **MODFDIS: Mode Fault Detection**

0 = Mode fault detection is enabled.

1 = Mode fault detection is disabled.

- **WDRBT: Wait Data Read Before Transfer**

0 = No Effect. In master mode, a transfer can be initiated whatever the state of the Receive Data Register is.

1 = In Master Mode, a transfer can start only if the Receive Data Register is empty, i.e. does not contain any unread data. This mode prevents overrun error in reception.

- **LLB: Local Loopback Enable**

0 = Local loopback path disabled.

1 = Local loopback path enabled

LLB controls the local loopback on the data serializer for testing in Master Mode only. (MISO is internally connected on MOSI.)

- **PCS: Peripheral Chip Select**

This field is only used if Fixed Peripheral Select is active (PS = 0).

If PCSDEC = 0:

| | |
|---|---|
| PCS = xxx0 | NPCS[3:0] = 1110 |
| PCS = xx01 | NPCS[3:0] = 1101 |
| PCS = x011 | NPCS[3:0] = 1011 |
| PCS = 0111 | NPCS[3:0] = 0111 |
| PCS = 1111 | forbidden (no peripheral is selected) |

(x = don’t care)

If PCSDEC = 1:

NPCS[3:0] output signals = PCS.

- **DLYBCS: Delay Between Chip Selects**

This field defines the delay from NPCS inactive to the activation of another NPCS. The DLYBCS time guarantees non-overlapping chip selects and solves bus contentions in case of peripherals having long data float times.

If DLYBCS is less than or equal to six, six MCK periods will be inserted by default.

Otherwise, the following equation determines the delay:

$$\text{Delay Between Chip Selects} = \frac{DLYBCS}{MCK}$$

### 28.8.3 SPI Receive Data Register

**Name:** SPI_RDR

**Address:** 0x40008008

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | PCS | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| RD | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RD | | | | | | | |

- **RD: Receive Data**

Data received by the SPI Interface is stored in this register right-justified. Unused bits read zero.

- **PCS: Peripheral Chip Select**

In Master Mode only, these bits indicate the value on the NPCS pins at the end of a transfer. Otherwise, these bits read zero.

Note: When using variable peripheral select mode (PS = 1 in SPI_MR) it is mandatory to also set the WDRBT field to 1 if the SPI_RDR PCS field is to be processed.

### 28.8.4 SPI Transmit Data Register

**Name:** SPI_TDR

**Address:** 0x4000800C

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | LASTXFER |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | PCS | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| TD | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| TD | | | | | | | |

- **TD: Transmit Data**

Data to be transmitted by the SPI Interface is stored in this register. Information to be transmitted must be written to the transmit data register in a right-justified format.

- **PCS: Peripheral Chip Select**

This field is only used if Variable Peripheral Select is active (PS = 1).

If PCSDEC = 0:

PCS = xxx0 NPCS[3:0] = 1110

PCS = xx01 NPCS[3:0] = 1101

PCS = x011 NPCS[3:0] = 1011

PCS = 0111 NPCS[3:0] = 0111

PCS = 1111 forbidden (no peripheral is selected)

(x = don’t care)

If PCSDEC = 1:

NPCS[3:0] output signals = PCS

- **LASTXFER: Last Transfer**

0 = No effect.

1 = The current NPCS will be deasserted after the character written in TD has been transferred. When CSAAT is set, this allows to close the communication with the current serial peripheral by raising the corresponding NPCS line as soon as TD transfer has completed.

This field is only used if Variable Peripheral Select is active (PS = 1).

### 28.8.5 SPI Status Register

**Name:** SPI_SR

**Address:** 0x40008010

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | SPIENS |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | UNDES | TXEMPTY | NSSR |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXBUFE | RXBUFF | ENDTX | ENDRX | OVRES | MODF | TDRE | RDRF |

- **RDRF: Receive Data Register Full**

0 = No data has been received since the last read of SPI_RDR

1 = Data has been received and the received data has been transferred from the serializer to SPI_RDR since the last read of SPI_RDR.

- **TDRE: Transmit Data Register Empty**

0 = Data has been written to SPI_TDR and not yet transferred to the serializer.

1 = The last data written in the Transmit Data Register has been transferred to the serializer.

TDRE equals zero when the SPI is disabled or at reset. The SPI enable command sets this bit to one.

- **MODF: Mode Fault Error**

0 = No Mode Fault has been detected since the last read of SPI_SR.

1 = A Mode Fault occurred since the last read of the SPI_SR.

- **OVRES: Overrun Error Status**

0 = No overrun has been detected since the last read of SPI_SR.

1 = An overrun has occurred since the last read of SPI_SR.

An overrun occurs when SPI_RDR is loaded at least twice from the serializer since the last read of the SPI_RDR.

- **ENDRX: End of RX buffer**

0 = The Receive Counter Register has not reached 0 since the last write in SPI_RCR(1) or SPI_RNCR(1).

1 = The Receive Counter Register has reached 0 since the last write in SPI_RCR(1) or SPI_RNCR(1).

- **ENDTX: End of TX buffer**

0 = The Transmit Counter Register has not reached 0 since the last write in SPI_TCR(1) or SPI_TNCR(1).

1 = The Transmit Counter Register has reached 0 since the last write in SPI_TCR(1) or SPI_TNCR(1).

- **RXBUFF: RX Buffer Full**

0 = SPI_RCR(1) or SPI_RNCR(1) has a value other than 0.

1 = Both SPI_RCR(1) and SPI_RNCR(1) have a value of 0.

- **TXBUFE: TX Buffer Empty**

0 = SPI_TCR(1) or SPI_TNCR(1) has a value other than 0.

1 = Both SPI_TCR(1) and SPI_TNCR(1) have a value of 0.

- **NSSR: NSS Rising**

0 = No rising edge detected on NSS pin since last read.

1 = A rising edge occurred on NSS pin since last read.

- **TXEMPTY: Transmission Registers Empty**

0 = As soon as data is written in SPI_TDR.

1 = SPI_TDR and internal shifter are empty. If a transfer delay has been defined, TXEMPTY is set after the completion of such delay.

- **UNDES: Underrun Error Status (Slave Mode Only)**

0 = No underrun has been detected since the last read of SPI_SR.

1 = A transfer begins whereas no data has been loaded in the Transmit Data Register.

- **SPIENS: SPI Enable Status**

0 = SPI is disabled.

1 = SPI is enabled.

Note: 1. SPI_RCR, SPI_RNCR, SPI_TCR, SPI_TNCR are physically located in the PDC.

### 28.8.6 SPI Interrupt Enable Register

**Name:** SPI_IER

**Address:** 0x40008014

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | UNDES | TXEMPTY | NSSR |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXBUFE | RXBUFF | ENDTX | ENDRX | OVRES | MODF | TDRE | RDRF |

0 = No effect.

1 = Enables the corresponding interrupt.

- **RDRF: Receive Data Register Full Interrupt Enable**
- **TDRE: SPI Transmit Data Register Empty Interrupt Enable**
- **MODF: Mode Fault Error Interrupt Enable**
- **OVRES: Overrun Error Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable**
- **ENDTX: End of Transmit Buffer Interrupt Enable**
- **RXBUFF: Receive Buffer Full Interrupt Enable**
- **TXBUFE: Transmit Buffer Empty Interrupt Enable**
- **NSSR: NSS Rising Interrupt Enable**
- **TXEMPTY: Transmission Registers Empty Enable**
- **UNDES: Underrun Error Interrupt Enable**

### 28.8.7 SPI Interrupt Disable Register

**Name:** SPI_IDR

**Address:** 0x40008018

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | UNDES | TXEMPTY | NSSR |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXBUFE | RXBUFF | ENDTX | ENDRX | OVRES | MODF | TDRE | RDRF |

0 = No effect.

1 = Disables the corresponding interrupt.

- **RDRF: Receive Data Register Full Interrupt Disable**
- **TDRE: SPI Transmit Data Register Empty Interrupt Disable**
- **MODF: Mode Fault Error Interrupt Disable**
- **OVRES: Overrun Error Interrupt Disable**
- **ENDRX: End of Receive Buffer Interrupt Disable**
- **ENDTX: End of Transmit Buffer Interrupt Disable**
- **RXBUFF: Receive Buffer Full Interrupt Disable**
- **TXBUFE: Transmit Buffer Empty Interrupt Disable**
- **NSSR: NSS Rising Interrupt Disable**
- **TXEMPTY: Transmission Registers Empty Disable**
- **UNDES: Underrun Error Interrupt Disable**

### 28.8.8 SPI Interrupt Mask Register

**Name:** SPI_IMR

**Address:** 0x4000801C

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | UNDES | TXEMPTY | NSSR |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXBUFE | RXBUFF | ENDTX | ENDRX | OVRES | MODF | TDRE | RDRF |

0 = The corresponding interrupt is not enabled.

1 = The corresponding interrupt is enabled.

- **RDRF: Receive Data Register Full Interrupt Mask**
- **TDRE: SPI Transmit Data Register Empty Interrupt Mask**
- **MODF: Mode Fault Error Interrupt Mask**
- **OVRES: Overrun Error Interrupt Mask**
- **ENDRX: End of Receive Buffer Interrupt Mask**
- **ENDTX: End of Transmit Buffer Interrupt Mask**
- **RXBUFF: Receive Buffer Full Interrupt Mask**
- **TXBUFE: Transmit Buffer Empty Interrupt Mask**
- **NSSR: NSS Rising Interrupt Mask**
- **TXEMPTY: Transmission Registers Empty Mask**
- **UNDES: Underrun Error Interrupt Mask**

### 28.8.9 SPI Chip Select Register

**Name:** SPI_CSRx[x=0..3]

**Address:** 0x40008030

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| DLYBCT | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| DLYBS | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| SCBR | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| BITS | | | | CSAAT | CSNAAT | NCPHA | CPOL |

Note: SPI_CSRx registers must be written even if the user wants to use the defaults. The BITS field will not be updated with the translated value unless the register is written.

- **CPOL: Clock Polarity**

0 = The inactive state value of SPCK is logic level zero.

1 = The inactive state value of SPCK is logic level one.

CPOL is used to determine the inactive state value of the serial clock (SPCK). It is used with NCPHA to produce the required clock/data relationship between master and slave devices.

- **NCPHA: Clock Phase**

0 = Data is changed on the leading edge of SPCK and captured on the following edge of SPCK.

1 = Data is captured on the leading edge of SPCK and changed on the following edge of SPCK.

NCPHA determines which edge of SPCK causes data to change and which edge causes data to be captured. NCPHA is used with CPOL to produce the required clock/data relationship between master and slave devices.

- **CSNAAT: Chip Select Not Active After Transfer (Ignored if CSAAT = 1)**

0 = The Peripheral Chip Select does not rise between two transfers if the SPI_TDR is reloaded before the end of the first transfer and if the two transfers occur on the same Chip Select.

1 = The Peripheral Chip Select rises systematically between each transfer performed on the same slave for a minimal duration of:

– $\frac{DLYBCS}{MCK}$ (if DLYBCT field is different from 0)

– $\frac{DLYBCS + 1}{MCK}$ (if DLYBCT field equal 0)

- **CSAAT: Chip Select Active After Transfer**

0 = The Peripheral Chip Select Line rises as soon as the last transfer is achieved.

1 = The Peripheral Chip Select does not rise after the last transfer is achieved. It remains active until a new transfer is requested on a different chip select.

- **BITS: Bits Per Transfer**

(See the (Note:) below the register table; Section 28.8.9 “SPI Chip Select Register” on page 467.)

The BITS field determines the number of data bits transferred. Reserved values should not be used.

| Value | Name | Description |
|---|---|---|
| 0 | 8_BIT | 8_bits for transfer |
| 1 | 9_BIT | 9_bits for transfer |
| 2 | 10_BIT | 8_bits for transfer |
| 3 | 11_BIT | 8_bits for transfer |
| 4 | 12_BIT | 8_bits for transfer |
| 5 | 13_BIT | 8_bits for transfer |
| 6 | 14_BIT | 8_bits for transfer |
| 7 | 15_BIT | 8_bits for transfer |
| 8 | 16_BIT | 8_bits for transfer |
| 10 | – | Reserved |
| 11 | – | Reserved |
| 12 | – | Reserved |
| 13 | – | Reserved |
| 14 | – | Reserved |
| 15 | – | Reserved |
| 16 | – | Reserved |

- **SCBR: Serial Clock Baud Rate**

In Master Mode, the SPI Interface uses a modulus counter to derive the SPCK baud rate from the Master Clock MCK. The Baud rate is selected by writing a value from 1 to 255 in the SCBR field. The following equations determine the SPCK baud rate:

$$\text{SPCK Baudrate} = \frac{MCK}{SCBR}$$

Programming the SCBR field at 0 is forbidden. Triggering a transfer while SCBR is at 0 can lead to unpredictable results.

At reset, SCBR is 0 and the user has to program it at a valid value before performing the first transfer.

Note: If one of the SCBR fields inSPI_CSRx is set to 1, the other SCBR fields in SPI_CSRx must be set to 1 as well, if they are required to process transfers. If they are not used to transfer data, they can be set at any value.

- **DLYBS: Delay Before SPCK**

This field defines the delay from NPCS valid to the first valid SPCK transition.

When DLYBS equals zero, the NPCS valid to SPCK transition is 1/2 the SPCK clock period.

Otherwise, the following equations determine the delay:

$$\text{Delay Before SPCK} = \frac{DLYBS}{MCK}$$

- **DLYBCT: Delay Between Consecutive Transfers**

This field defines the delay between two consecutive transfers with the same peripheral without removing the chip select. The delay is always inserted after each transfer and before removing the chip select if needed.

When DLYBCT equals zero, no delay between consecutive transfers is inserted and the clock keeps its duty cycle over the character transfers.

Otherwise, the following equation determines the delay:

$$\text{Delay Between Consecutive Transfers} = \frac{32 \times DLYBCT}{MCK}$$

### 28.8.10 SPI Write Protection Mode Register

**Name:** SPI_WPMR

**Address:** 0x400080E4

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| SPIWPKEY | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| SPIWPKEY | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| SPIWPKEY | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| - | - | - | - | - | - | - | SPIWPEN |

- **SPIWPEN: SPI Write Protection Enable**

0: The Write Protection is Disabled

1: The Write Protection is Enabled

- **SPIWPKEY: SPI Write Protection Key Password**

If a value is written in SPIWPEN, the value is taken into account only if SPIWPKEY is written with “SPI” (SPI written in ASCII Code, ie 0x535049 in hexadecimal).

### 28.8.11 SPI Write Protection Status Register

**Name:** SPI_WPSR

**Address:** 0x400080E8

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| SPIWPVSRC | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | SPIWPVS | | |

- **SPIWPVS: SPI Write Protection Violation Status**

| SPIWPVS value | Violation Type |
|---|---|
| 0x1 | The Write Protection has blocked a Write access to a protected register (since the last read). |
| 0x2 | Software Reset has been performed while Write Protection was enabled (since the last read or since the last write access on SPI_MR, SPI_IER, SPI_IDR or SPI_CSRx). |
| 0x3 | Both Write Protection violation and software reset with Write Protection enabled have occurred since the last read. |
| 0x4 | Write accesses have been detected on SPI_MR (while a chip select was active) or on SPI_CSRi (while the Chip Select “i” was active) since the last read. |
| 0x5 | The Write Protection has blocked a Write access to a protected register and write accesses have been detected on SPI_MR (while a chip select was active) or on SPI_CSRi (while the Chip Select “i” was active) since the last read. |
| 0x6 | Software Reset has been performed while Write Protection was enabled (since the last read or since the last write access on SPI_MR, SPI_IER, SPI_IDR or SPI_CSRx) and some write accesses have been detected on SPI_MR (while a chip select was active) or on SPI_CSRi (while the Chip Select “i” was active) since the last read. |
| 0x7 | - The Write Protection has blocked a Write access to a protected register.<br>and<br>- Software Reset has been performed while Write Protection was enabled.<br>and<br>- Write accesses have been detected on SPI_MR (while a chip select was active) or on SPI_CSRi (while the Chip Select “i” was active) since the last read. |

- **SPIWPVSRC: SPI Write Protection Violation Source**

This Field indicates the APB Offset of the register concerned by the violation (SPI_MR or SPI_CSRx)

# 29. Two-wire Interface (TWI)

## 29.1 Description

The Atmel Two-wire Interface (TWI) interconnects components on a unique two-wire bus, made up of one clock line and one data line with speeds of up to 400 Kbits per second, based on a byte-oriented transfer format. It can be used with any Atmel Two-wire Interface bus Serial EEPROM and I²C compatible device such as Real Time Clock (RTC), Dot Matrix/Graphic LCD Controllers and Temperature Sensor, to name but a few. The TWI is programmable as a master or a slave with sequential or single-byte access. Multiple master capability is supported. 20

Arbitration of the bus is performed internally and puts the TWI in slave mode automatically if the bus arbitration is lost.

A configurable baud rate generator permits the output data rate to be adapted to a wide range of core clock frequencies.

Below, Table 29-1 lists the compatibility level of the Atmel Two-wire Interface in Master Mode and a full I2C compatible device.

**Table 29-1. Atmel TWI compatibility with i2C Standard**

| I2C Standard | Atmel TWI |
|---|---|
| Standard Mode Speed (100 KHz) | Supported |
| Fast Mode Speed (400 KHz) | Supported |
| 7 or 10 bits Slave Addressing | Supported |
| START BYTE(1) | Not Supported |
| Repeated Start (Sr) Condition | Supported |
| ACK and NACK Management | Supported |
| Slope control and input filtering (Fast mode) | Not Supported |
| Clock stretching | Supported |
| Multi Master Capability | Supported |

Note: 1. START + b000000001 + Ack + Sr

## 29.2 Embedded Characteristics

- Two TWIs
- Compatible with Atmel Two-wire Interface Serial Memory and I²C Compatible Devices(Note:)
- One, Two or Three Bytes for Slave Address
- Sequential Read-write Operations
- Master, Multi-master and Slave Mode Operation
- Bit Rate: Up to 400 Kbits
- General Call Supported in Slave mode
- SMBUS Quick Command Supported in Master Mode
- Connection to Peripheral DMA Controller (PDC) Channel Capabilities Optimizes Data Transfers in Master Mode Only
  - One Channel for the Receiver, One Channel for the Transmitter
  - Next Buffer Support
- Connection to DMA Controller (DMAC) Channel Capabilities Optimizes Data Transfers in Master Mode Only

Note: See Table 29-1 for details on compatibility with I²C Standard.

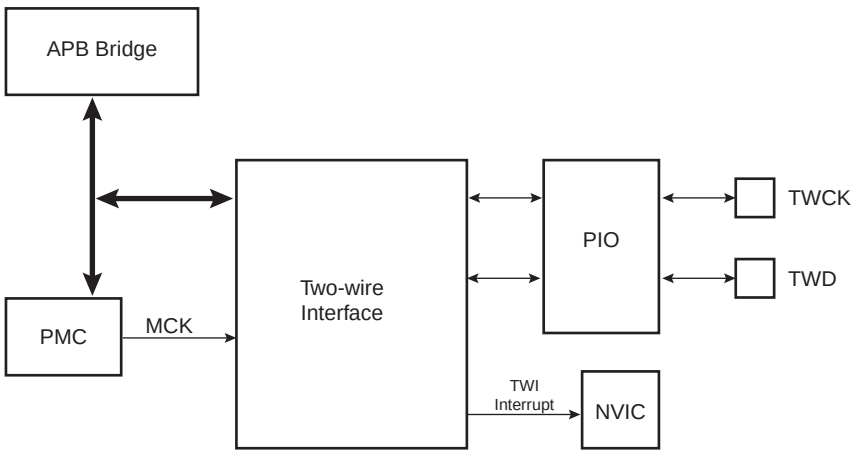
## 29.3 List of Abbreviations

**Table 29-2. Abbreviations**

| Abbreviation | Description |
|---|---|
| TWI | Two-wire Interface |
| A | Acknowledge |
| NA | Non Acknowledge |
| P | Stop |
| S | Start |
| Sr | Repeated Start |
| SADR | Slave Address |
| ADR | Any address except SADR |
| R | Read |
| W | Write |

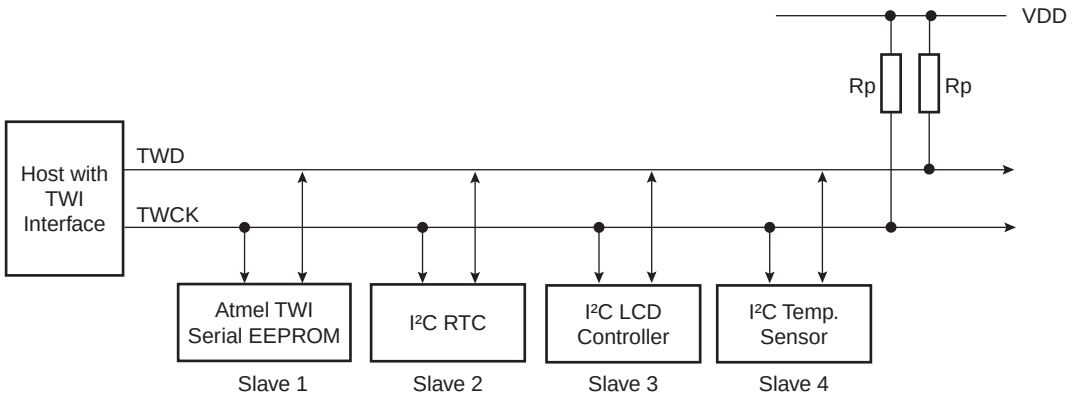
## 29.4 Block Diagram

**Figure 29-1. Block Diagram**



## 29.5 Application Block Diagram

**Figure 29-2. Application Block Diagram**




Rp: Pull up value as given by the I²C Standard

### 29.5.1 I/O Lines Description

**Table 29-3. I/O Lines Description**

| Pin Name | Pin Description | Type |
|---|---|---|
| TWD | Two-wire Serial Data | Input/Output |
| TWCK | Two-wire Serial Clock | Input/Output |

## 29.6 Product Dependencies

### 29.6.1 I/O Lines

Both TWD and TWCK are bidirectional lines, connected to a positive supply voltage via a current source or pull-up resistor (see Figure 29-2 on page 474). When the bus is free, both lines are high. The output stages of devices connected to the bus must have an open-drain or open-collector to perform the wired-AND function.

TWD and TWCK pins may be multiplexed with PIO lines. To enable the TWI, the programmer must perform the following step:

- Program the PIO controller to dedicate TWD and TWCK as peripheral lines.

The user must not program TWD and TWCK as open-drain. It is already done by the hardware.

**Table 29-4. I/O Lines**

| Instance | Signal | I/O Line | Peripheral |
|---|---|---|---|
| TWI0 | TWCK0 | PA4 | A |
| TWI0 | TWD0 | PA3 | A |
| TWI1 | TWCK1 | PB5 | A |
| TWI1 | TWD1 | PB4 | A |

### 29.6.2 Power Management

- Enable the peripheral clock.

The TWI interface may be clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the TWI clock.

### 29.6.3 Interrupt

The TWI interface has an interrupt line connected to the Nested Vector Interrupt Controller (NVIC). In order to handle interrupts, the NVIC must be programmed before configuring the TWI.

**Table 29-5. Peripheral IDs**

| Instance | ID |
|---|---|
| TWI0 | 19 |
| TWI1 | 20 |

## 29.7 Functional Description

### 29.7.1 Transfer Format

The data put on the TWD line must be 8 bits long. Data is transferred MSB first; each byte must be followed by an acknowledgement. The number of bytes per transfer is unlimited (see Figure 29-4).

Each transfer begins with a START condition and terminates with a STOP condition (see Figure 29-3).

- A high-to-low transition on the TWD line while TWCK is high defines the START condition.
- A low-to-high transition on the TWD line while TWCK is high defines a STOP condition.

**Figure 29-3. START and STOP Conditions**

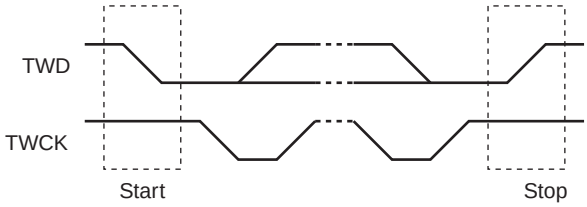
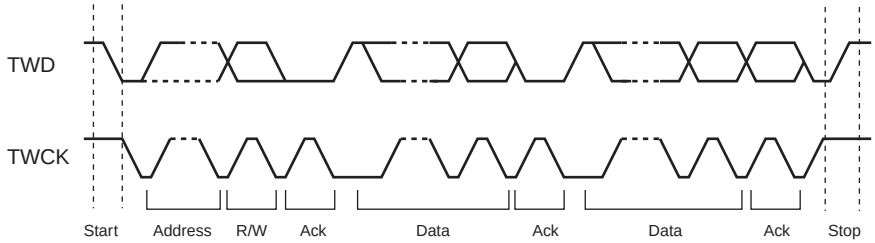



**Figure 29-4. Transfer Format**




### 29.7.2 Modes of Operation

The TWI has six modes of operations:

- Master transmitter mode
- Master receiver mode
- Multi-master transmitter mode
- Multi-master receiver mode
- Slave transmitter mode
- Slave receiver mode

These modes are described in the following chapters.

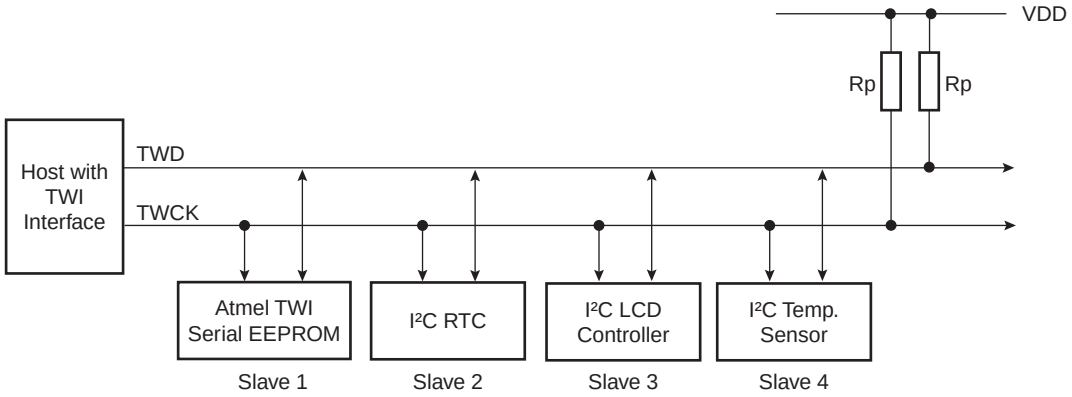
## 29.8 Master Mode

### 29.8.1 Definition

The Master is the device that starts a transfer, generates a clock and stops it.

### 29.8.2 Application Block Diagram

**Figure 29-5. Master Mode Typical Application Block Diagram**




Rp: Pull up value as given by the I²C Standard

### 29.8.3 Programming Master Mode

The following registers have to be programmed before entering Master mode:

1. DADR (+ IADRSZ + IADR if a 10 bit device is addressed): The device address is used to access slave devices in read or write mode.
2. CKDIV + CHDIV + CLDIV: Clock Waveform.
3. SVDIS: Disable the slave mode.
4. MSEN: Enable the master mode.

### 29.8.4 Master Transmitter Mode

After the master initiates a Start condition when writing into the Transmit Holding Register, TWI_THR, it sends a 7-bit slave address, configured in the Master Mode register (DADR in TWI_MMR), to notify the slave device. The bit following the slave address indicates the transfer direction, 0 in this case (MREAD = 0 in TWI_MMR).

The TWI transfers require the slave to acknowledge each received byte. During the acknowledge clock pulse (9th pulse), the master releases the data line (HIGH), enabling the slave to pull it down in order to generate the acknowledge. The master polls the data line during this clock pulse and sets the Not Acknowledge bit (**NACK)** in the status register if the slave does not acknowledge the byte. As with the other status bits, an interrupt can be generated if enabled in the interrupt enable register (TWI_IER). If the slave acknowledges the byte, the data written in the TWI_THR, is then shifted in the internal shifter and transferred. When an acknowledge is detected, the TXRDY bit is set until a new write in the TWI_THR.

While no new data is written in the TWI_THR, the Serial Clock Line is tied low. When new data is written in the TWI_THR, the SCL is released and the data is sent. To generate a STOP event, the STOP command must be performed by writing in the STOP field of TWI_CR.

After a Master Write transfer, the Serial Clock line is stretched (tied low) while no new data is written in the TWI_THR or until a STOP command is performed.

See Figure 29-6, Figure 29-7, and Figure 29-8.

**Figure 29-6. Master Write with One Data Byte**

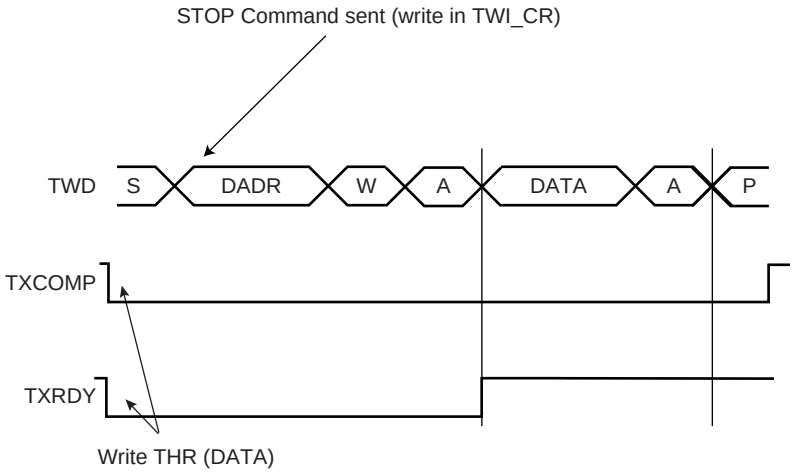



**Figure 29-7. Master Write with Multiple Data Bytes**

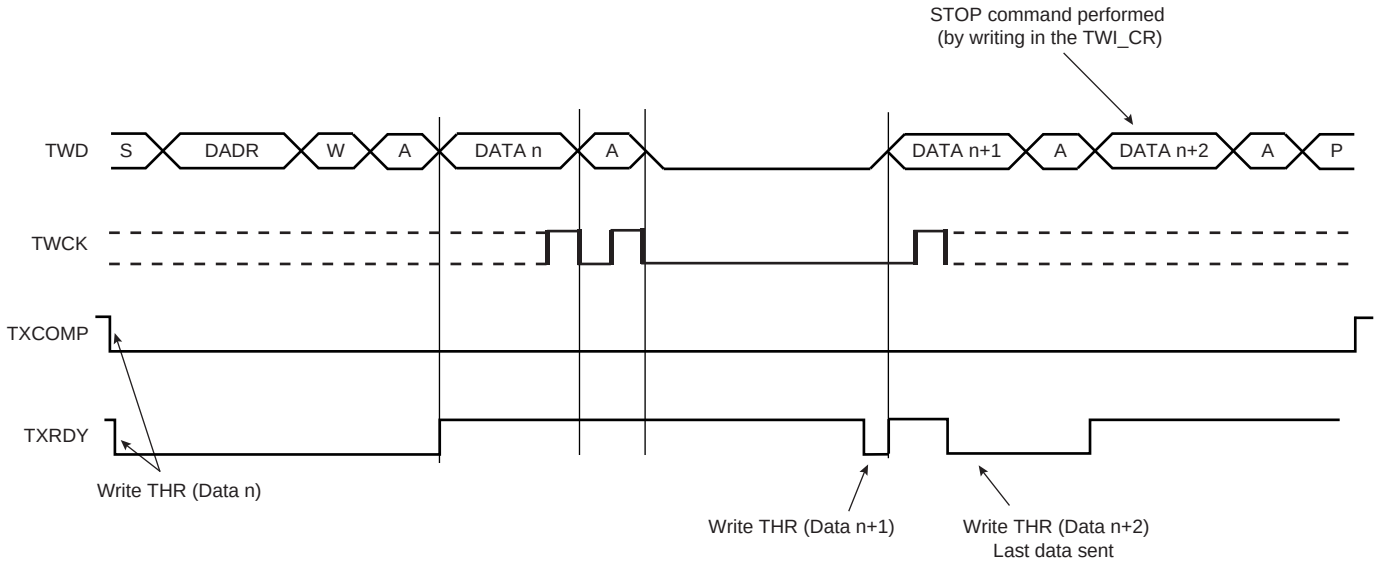
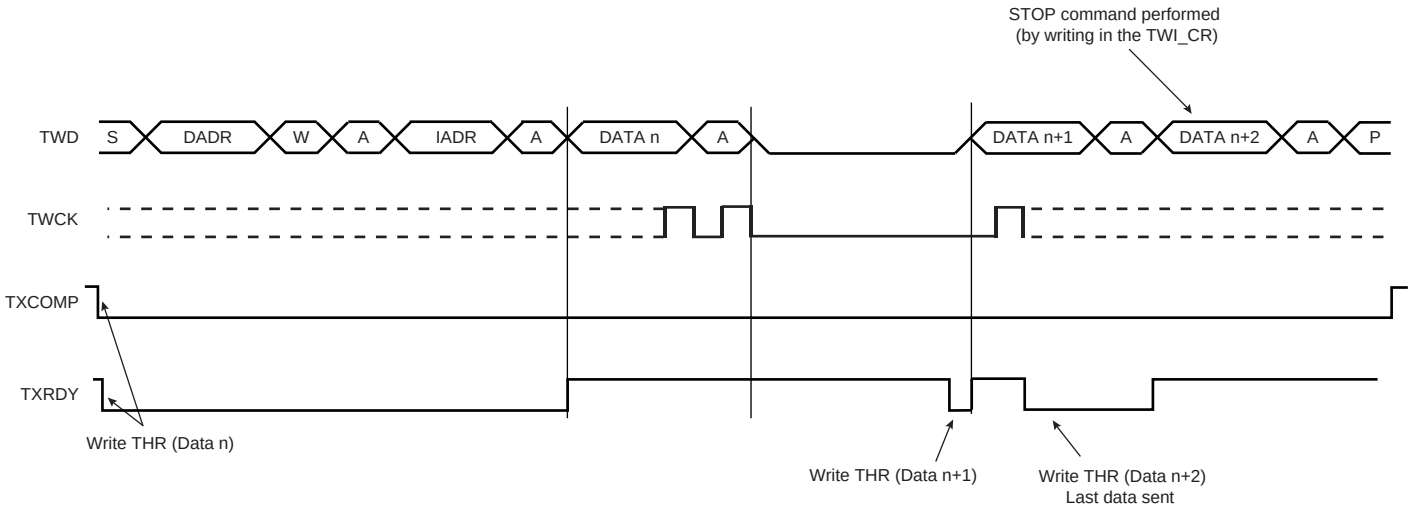


**Figure 29-8. Master Write with One Byte Internal Address and Multiple Data Bytes**




TXRDY is used as Transmit Ready for the PDC transmit channel.

### 29.8.5 Master Receiver Mode

The read sequence begins by setting the START bit. After the start condition has been sent, the master sends a 7-bit slave address to notify the slave device. The bit following the slave address indicates the transfer direction, 1 in this case (MREAD = 1 in TWI_MMR). During the acknowledge clock pulse (9th pulse), the master releases the data line (HIGH), enabling the slave to pull it down in order to generate the acknowledge. The master polls the data line during this clock pulse and sets the **NACK** bit in the status register if the slave does not acknowledge the byte.

If an acknowledge is received, the master is then ready to receive data from the slave. After data has been received, the master sends an acknowledge condition to notify the slave that the data has been received except for the last data, after the stop condition. See Figure 29-9. When the RXRDY bit is set in the status register, a character has been received in the receive-holding register (TWI_RHR). The RXRDY bit is reset when reading the TWI_RHR.

When a single data byte read is performed, with or without internal address **(IADR**), the START and STOP bits must be set at the same time. See Figure 29-9. When a multiple data byte read is performed, with or without internal address **(IADR**), the STOP bit must be set after the next-to-last data received. See Figure 29-10. For Internal Address usage see Section 29.8.6.

**Figure 29-9. Master Read with One Data Byte**

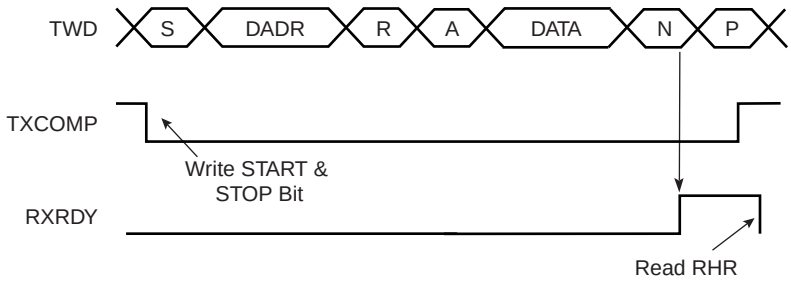
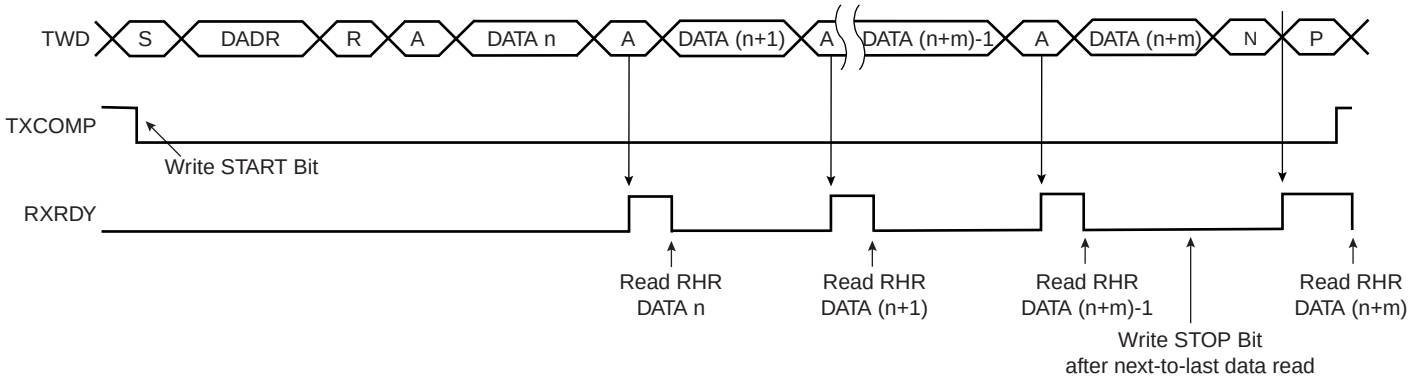


**Figure 29-10. Master Read with Multiple Data Bytes**




RXRDY is used as Receive Ready for the PDC receive channel.

### 29.8.6 Internal Address

The TWI interface can perform various transfer formats: Transfers with 7-bit slave address devices and 10-bit slave address devices.

#### 29.8.6.1 7-bit Slave Addressing

When Addressing 7-bit slave devices, the internal address bytes are used to perform random address (read or write) accesses to reach one or more data bytes, within a memory page location in a serial memory, for example. When performing read operations with an internal address, the TWI performs a write operation to set the internal address into the slave device, and then switch to Master Receiver mode. Note that the second start condition (after sending the IADR) is sometimes called “repeated start” (Sr) in I2C fully-compatible devices. See Figure 29-12. See Figure 29-11 and Figure 29-13 for Master Write operation with internal address.

The three internal address bytes are configurable through the Master Mode register (TWI_MMR).

If the slave device supports only a 7-bit address, i.e. no internal address, **IADRSZ** must be set to 0.

In the figures below the following abbreviations are used:

- S Start
- Sr Repeated Start
- P Stop
- W Write
- R Read
- A Acknowledge
- N Not Acknowledge
- DADR Device Address
- IADR Internal Address

**Figure 29-11. Master Write with One, Two or Three Bytes Internal Address and One Data Byte**

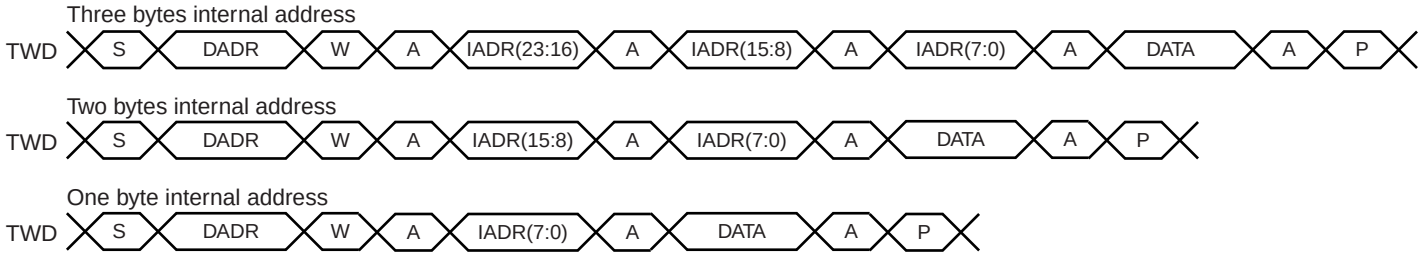
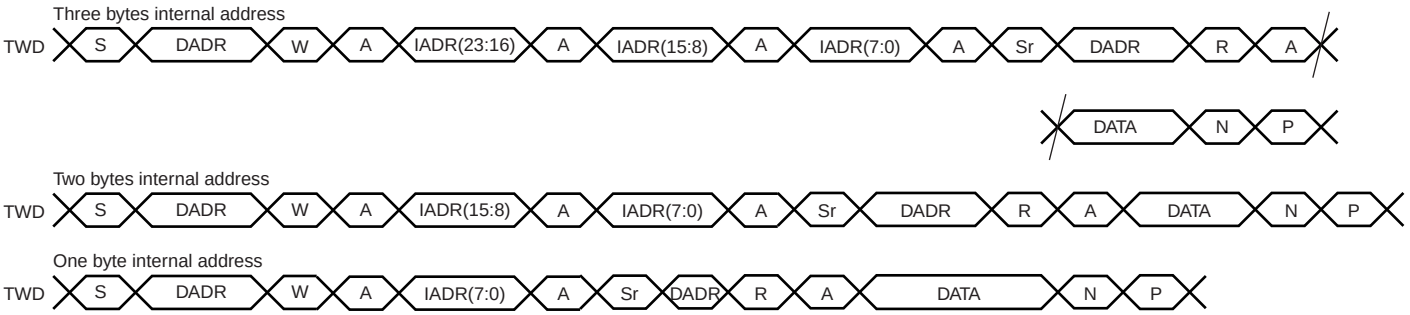



**Figure 29-12. Master Read with One, Two or Three Bytes Internal Address and One Data Byte**




### 29.8.6.2 10-bit Slave Addressing

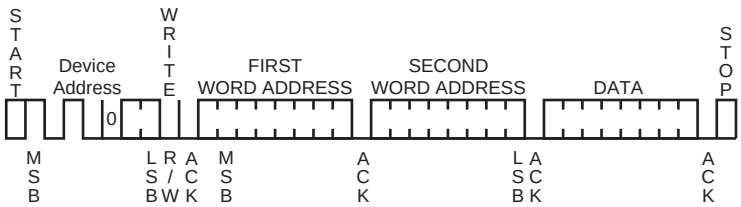
For a slave address higher than 7 bits, the user must configure the address size **(IADRSZ**) and set the other slave address bits in the internal address register (TWI_IADR). The two remaining Internal address bytes, IADR[15:8] and IADR[23:16] can be used the same as in 7-bit Slave Addressing.

**Example:** Address a 10-bit device (10-bit device address is b1 b2 b3 b4 b5 b6 b7 b8 b9 b10)

1. Program IADRSZ = 1,
2. Program DADR with 1 1 1 1 0 b1 b2 (b1 is the MSB of the 10-bit address, b2, etc.)
3. Program TWI_IADR with b3 b4 b5 b6 b7 b8 b9 b10 (b10 is the LSB of the 10-bit address)

Figure 29-13 below shows a byte write to an Atmel AT24LC512 EEPROM. This demonstrates the use of internal addresses to access the device.

**Figure 29-13. Internal Address Usage**



### 29.8.7 Using the Peripheral DMA Controller (PDC)

The use of the PDC significantly reduces the CPU load.

To assure correct implementation, respect the following programming sequences:

#### 29.8.7.1 Data Transmit with the PDC

1. Initialize the transmit PDC (memory pointers, size, etc.).
2. Configure the master mode (DADR, CKDIV, etc.).
3. Start the transfer by setting the PDC TXTEN bit.
4. Wait for the PDC end TX flag.
5. Disable the PDC by setting the PDC TXDIS bit.

#### 29.8.7.2 Data Receive with the PDC

1. Initialize the receive PDC (memory pointers, size - 1, etc.).
2. Configure the master mode (DADR, CKDIV, etc.).
3. Start the transfer by setting the PDC RXTEN bit.
4. Wait for the PDC end RX flag.
5. Disable the PDC by setting the PDC RXDIS bit.

### 29.8.8 Using the DMA Controller (DMAC)

The use of the DMAC significantlly reduces the CPU load.

To assure correct implementation, respect the following programming sequence.

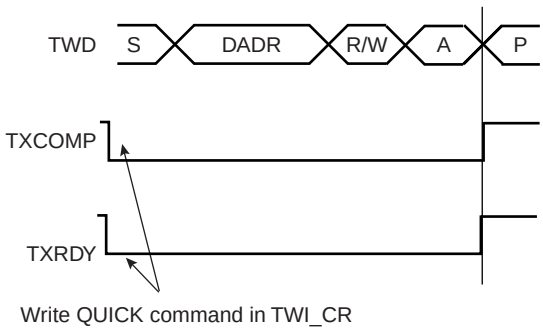
1. Initialize the DMAC (channels, memory pointers , size, etc.);
1. Configure the master mode (DADR, CKDIV, etc.).
1. Enable the DMAC.
1. Wait for the DMAC flag.
1. Disable the DMAC.

### 29.8.9 SMBUS Quick Command (Master Mode Only)

The TWI interface can perform a Quick Command:

1. Configure the master mode (DADR, CKDIV, etc.).
2. Write the MREAD bit in the TWI_MMR register at the value of the one-bit command to be sent.
3. Start the transfer by setting the QUICK bit in the TWI_CR.

**Figure 29-14. SMBUS Quick Command**



### 29.8.10 Read-write Flowcharts

The following flowcharts shown in Figure 29-16 on page 484, Figure 29-17 on page 485, Figure 29-18 on page 486, Figure 29-19 on page 487 and Figure 29-20 on page 488 give examples for read and write operations. A polling or interrupt method can be used to check the status bits. The interrupt method requires that the interrupt enable register (TWI_IER) be configured first.

**Figure 29-15. TWI Write Operation with Single Data Byte without Internal Address**

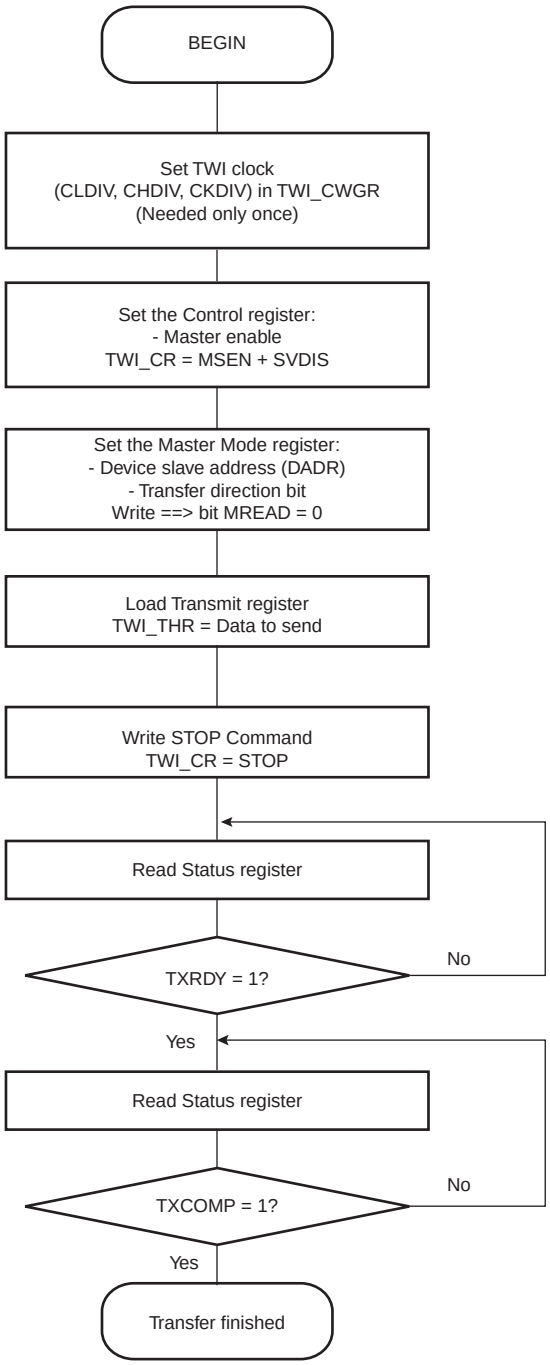


**Figure 29-16. TWI Write Operation with Single Data Byte and Internal Address**

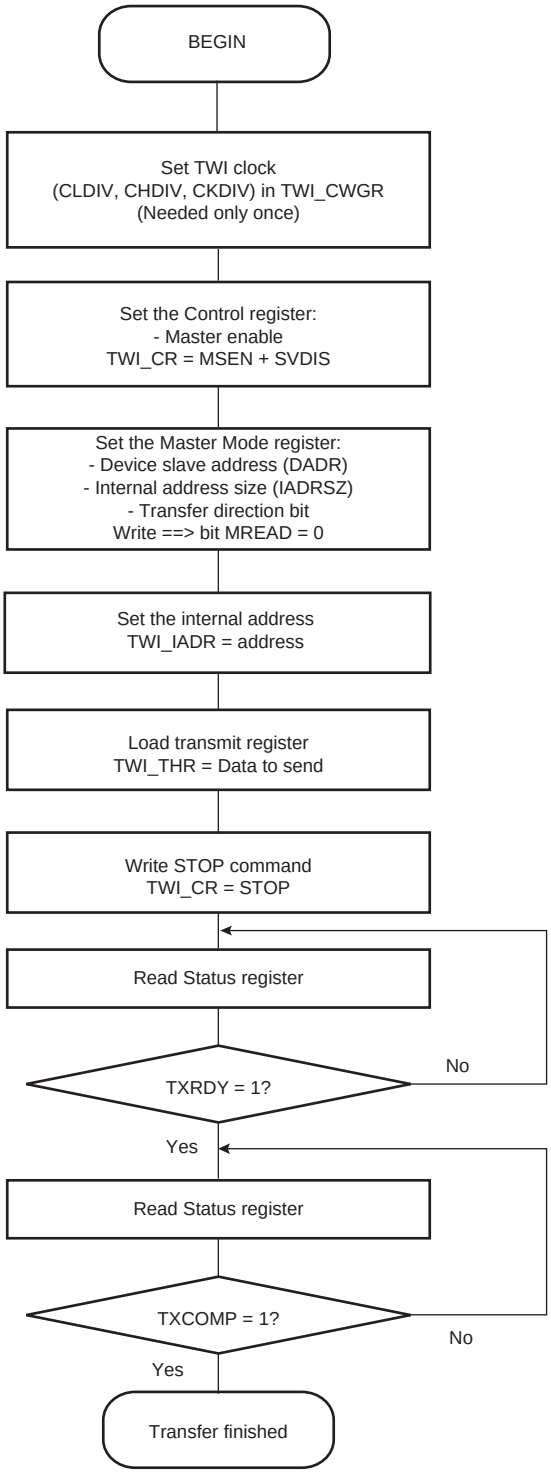


**Figure 29-17. TWI Write Operation with Multiple Data Bytes with or without Internal Address**

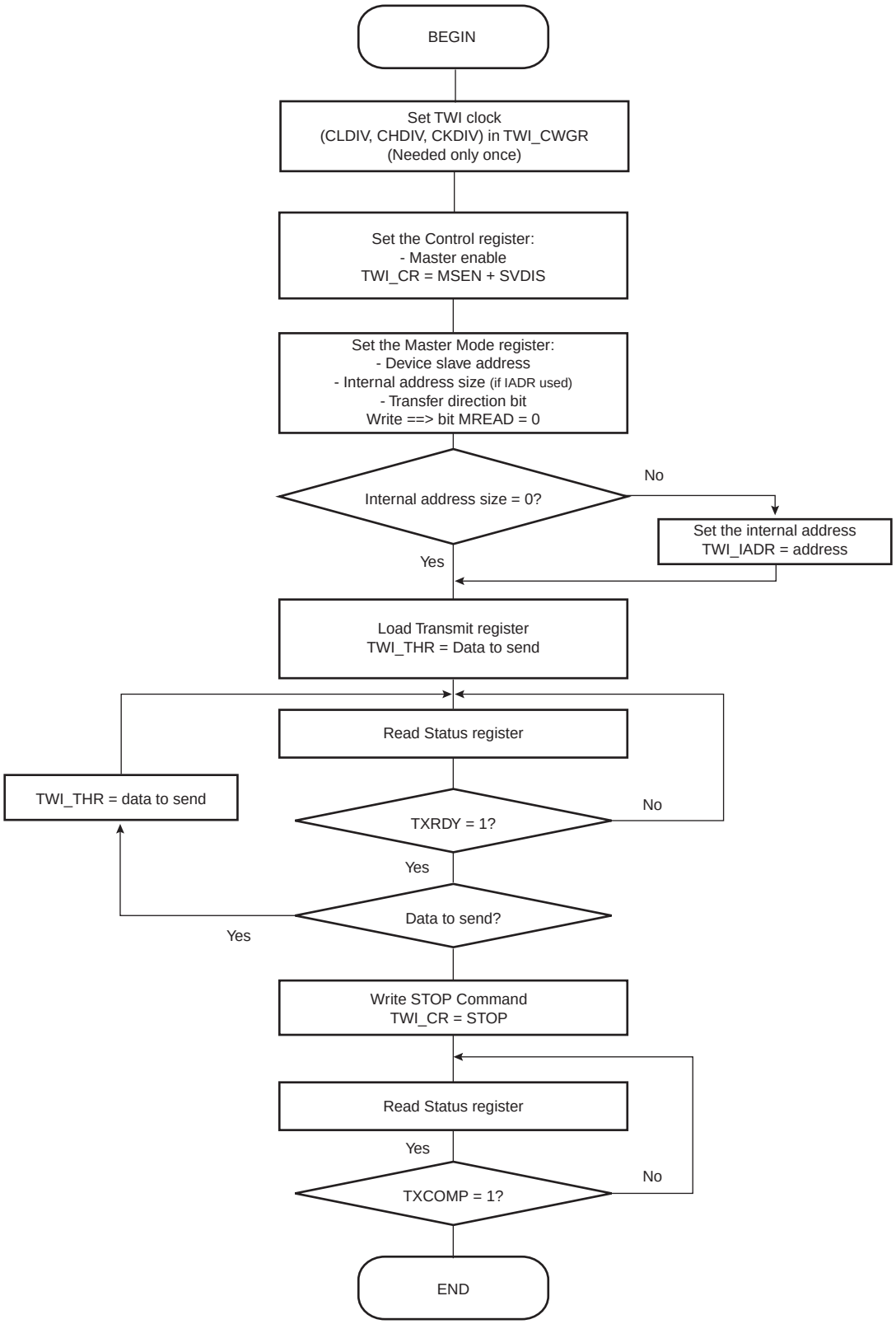


**Figure 29-18. TWI Read Operation with Single Data Byte without Internal Address**

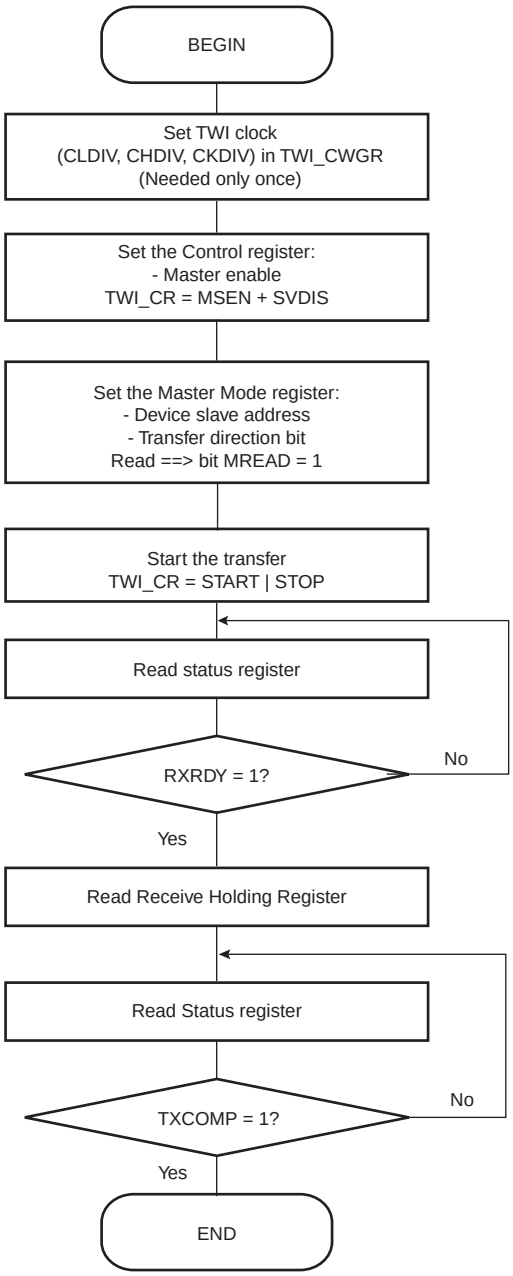


**Figure 29-19. TWI Read Operation with Single Data Byte and Internal Address**

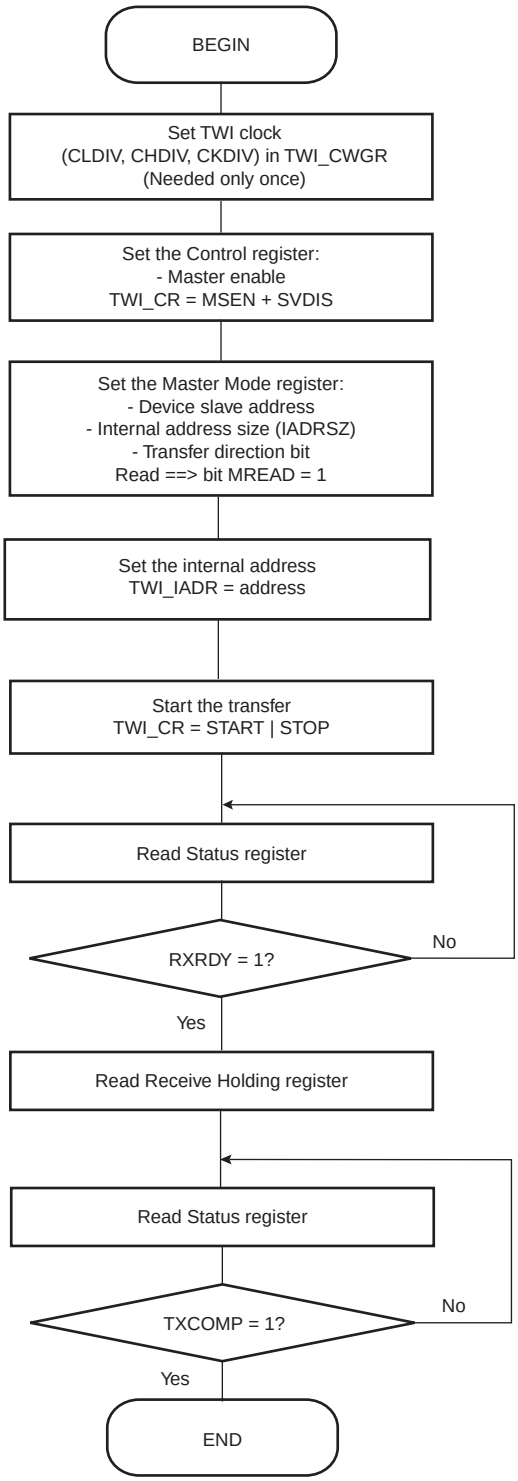
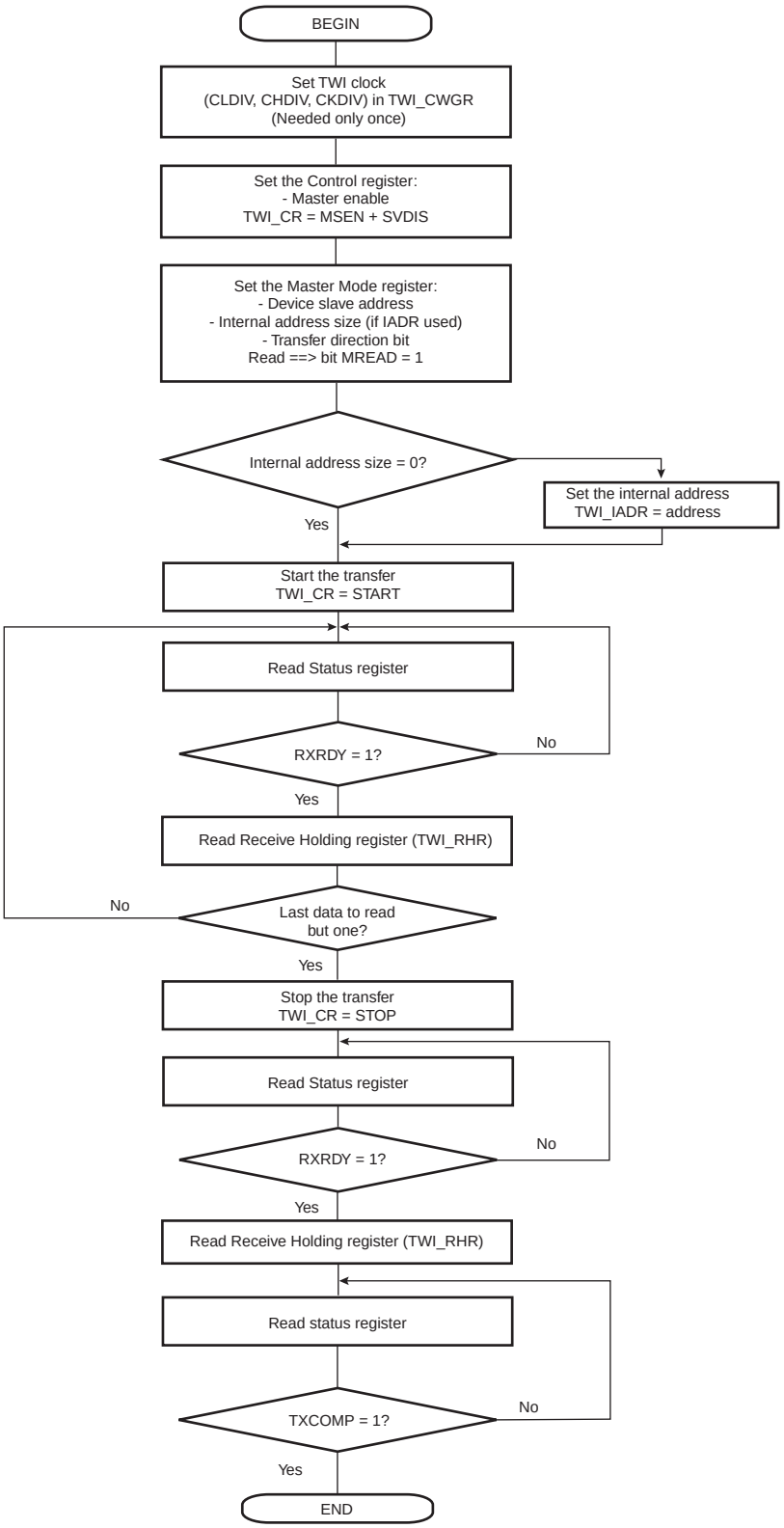


**Figure 29-20. TWI Read Operation with Multiple Data Bytes with or without Internal Address**



## 29.9 Multi-master Mode

### 29.9.1 Definition

More than one master may handle the bus at the same time without data corruption by using arbitration.

Arbitration starts as soon as two or more masters place information on the bus at the same time, and stops (arbitration is lost) for the master that intends to send a logical one while the other master sends a logical zero.

As soon as arbitration is lost by a master, it stops sending data and listens to the bus in order to detect a stop. When the stop is detected, the master who has lost arbitration may put its data on the bus by respecting arbitration.

Arbitration is illustrated in Figure 29-22 on page 490.

### 29.9.2 Different Multi-master Modes

Two multi-master modes may be distinguished:

1. TWI is considered as a Master only and will never be addressed.
2. TWI may be either a Master or a Slave and may be addressed.

Note: In both Multi-master modes arbitration is supported.

#### 29.9.2.1 TWI as Master Only

In this mode, TWI is considered as a Master only (MSEN is always at one) and must be driven like a Master with the ARBLST (ARBitration Lost) flag in addition.

If arbitration is lost (ARBLST = 1), the programmer must reinitiate the data transfer.

If the user starts a transfer (ex.: DADR + START + W + Write in THR) and if the bus is busy, the TWI automatically waits for a STOP condition on the bus to initiate the transfer (see Figure 29-21 on page 490).

Note: The state of the bus (busy or free) is not indicated in the user interface.

#### 29.9.2.2 TWI as Master or Slave

The automatic reversal from Master to Slave is not supported in case of a lost arbitration.

Then, in the case where TWI may be either a Master or a Slave, the programmer must manage the pseudo Multi-master mode described in the steps below.

1. Program TWI in Slave mode (SADR + MSDIS + SVEN) and perform Slave Access (if TWI is addressed).
2. If TWI has to be set in Master mode, wait until TXCOMP flag is at 1.
3. Program Master mode (DADR + SVDIS + MSEN) and start the transfer (ex: START + Write in THR).
4. As soon as the Master mode is enabled, TWI scans the bus in order to detect if it is busy or free. When the bus is considered as free, TWI initiates the transfer.
5. As soon as the transfer is initiated and until a STOP condition is sent, the arbitration becomes relevant and the user must monitor the ARBLST flag.
6. If the arbitration is lost (ARBLST is set to 1), the user must program the TWI in Slave mode in the case where the Master that won the arbitration wanted to access the TWI.
7. If TWI has to be set in Slave mode, wait until TXCOMP flag is at 1 and then program the Slave mode.

Note: In the case where the arbitration is lost and TWI is addressed, TWI will not acknowledge even if it is programmed in Slave mode as soon as ARBLST is set to 1. Then, the Master must repeat SADR.

**Figure 29-21. Programmer Sends Data While the Bus is Busy**

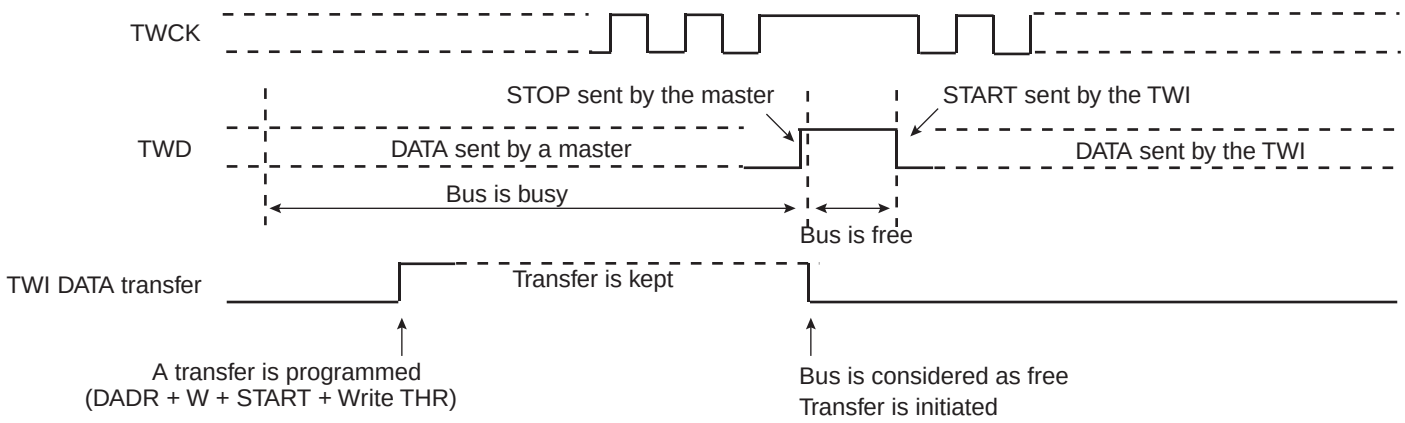
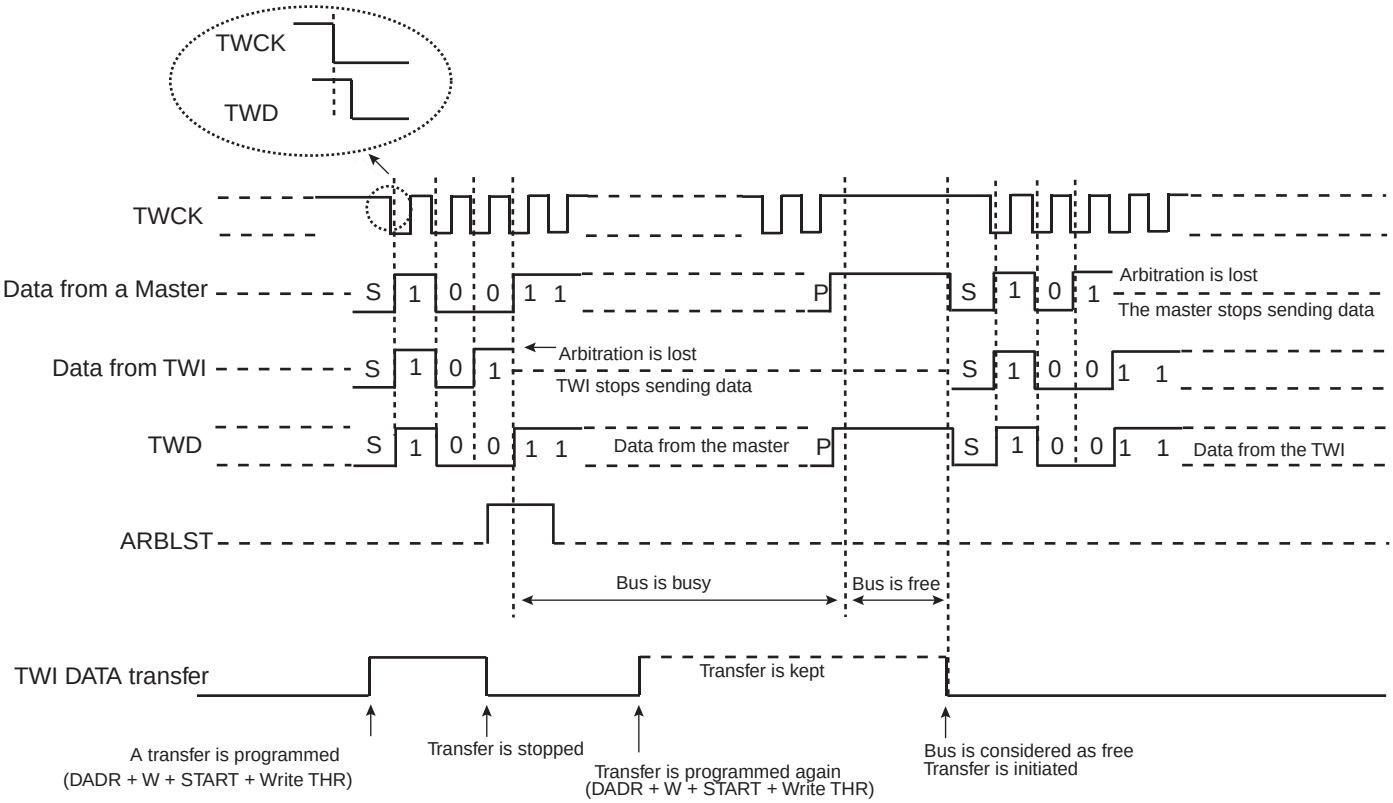
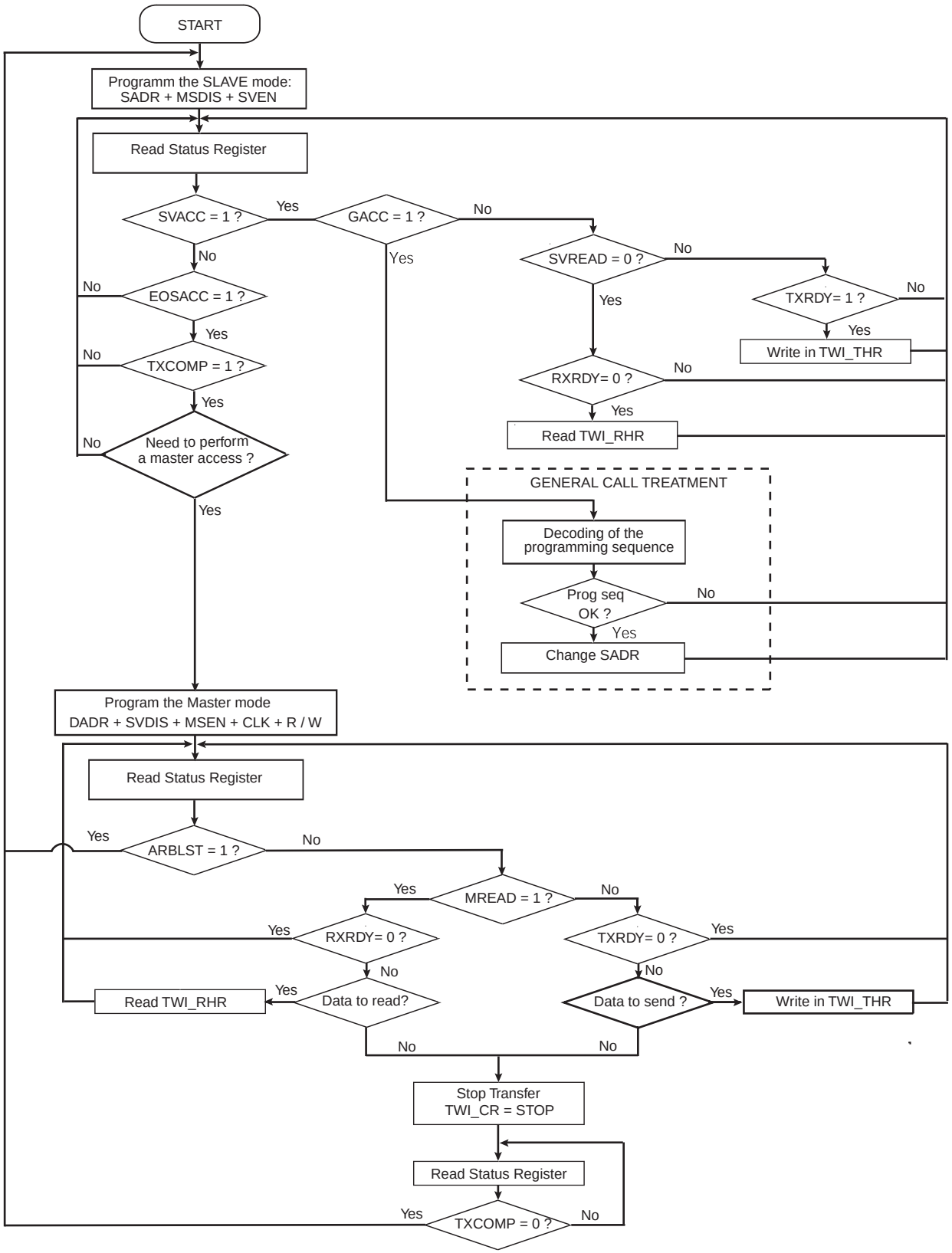



**Figure 29-22. Arbitration Cases**




The flowchart shown in Figure 29-23 on page 491 gives an example of read and write operations in Multi-master mode.

**Figure 29-23. Multi-master Flowchart**



## 29.10 Slave Mode

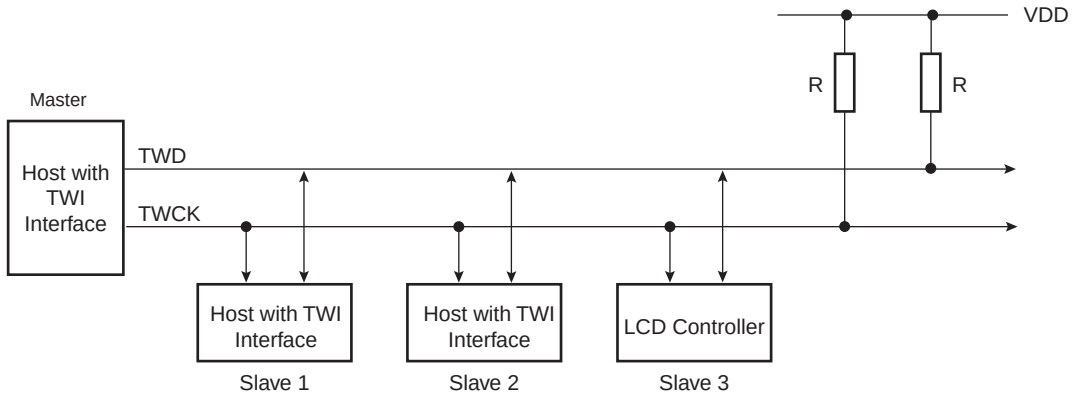
### 29.10.1 Definition

The Slave Mode is defined as a mode where the device receives the clock and the address from another device called the master.

In this mode, the device never initiates and never completes the transmission (START, REPEATED_START and STOP conditions are always provided by the master).

### 29.10.2 Application Block Diagram

**Figure 29-24. Slave Mode Typical Application Block Diagram**




### 29.10.3 Programming Slave Mode

The following fields must be programmed before entering Slave mode:

1. SADR (TWI_SMR): The slave device address is used in order to be accessed by master devices in read or write mode.
2. MSDIS (TWI_CR): Disable the master mode.
3. SVEN (TWI_CR): Enable the slave mode.

As the device receives the clock, values written in TWI_CWGR are not taken into account.

### 29.10.4 Receiving Data

After a Start or Repeated Start condition is detected and if the address sent by the Master matches with the Slave address programmed in the SADR (Slave ADdress) field, SVACC (Slave ACCess) flag is set and SVREAD (Slave READ) indicates the direction of the transfer.

SVACC remains high until a STOP condition or a repeated START is detected. When such a condition is detected, EOSACC (End Of Slave ACCess) flag is set.

#### 29.10.4.1 Read Sequence

In the case of a Read sequence (SVREAD is high), TWI transfers data written in the TWI_THR (TWI Transmit Holding Register) until a STOP condition or a REPEATED_START + an address different from SADR is detected. Note that at the end of the read sequence TXCOMP (Transmission Complete) flag is set and SVACC reset.

As soon as data is written in the TWI_THR, TXRDY (Transmit Holding Register Ready) flag is reset, and it is set when the shift register is empty and the sent data acknowledged or not. If the data is not acknowledged, the NACK flag is set.

Note that a STOP or a repeated START always follows a NACK.

See Figure 29-25 on page 494.

#### 29.10.4.2 Write Sequence

In the case of a Write sequence (SVREAD is low), the RXRDY (Receive Holding Register Ready) flag is set as soon as a character has been received in the TWI_RHR (TWI Receive Holding Register). RXRDY is reset when reading the TWI_RHR.

TWI continues receiving data until a STOP condition or a REPEATED_START + an address different from SADR is detected. Note that at the end of the write sequence TXCOMP flag is set and SVACC reset.

See Figure 29-26 on page 494.

#### 29.10.4.3 Clock Synchronization Sequence

In the case where TWI_THR or TWI_RHR is not written/read in time, TWI performs a clock synchronization.

Clock stretching information is given by the SCLWS (Clock Wait state) bit.

See Figure 29-28 on page 495 and Figure 29-29 on page 496.

#### 29.10.4.4 General Call

In the case where a GENERAL CALL is performed, GACC (General Call ACCess) flag is set.

After GACC is set, it is up to the programmer to interpret the meaning of the GENERAL CALL and to decode the new address programming sequence.

See Figure 29-27 on page 495.

#### 29.10.4.5 PDC

As it is impossible to know the exact number of data to receive/send, the use of PDC is NOT recommended in SLAVE mode.

#### 29.10.4.6 DMAC

As it is impossible to know the exact number of data to receive/send, the use of DMAC is NOT recommended in SLAVE mode.

### 29.10.5 Data Transfer

#### 29.10.5.1 Read Operation

The read mode is defined as a data requirement from the master.

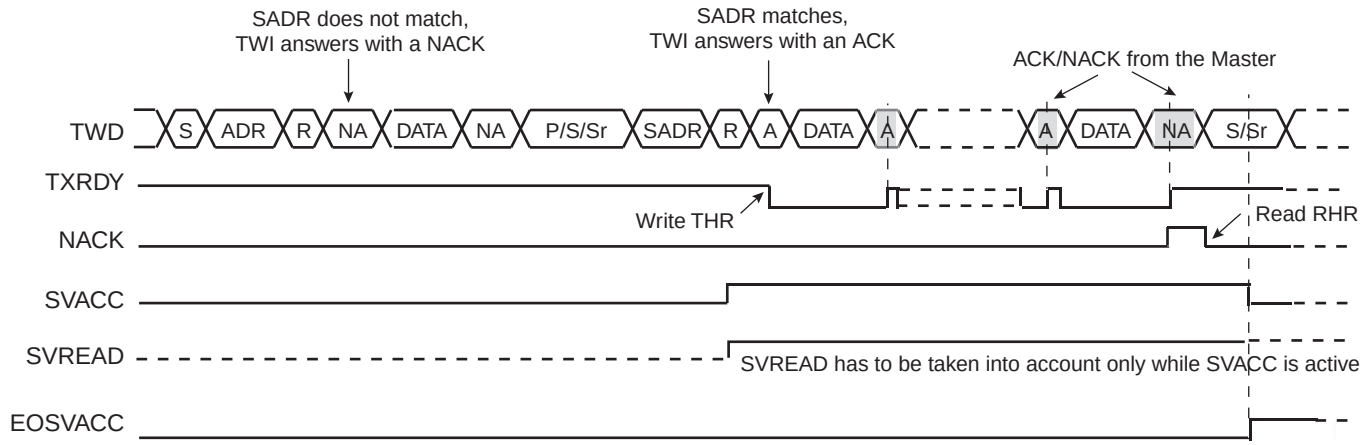
After a START or a REPEATED START condition is detected, the decoding of the address starts. If the slave address (SADR) is decoded, SVACC is set and SVREAD indicates the direction of the transfer.

Until a STOP or REPEATED START condition is detected, TWI continues sending data loaded in the TWI_THR register.

If a STOP condition or a REPEATED START + an address different from SADR is detected, SVACC is reset.

Figure 29-25 on page 494 describes the write operation.

**Figure 29-25. Read Access Ordered by a MASTER**




Notes:
1. When SVACC is low, the state of SVREAD becomes irrelevant.
2. TXRDY is reset when data has been transmitted from TWI_THR to the shift register and set when this data has been acknowledged or non acknowledged.

### 29.10.5.2 Write Operation

The write mode is defined as a data transmission from the master.

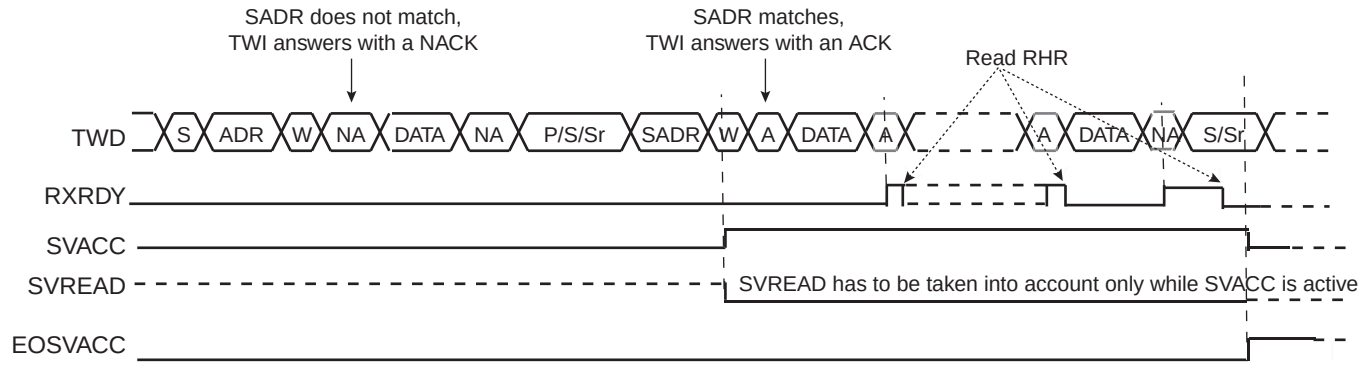
After a START or a REPEATED START, the decoding of the address starts. If the slave address is decoded, SVACC is set and SVREAD indicates the direction of the transfer (SVREAD is low in this case).

Until a STOP or REPEATED START condition is detected, TWI stores the received data in the TWI_RHR register.

If a STOP condition or a REPEATED START + an address different from SADR is detected, SVACC is reset.

Figure 29-26 on page 494 describes the Write operation.

**Figure 29-26. Write Access Ordered by a Master**




Notes:
1. When SVACC is low, the state of SVREAD becomes irrelevant.
2. RXRDY is set when data has been transmitted from the shift register to the TWI_RHR and reset when this data is read.

### 29.10.5.3 General Call

The general call is performed in order to change the address of the slave.

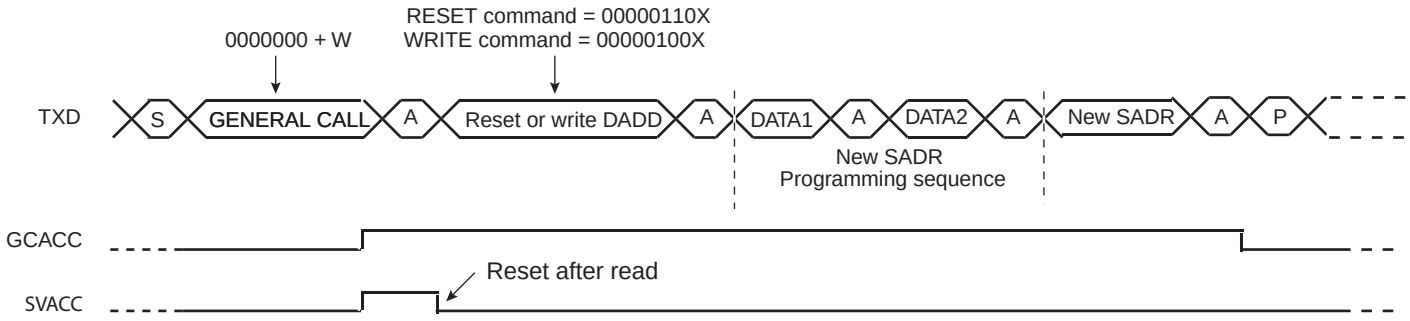
If a GENERAL CALL is detected, GACC is set.

After the detection of General Call, it is up to the programmer to decode the commands which come afterwards.

In case of a WRITE command, the programmer has to decode the programming sequence and program a new SADR if the programming sequence matches.

Figure 29-27 on page 495 describes the General Call access.

**Figure 29-27. Master Performs a General Call**




Note: This method allows the user to create an own programming sequence by choosing the programming bytes and the number of them. The programming sequence has to be provided to the master.

### 29.10.5.4 Clock Synchronization

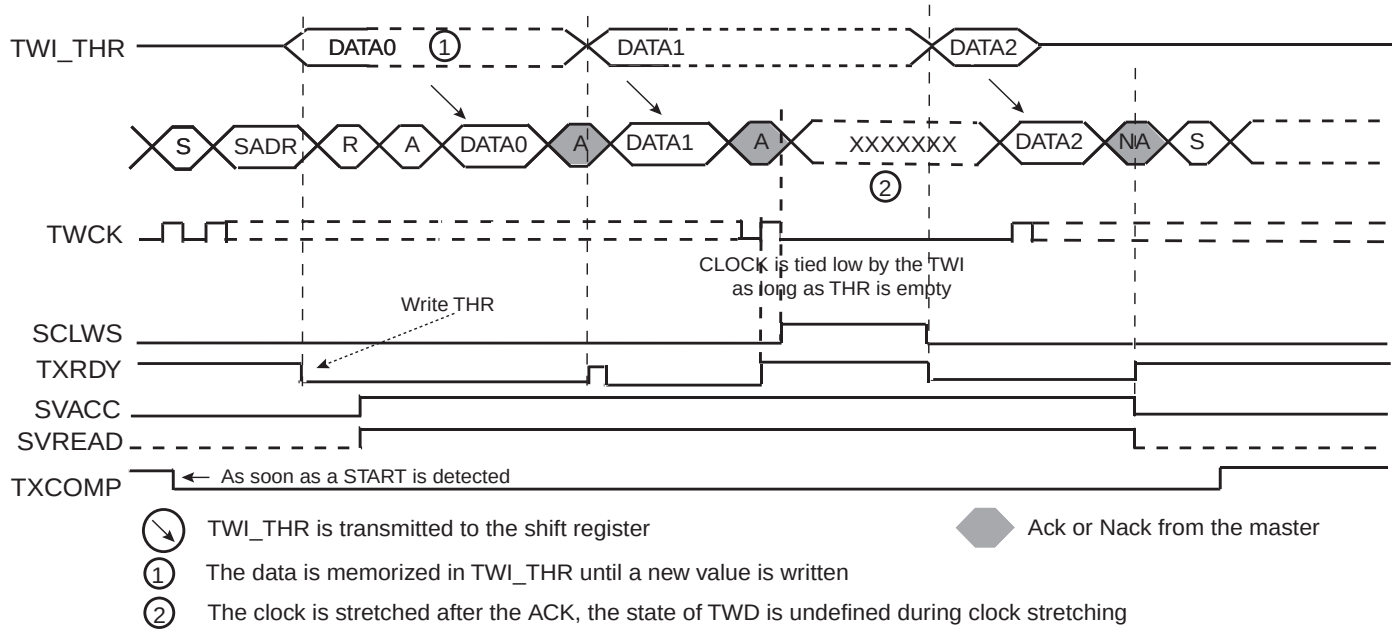
In both read and write modes, it may happen that TWI_THR/TWI_RHR buffer is not filled /emptied before the emission/reception of a new character. In this case, to avoid sending/receiving undesired data, a clock stretching mechanism is implemented.

*Clock Synchronization in Read Mode*

The clock is tied low if the shift register is empty and if a STOP or REPEATED START condition was not detected. It is tied low until the shift register is loaded.

Figure 29-28 on page 495 describes the clock synchronization in Read mode.

**Figure 29-28. Clock Synchronization in Read Mode**



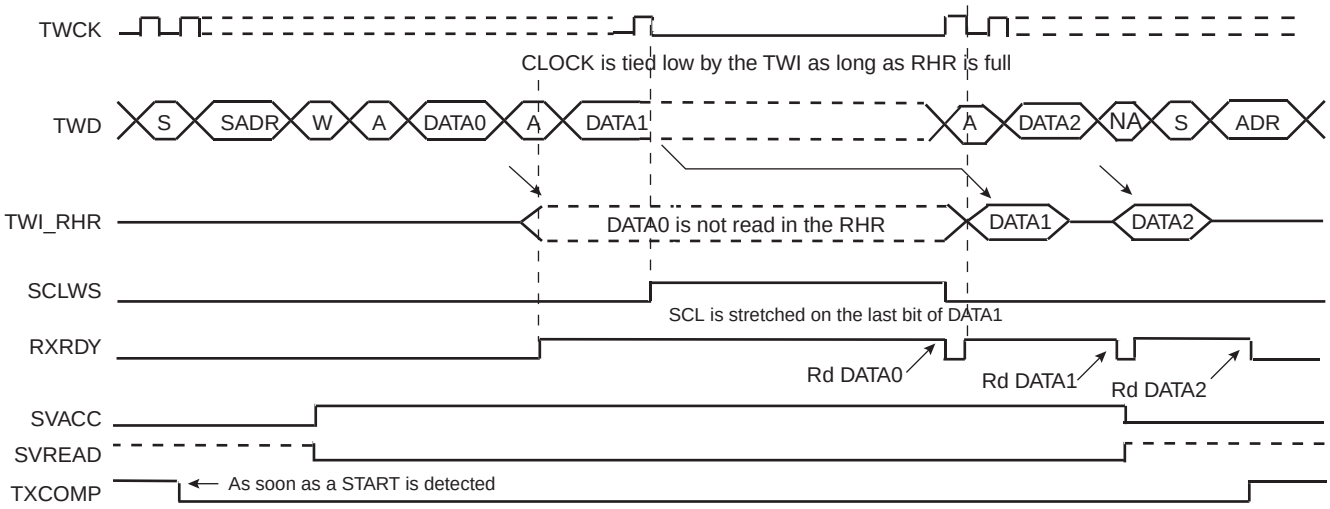

Notes:
1. TXRDY is reset when data has been written in the TWI_THR to the shift register and set when this data has been acknowledged or non acknowledged.
2. At the end of the read sequence, TXCOMP is set after a STOP or after a REPEATED_START + an address different from SADR.
3. SCLWS is automatically set when the clock synchronization mechanism is started.

*Clock Synchronization in Write Mode*

The clock is tied low if the shift register and the TWI_RHR is full. If a STOP or REPEATED_START condition was not detected, it is tied low until TWI_RHR is read.

Figure 29-29 on page 496 describes the clock synchronization in Read mode.

**Figure 29-29. Clock Synchronization in Write Mode**




Notes:
1. At the end of the read sequence, TXCOMP is set after a STOP or after a REPEATED_START + an address different from SADR.
2. SCLWS is automatically set when the clock synchronization mechanism is started and automatically reset when the mechanism is finished.

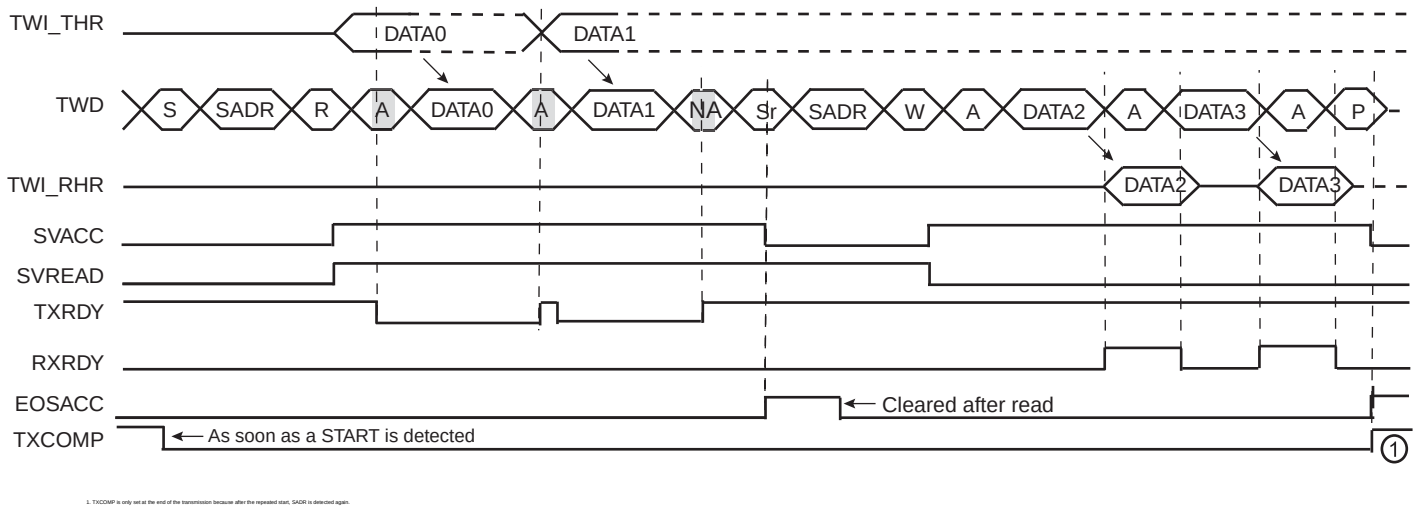
### 29.10.5.5 Reversal after a Repeated Start

*Reversal of Read to Write*

The master initiates the communication by a read command and finishes it by a write command.

Figure 29-30 on page 497 describes the repeated start + reversal from Read to Write mode.

**Figure 29-30. Repeated Start + Reversal from Read to Write Mode**

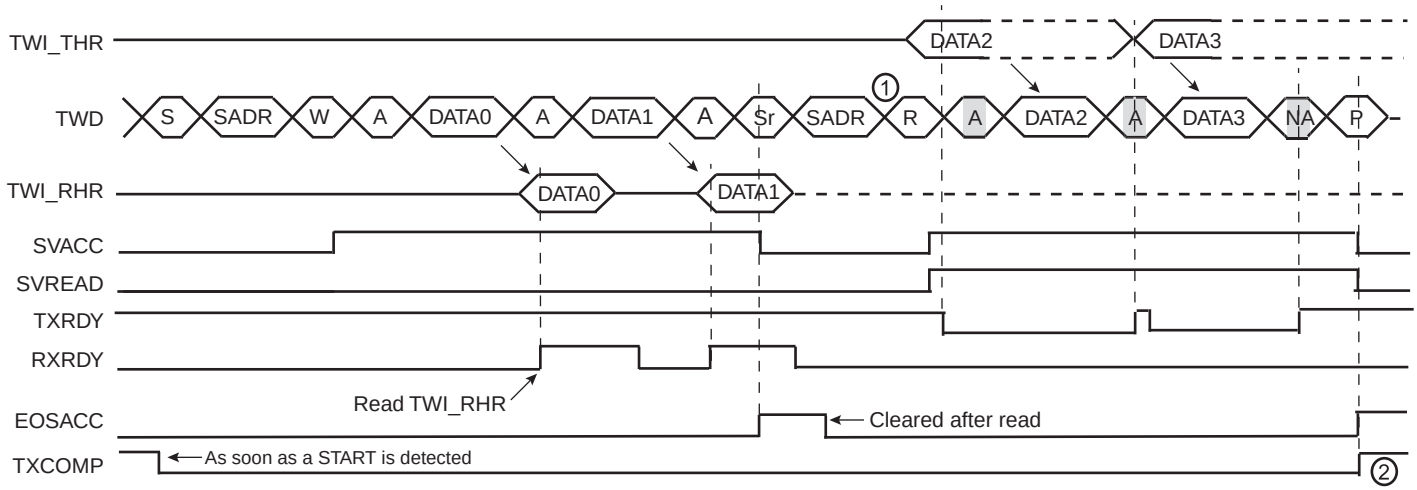



1. TXCOMP is only set at the end of the transmission because after the repeated start, SADR is detected again.

*Reversal of Write to Read*

The master initiates the communication by a write command and finishes it by a read command.Figure 29-31 on page 497 describes the repeated start + reversal from Write to Read mode.

**Figure 29-31. Repeated Start + Reversal from Write to Read Mode**

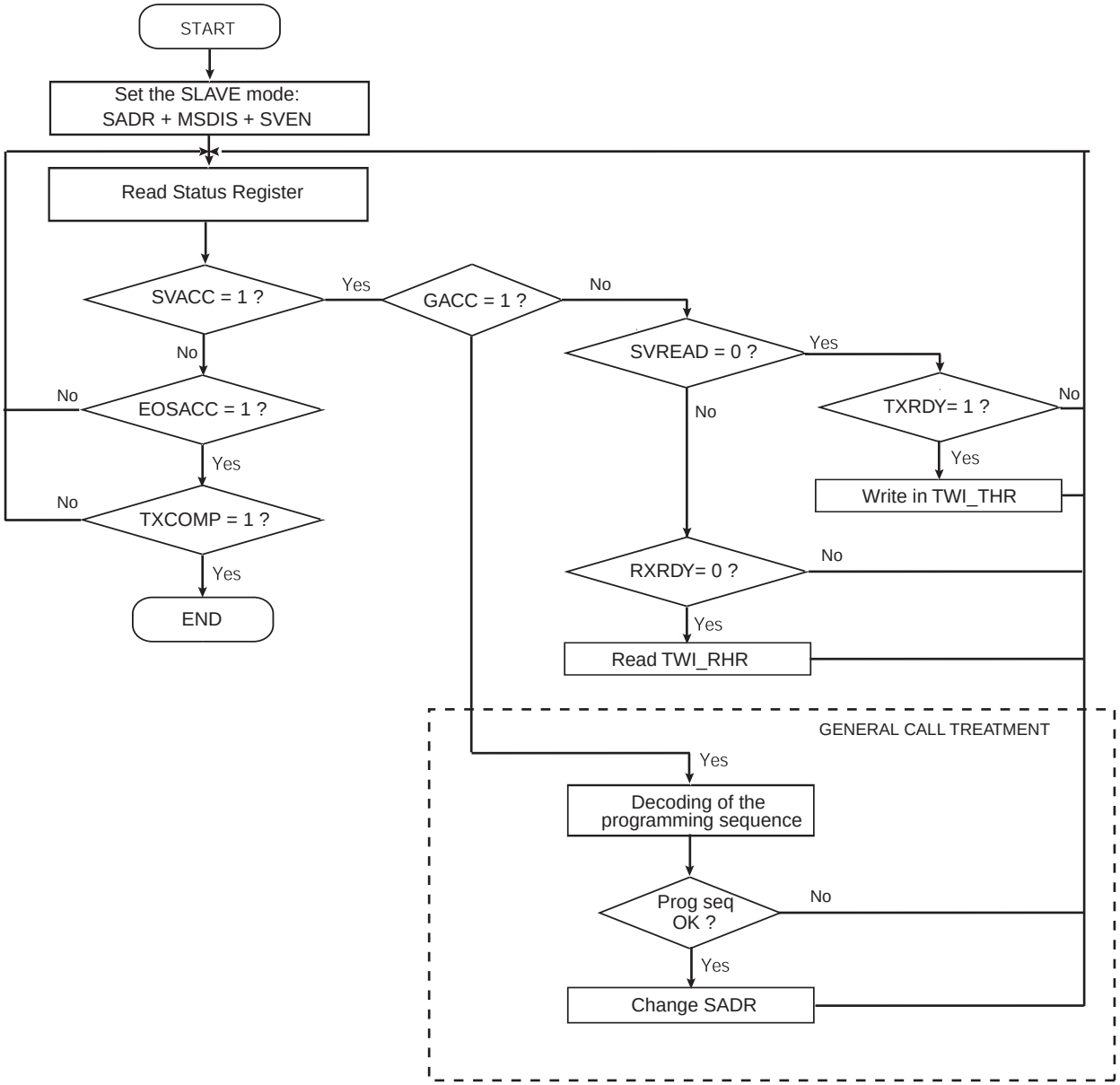



Notes:
1. In this case, if TWI_THR has not been written at the end of the read command, the clock is automatically stretched before the ACK.
2. TXCOMP is only set at the end of the transmission because after the repeated start, SADR is detected again.

### 29.10.6 Read Write Flowcharts

The flowchart shown in Figure 29-32 on page 498 gives an example of read and write operations in Slave mode. A polling or interrupt method can be used to check the status bits. The interrupt method requires that the interrupt enable register (TWI_IER) be configured first.

**Figure 29-32. Read Write Flowchart in Slave Mode**



## 29.11 Two-wire Interface (TWI) User Interface

**Table 29-6. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x00 | Control Register | TWI_CR | Write-only | N / A |
| 0x04 | Master Mode Register | TWI_MMR | Read-write | 0x00000000 |
| 0x08 | Slave Mode Register | TWI_SMR | Read-write | 0x00000000 |
| 0x0C | Internal Address Register | TWI_IADR | Read-write | 0x00000000 |
| 0x10 | Clock Waveform Generator Register | TWI_CWGR | Read-write | 0x00000000 |
| 0x14 - 0x1C | Reserved | – | – | – |
| 0x20 | Status Register | TWI_SR | Read-only | 0x0000F009 |
| 0x24 | Interrupt Enable Register | TWI_IER | Write-only | N / A |
| 0x28 | Interrupt Disable Register | TWI_IDR | Write-only | N / A |
| 0x2C | Interrupt Mask Register | TWI_IMR | Read-only | 0x00000000 |
| 0x30 | Receive Holding Register | TWI_RHR | Read-only | 0x00000000 |
| 0x34 | Transmit Holding Register | TWI_THR | Write-only | 0x00000000 |
| 0xEC - 0xFC(1) | Reserved | – | – | – |
| 0x100 - 0x124 | Reserved for the PDC | – | – | – |

Note: 1. All unlisted offset values are conisedered as “reserved”.

### 29.11.1 TWI Control Register

Name: TWI_CR

**Addresses:** 0x40018000 (0), 0x4001C000 (1)

Access: Write-only

**Reset:** 0x00000000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SWRST | QUICK | SVDIS | SVEN | MSDIS | MSEN | STOP | START |

- **START: Send a START Condition**

0 = No effect.

1 = A frame beginning with a START bit is transmitted according to the features defined in the mode register.

This action is necessary when the TWI peripheral wants to read data from a slave. When configured in Master Mode with a write operation, a frame is sent as soon as the user writes a character in the Transmit Holding Register (TWI_THR).

- **STOP: Send a STOP Condition**

0 = No effect.

1 = STOP Condition is sent just after completing the current byte transmission in master read mode.

  – In single data byte master read, the START and STOP must both be set.
  – In multiple data bytes master read, the STOP must be set after the last data received but one.
  – In master read mode, if a NACK bit is received, the STOP is automatically performed.
  – In master data write operation, a STOP condition will be sent after the transmission of the current data is finished.

- **MSEN: TWI Master Mode Enabled**

0 = No effect.

1 = If MSDIS = 0, the master mode is enabled.

Note: Switching from Slave to Master mode is only permitted when TXCOMP = 1.

- **MSDIS: TWI Master Mode Disabled**

0 = No effect.

1 = The master mode is disabled, all pending data is transmitted. The shifter and holding characters (if it contains data) are transmitted in case of write operation. In read operation, the character being transferred must be completely received before disabling.

- **SVEN: TWI Slave Mode Enabled**

0 = No effect.

1 = If SVDIS = 0, the slave mode is enabled.

Note: Switching from Master to Slave mode is only permitted when TXCOMP = 1.

- **SVDIS: TWI Slave Mode Disabled**

0 = No effect.

1 = The slave mode is disabled. The shifter and holding characters (if it contains data) are transmitted in case of read operation. In write operation, the character being transferred must be completely received before disabling.

- **QUICK: SMBUS Quick Command**

0 = No effect.

1 = If Master mode is enabled, a SMBUS Quick Command is sent.

- **SWRST: Software Reset**

0 = No effect.

1 = Equivalent to a system reset.

### 29.11.2 TWI Master Mode Register

**Name:** TWI_MMR

**Addresses:** 0x40018004 (0), 0x4001C004 (1)

Access: Read-write

**Reset:** 0x00000000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | DADR | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | MREAD | – | – | IADRSZ | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

- **IADRSZ: Internal Device Address Size**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | No internal device address |
| 1 | 1_BYTE | One-byte internal device address |
| 2 | 2_BYTE | Two-byte internal device address |
| 3 | 3_BYTE | Three-byte internal device address |

- **MREAD: Master Read Direction**

0 = Master write direction.

1 = Master read direction.

- **DADR: Device Address**

The device address is used to access slave devices in read or write mode. Those bits are only used in Master mode.

### 29.11.3 TWI Slave Mode Register

Name: TWI_SMR

**Addresses:** 0x40018008 (0), 0x4001C008 (1)

Access: Read-write

**Reset:** 0x00000000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | SADR | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

- **SADR: Slave Address**

The slave device address is used in Slave mode in order to be accessed by master devices in read or write mode.

SADR must be programmed before enabling the Slave mode or after a general call. Writes at other times have no effect.

### 29.11.4 TWI Internal Address Register

Name: TWI_IADR

**Addresses:** 0x4001800C (0), 0x4001C00C (1)

Access: Read-write

**Reset:** 0x00000000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| IADR | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| IADR | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| IADR | | | | | | | |

- **IADR: Internal Address**

0, 1, 2 or 3 bytes depending on IADRSZ.

### 29.11.5 TWI Clock Waveform Generator Register

Name: TWI_CWGR

**Addresses:** 0x40018010 (0), 0x4001C010 (1)

Access: Read-write

**Reset:** 0x00000000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| | | | | | CKDIV | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| CHDIV | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CLDIV | | | | | | | |

TWI_CWGR is only used in Master mode.

- **CLDIV: Clock Low Divider**

The SCL low period is defined as follows:

$$T_{low} = ((\text{CLDIV} \times 2^{\text{CKDIV}}) + 4) \times T_{MCK}$$

- **CHDIV: Clock High Divider**

The SCL high period is defined as follows:

$$T_{high} = ((\text{CHDIV} \times 2^{\text{CKDIV}}) + 4) \times T_{MCK}$$

- **CKDIV: Clock Divider**

The CKDIV is used to increase both SCL high and low periods.

### 29.11.6 TWI Status Register

Name: TWI_SR

**Addresses:** 0x40018020 (0), 0x4001C020 (1)

Access: Read-only

**Reset:** 0x0000F009

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TXBUFE | RXBUFF | ENDTX | ENDRX | EOSACC | SCLWS | ARBLST | NACK |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | OVRE | GACC | SVACC | SVREAD | TXRDY | RXRDY | TXCOMP |

- **TXCOMP: Transmission Completed (automatically set / reset)**

TXCOMP used in Master mode:

0 = During the length of the current frame.

1 = When both holding and shifter registers are empty and STOP condition has been sent.

*TXCOMP behavior in Master mode* can be seen in Figure 29-8 on page 479 and in Figure 29-10 on page 480.

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 29-28 on page 495, Figure 29-29 on page 496, Figure 29-30 on page 497 and Figure 29-31 on page 497.

- **RXRDY: Receive Holding Register Ready (automatically set / reset)**

0 = No character has been received since the last TWI_RHR read operation.

1 = A byte has been received in the TWI_RHR since the last read.

*RXRDY behavior in Master mode* can be seen in Figure 29-10 on page 480.

*RXRDY behavior in Slave mode* can be seen in Figure 29-26 on page 494, Figure 29-29 on page 496, Figure 29-30 on page 497 and Figure 29-31 on page 497.

- **TXRDY: Transmit Holding Register Ready (automatically set / reset)**

TXRDY used in Master mode:

0 = The transmit holding register has not been transferred into shift register. Set to 0 when writing into TWI_THR register.

1 = As soon as a data byte is transferred from TWI_THR to internal shifter or if a NACK error is detected, TXRDY is set at the same time as TXCOMP and NACK. TXRDY is also set when MSEN is set (enable TWI).

*TXRDY behavior in Master mode* can be seen in Figure 29-8 on page 479.

TXRDY used in Slave mode:

0 = As soon as data is written in the TWI_THR, until this data has been transmitted and acknowledged (ACK or NACK).

1 = It indicates that the TWI_THR is empty and that data has been transmitted and acknowledged.

If TXRDY is high and if a NACK has been detected, the transmission will be stopped. Thus when TRDY = NACK = 1, the programmer must not fill TWI_THR to avoid losing it.

*TXRDY behavior in Slave mode* can be seen in Figure 29-25 on page 494, Figure 29-28 on page 495, Figure 29-30 on page 497 and Figure 29-31 on page 497.

- **SVREAD: Slave Read (automatically set / reset)**

This bit is only used in Slave mode. When SVACC is low (no Slave access has been detected) SVREAD is irrelevant.

0 = Indicates that a write access is performed by a Master.

1 = Indicates that a read access is performed by a Master.

*SVREAD behavior* can be seen in Figure 29-25 on page 494, Figure 29-26 on page 494, Figure 29-30 on page 497 and Figure 29-31 on page 497.

- **SVACC: Slave Access (automatically set / reset)**

This bit is only used in Slave mode.

0 = TWI is not addressed. SVACC is automatically cleared after a NACK or a STOP condition is detected.

1 = Indicates that the address decoding sequence has matched (A Master has sent SADR). SVACC remains high until a NACK or a STOP condition is detected.

*SVACC behavior* can be seen in Figure 29-25 on page 494, Figure 29-26 on page 494, Figure 29-30 on page 497 and Figure 29-31 on page 497.

- **GACC: General Call Access (clear on read)**

This bit is only used in Slave mode.

0 = No General Call has been detected.

1 = A General Call has been detected. After the detection of General Call, if need be, the programmer may acknowledge this access and decode the following bytes and respond according to the value of the bytes.

*GACC behavior* can be seen in Figure 29-27 on page 495.

- **OVRE: Overrun Error (clear on read)**

This bit is only used in Master mode.

0 = TWI_RHR has not been loaded while RXRDY was set

1 = TWI_RHR has been loaded while RXRDY was set. Reset by read in TWI_SR when TXCOMP is set.

- **NACK: Not Acknowledged (clear on read)**

NACK used in Master mode:

0 = Each data byte has been correctly received by the far-end side TWI slave component.

1 = A data byte has not been acknowledged by the slave component. Set at the same time as TXCOMP.

NACK used in Slave Read mode:

0 = Each data byte has been correctly received by the Master.

1 = In read mode, a data byte has not been acknowledged by the Master. When NACK is set the programmer must not fill TWI_THR even if TXRDY is set, because it means that the Master will stop the data transfer or re initiate it.

Note that in Slave Write mode all data are acknowledged by the TWI.

- **ARBLST: Arbitration Lost (clear on read)**

This bit is only used in Master mode.

0: Arbitration won.

1: Arbitration lost. Another master of the TWI bus has won the multi-master arbitration. TXCOMP is set at the same time.

- **SCLWS: Clock Wait State (automatically set / reset)**

This bit is only used in Slave mode.

0 = The clock is not stretched.

1 = The clock is stretched. TWI_THR / TWI_RHR buffer is not filled / emptied before the emission / reception of a new character.

*SCLWS behavior* can be seen in Figure 29-28 on page 495 and Figure 29-29 on page 496.

- **EOSACC: End Of Slave Access (clear on read)**

This bit is only used in Slave mode.

0 = A slave access is being performing.

1 = The Slave Access is finished. End Of Slave Access is automatically set as soon as SVACC is reset.

*EOSACC behavior* can be seen in Figure 29-30 on page 497 and Figure 29-31 on page 497

- **ENDRX: End of RX buffer**

This bit is only used in Master mode.

0 = The Receive Counter Register has not reached 0 since the last write in TWI_RCR or TWI_RNCR.

1 = The Receive Counter Register has reached 0 since the last write in TWI_RCR or TWI_RNCR.

- **ENDTX: End of TX buffer**

This bit is only used in Master mode.

0 = The Transmit Counter Register has not reached 0 since the last write in TWI_TCR or TWI_TNCR.

1 = The Transmit Counter Register has reached 0 since the last write in TWI_TCR or TWI_TNCR.

- **RXBUFF: RX Buffer Full**

This bit is only used in Master mode.

0 = TWI_RCR or TWI_RNCR have a value other than 0.

1 = Both TWI_RCR and TWI_RNCR have a value of 0.

- **TXBUFE: TX Buffer Empty**

This bit is only used in Master mode.

0 = TWI_TCR or TWI_TNCR have a value other than 0.

1 = Both TWI_TCR and TWI_TNCR have a value of 0.

### 29.11.7 TWI Interrupt Enable Register

Name: TWI_IER

**Addresses:** 0x40018024 (0), 0x4001C024 (1)

Access: Write-only

**Reset:** 0x00000000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TXBUFE | RXBUFF | ENDTX | ENDRX | EOSACC | SCL_WS | ARBLST | NACK |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | OVRE | GACC | SVACC | – | TXRDY | RXRDY | TXCOMP |

- **TXCOMP: Transmission Completed Interrupt Enable**
- **RXRDY: Receive Holding Register Ready Interrupt Enable**
- **TXRDY: Transmit Holding Register Ready Interrupt Enable**
- **SVACC: Slave Access Interrupt Enable**
- **GACC: General Call Access Interrupt Enable**
- **OVRE: Overrun Error Interrupt Enable**
- **NACK: Not Acknowledge Interrupt Enable**
- **ARBLST: Arbitration Lost Interrupt Enable**
- **SCL_WS: Clock Wait State Interrupt Enable**
- **EOSACC: End Of Slave Access Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable**
- **ENDTX: End of Transmit Buffer Interrupt Enable**
- **RXBUFF: Receive Buffer Full Interrupt Enable**
- **TXBUFE: Transmit Buffer Empty Interrupt Enable**

0 = No effect.

1 = Enables the corresponding interrupt.

### 29.11.8 TWI Interrupt Disable Register

Name: TWI_IDR

**Addresses:** 0x40018028 (0), 0x4001C028 (1)

Access: Write-only

**Reset:** 0x00000000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TXBUFE | RXBUFF | ENDTX | ENDRX | EOSACC | SCL_WS | ARBLST | NACK |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | OVRE | GACC | SVACC | – | TXRDY | RXRDY | TXCOMP |

- **TXCOMP: Transmission Completed Interrupt Disable**
- **RXRDY: Receive Holding Register Ready Interrupt Disable**
- **TXRDY: Transmit Holding Register Ready Interrupt Disable**
- **SVACC: Slave Access Interrupt Disable**
- **GACC: General Call Access Interrupt Disable**
- **OVRE: Overrun Error Interrupt Disable**
- **NACK: Not Acknowledge Interrupt Disable**
- **ARBLST: Arbitration Lost Interrupt Disable**
- **SCL_WS: Clock Wait State Interrupt Disable**
- **EOSACC: End Of Slave Access Interrupt Disable**
- **ENDRX: End of Receive Buffer Interrupt Disable**
- **ENDTX: End of Transmit Buffer Interrupt Disable**
- **RXBUFF: Receive Buffer Full Interrupt Disable**
- **TXBUFE: Transmit Buffer Empty Interrupt Disable**

0 = No effect.

1 = Disables the corresponding interrupt.

### 29.11.9 TWI Interrupt Mask Register

Name: TWI_IMR

**Addresses:** 0x4001802C (0), 0x4001C02C (1)

Access: Read-only

**Reset:** 0x00000000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TXBUFE | RXBUFF | ENDTX | ENDRX | EOSACC | SCL_WS | ARBLST | NACK |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | OVRE | GACC | SVACC | – | TXRDY | RXRDY | TXCOMP |

- **TXCOMP: Transmission Completed Interrupt Mask**
- **RXRDY: Receive Holding Register Ready Interrupt Mask**
- **TXRDY: Transmit Holding Register Ready Interrupt Mask**
- **SVACC: Slave Access Interrupt Mask**
- **GACC: General Call Access Interrupt Mask**
- **OVRE: Overrun Error Interrupt Mask**
- **NACK: Not Acknowledge Interrupt Mask**
- **ARBLST: Arbitration Lost Interrupt Mask**
- **SCL_WS: Clock Wait State Interrupt Mask**
- **EOSACC: End Of Slave Access Interrupt Mask**
- **ENDRX: End of Receive Buffer Interrupt Mask**
- **ENDTX: End of Transmit Buffer Interrupt Mask**
- **RXBUFF: Receive Buffer Full Interrupt Mask**
- **TXBUFE: Transmit Buffer Empty Interrupt Mask**

0 = The corresponding interrupt is disabled.

1 = The corresponding interrupt is enabled.

### 29.11.10TWI Receive Holding Register

Name: TWI_RHR

**Addresses:** 0x40018030 (0), 0x4001C030 (1)

Access: Read-only

**Reset:** 0x00000000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXDATA | | | | | | | |

- **RXDATA: Master or Slave Receive Holding Data**

### 29.11.11TWI Transmit Holding Register

Name: TWI_THR

**Addresses:** 0x40018034 (0), 0x4001C034 (1)

Access: Read-write

**Reset:** 0x00000000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXDATA | | | | | | | |

- **TXDATA: Master or Slave Transmit Holding Data**

# 30. Universal Asynchronous Receiver Transceiver (UART)

## 30.1 Description

The Universal Asynchronous Receiver Transmitter features a two-pin UART that can be used for communication and trace purposes and offers an ideal medium for in-situ programming solutions. Moreover, the association with two peripheral DMA controller (PDC) channels permits packet handling for these tasks with processor time reduced to a minimum.

## 30.2 Embedded Characteristics

- Two-pin UART
  - Implemented Features are USART Compatible
  - Independent Receiver and Transmitter with a Common Programmable Baud Rate Generator
  - Even, Odd, Mark or Space Parity Generation
  - Parity, Framing and Overrun Error Detection
  - Automatic Echo, Local Loopback and Remote Loopback Channel Modes
  - Interrupt Generation
  - Support for Two PDC Channels with Connection to Receiver and Transmitter

## 30.3 Block Diagram

**Figure 30-1. UART Functional Block Diagram**

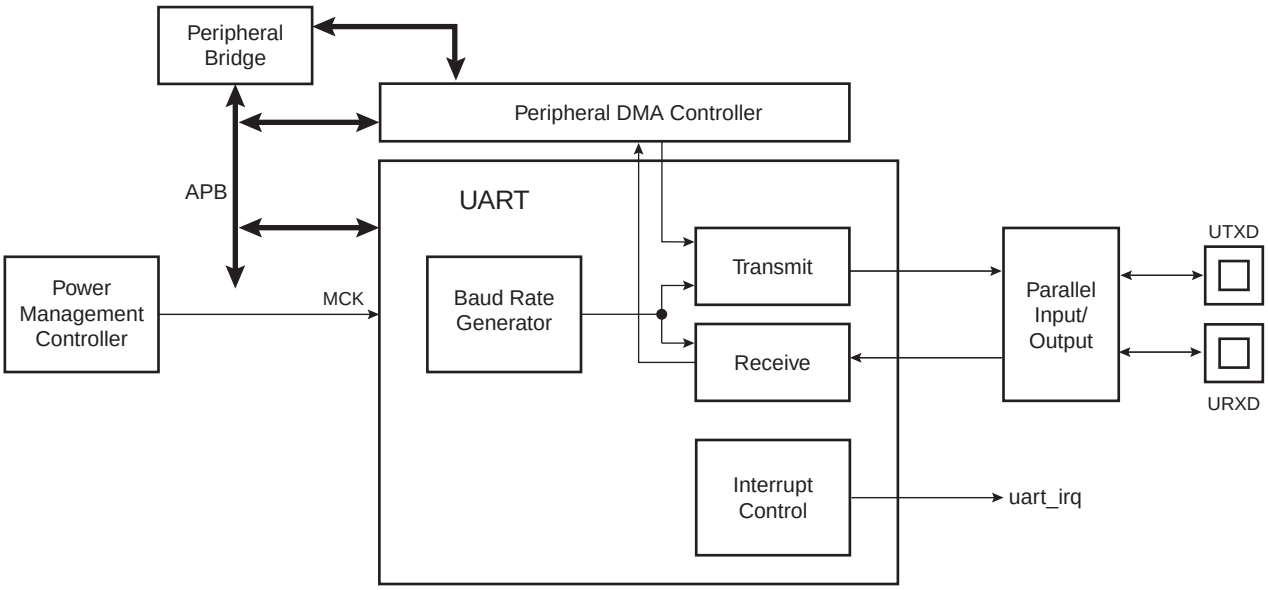



**Table 30-1. UART Pin Description**

| Pin Name | Description | Type |
|---|---|---|
| URXD | UART Receive Data | Input |
| UTXD | UART Transmit Data | Output |

## 30.4 Product Dependencies

### 30.4.1 I/O Lines

The UART pins are multiplexed with PIO lines. The programmer must first configure the corresponding PIO Controller to enable I/O line operations of the UART.

**Table 30-2. I/O Lines**

| Instance | Signal | I/O Line | Peripheral |
|---|---|---|---|
| UART0 | URXD0 | PA9 | A |
| UART0 | UTXD0 | PA10 | A |
| UART1 | URXD1 | PB2 | A |
| UART1 | UTXD1 | PB3 | A |

### 30.4.2 Power Management

The UART clock is controllable through the Power Management Controller. In this case, the programmer must first configure the PMC to enable the UART clock. Usually, the peripheral identifier used for this purpose is 1.

### 30.4.3 Interrupt Source

The UART interrupt line is connected to one of the interrupt sources of the Nested Vectored Interrupt Controller (NVIC). Interrupt handling requires programming of the NVIC before configuring the UART.

## 30.5 UART Operations

The UART operates in asynchronous mode only and supports only 8-bit character handling (with parity). It has no clock pin.

The UART is made up of a receiver and a transmitter that operate independently, and a common baud rate generator. Receiver timeout and transmitter time guard are not implemented. However, all the implemented features are compatible with those of a standard USART.

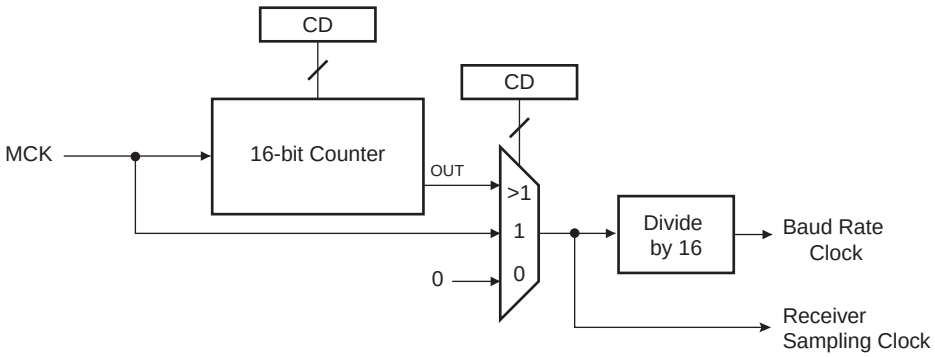
### 30.5.1 Baud Rate Generator

The baud rate generator provides the bit period clock named baud rate clock to both the receiver and the transmitter.

The baud rate clock is the master clock divided by 16 times the value (CD) written in UART_BRGR (Baud Rate Generator Register). If UART_BRGR is set to 0, the baud rate clock is disabled and the UART remains inactive. The maximum allowable baud rate is Master Clock divided by 16. The minimum allowable baud rate is Master Clock divided by (16 x 65536).

$$\text{Baud Rate} = \frac{\text{MCK}}{16 \times \text{CD}}$$

**Figure 30-2. Baud Rate Generator**




### 30.5.2 Receiver

#### 30.5.2.1 Receiver Reset, Enable and Disable

After device reset, the UART receiver is disabled and must be enabled before being used. The receiver can be enabled by writing the control register UART_CR with the bit RXEN at 1. At this command, the receiver starts looking for a start bit.

The programmer can disable the receiver by writing UART_CR with the bit RXDIS at 1. If the receiver is waiting for a start bit, it is immediately stopped. However, if the receiver has already detected a start bit and is receiving the data, it waits for the stop bit before actually stopping its operation.

The programmer can also put the receiver in its reset state by writing UART_CR with the bit RSTRX at 1. In doing so, the receiver immediately stops its current operations and is disabled, whatever its current state. If RSTRX is applied when data is being processed, this data is lost.

#### 30.5.2.2 Start Detection and Data Sampling

The UART only supports asynchronous operations, and this affects only its receiver. The UART receiver detects the start of a received character by sampling the URXD signal until it detects a valid start bit. A low level (space) on URXD is interpreted as a valid start bit if it is detected for more than 7 cycles of the sampling clock, which is 16

times the baud rate. Hence, a space that is longer than 7/16 of the bit period is detected as a valid start bit. A space which is 7/16 of a bit period or shorter is ignored and the receiver continues to wait for a valid start bit.

When a valid start bit has been detected, the receiver samples the URXD at the theoretical midpoint of each bit. It is assumed that each bit lasts 16 cycles of the sampling clock (1-bit period) so the bit sampling point is eight cycles (0.5-bit period) after the start of the bit. The first sampling point is therefore 24 cycles (1.5-bit periods) after the falling edge of the start bit was detected.

Each subsequent bit is sampled 16 cycles (1-bit period) after the previous one.

**Figure 30-3. Start Bit Detection**

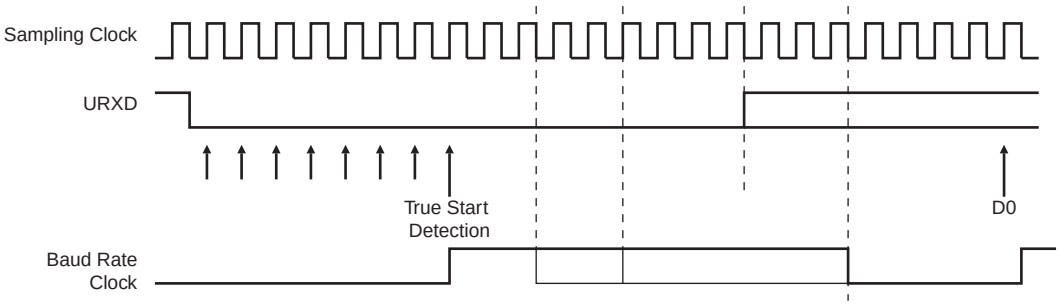
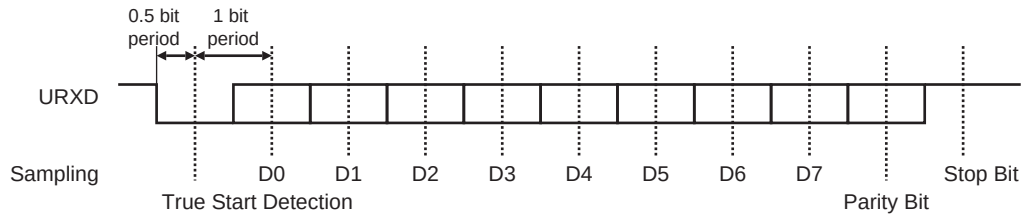



**Figure 30-4. Character Reception**

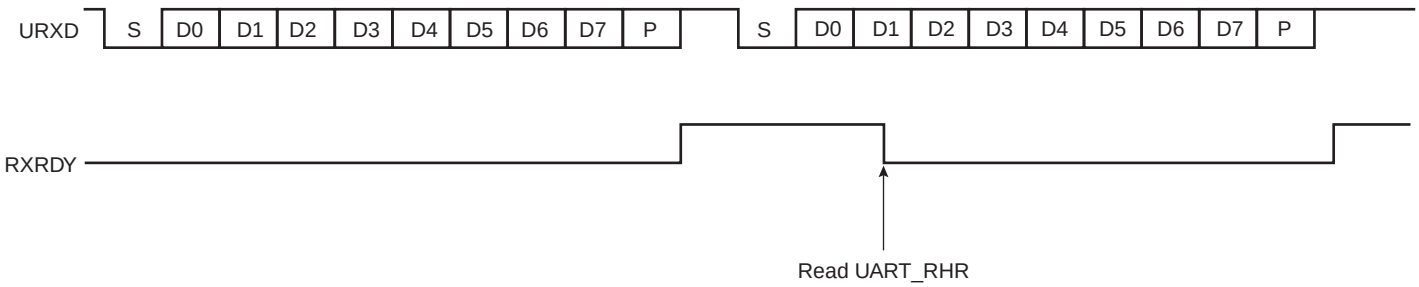
Example: 8-bit, parity enabled 1 stop




### 30.5.2.3 Receiver Ready

When a complete character is received, it is transferred to the UART_RHR and the RXRDY status bit in UART_SR (Status Register) is set. The bit RXRDY is automatically cleared when the receive holding register UART_RHR is read.

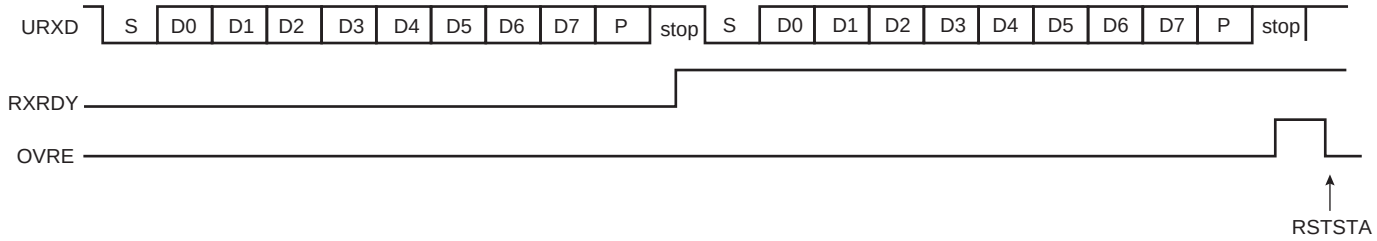
**Figure 30-5. Receiver Ready**



#### 30.5.2.4 Receiver Overrun

If UART_RHR has not been read by the software (or the Peripheral Data Controller or DMA Controller) since the last transfer, the RXRDY bit is still set and a new character is received, the OVRE status bit in UART_SR is set. OVRE is cleared when the software writes the control register UART_CR with the bit RSTSTA (Reset Status) at 1.

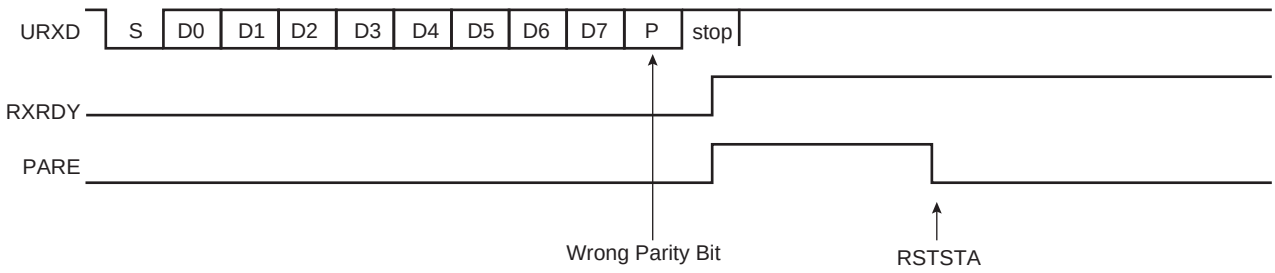
**Figure 30-6. Receiver Overrun**




#### 30.5.2.5 Parity Error

Each time a character is received, the receiver calculates the parity of the received data bits, in accordance with the field PAR in UART_MR. It then compares the result with the received parity bit. If different, the parity error bit PARE in UART_SR is set at the same time the RXRDY is set. The parity bit is cleared when the control register UART_CR is written with the bit RSTSTA (Reset Status) at 1. If a new character is received before the reset status command is written, the PARE bit remains at 1.

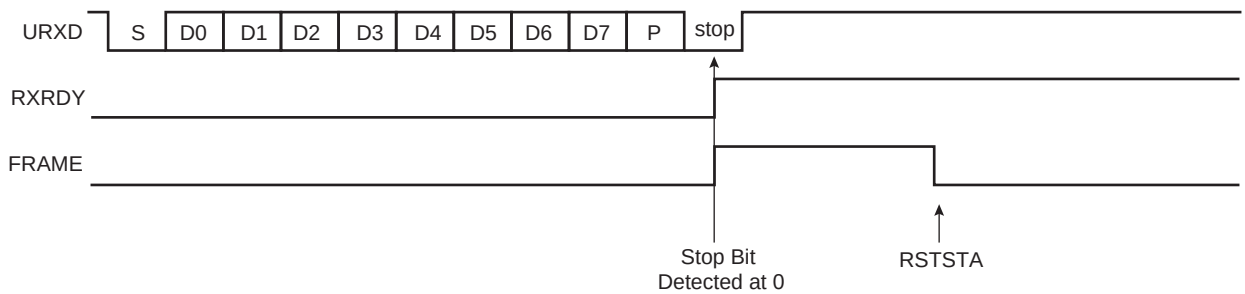
**Figure 30-7. Parity Error**




#### 30.5.2.6 Receiver Framing Error

When a start bit is detected, it generates a character reception when all the data bits have been sampled. The stop bit is also sampled and when it is detected at 0, the FRAME (Framing Error) bit in UART_SR is set at the same time the RXRDY bit is set. The FRAME bit remains high until the control register UART_CR is written with the bit RSTSTA at 1.

**Figure 30-8. Receiver Framing Error**



### 30.5.3 Transmitter

#### 30.5.3.1 Transmitter Reset, Enable and Disable

After device reset, the UART transmitter is disabled and it must be enabled before being used. The transmitter is enabled by writing the control register UART_CR with the bit TXEN at 1. From this command, the transmitter waits for a character to be written in the Transmit Holding Register (UART_THR) before actually starting the transmission.

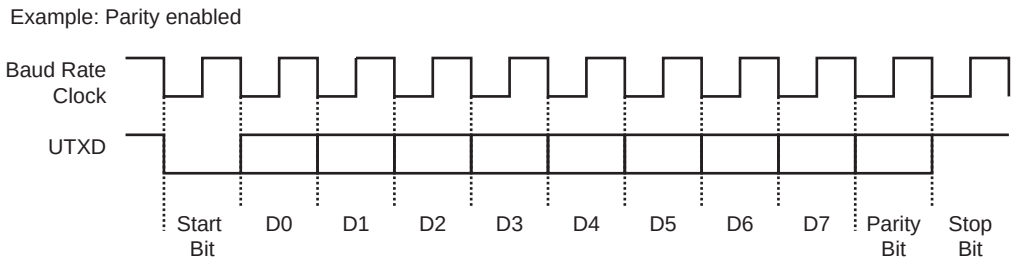
The programmer can disable the transmitter by writing UART_CR with the bit TXDIS at 1. If the transmitter is not operating, it is immediately stopped. However, if a character is being processed into the Shift Register and/or a character has been written in the Transmit Holding Register, the characters are completed before the transmitter is actually stopped.

The programmer can also put the transmitter in its reset state by writing the UART_CR with the bit RSTTX at 1. This immediately stops the transmitter, whether or not it is processing characters.

#### 30.5.3.2 Transmit Format

The UART transmitter drives the pin UTXD at the baud rate clock speed. The line is driven depending on the format defined in the Mode Register and the data stored in the Shift Register. One start bit at level 0, then the 8 data bits, from the lowest to the highest bit, one optional parity bit and one stop bit at 1 are consecutively shifted out as shown in the following figure. The field PARE in the mode register UART_MR defines whether or not a parity bit is shifted out. When a parity bit is enabled, it can be selected between an odd parity, an even parity, or a fixed space or mark bit.

**Figure 30-9. Character Transmission**

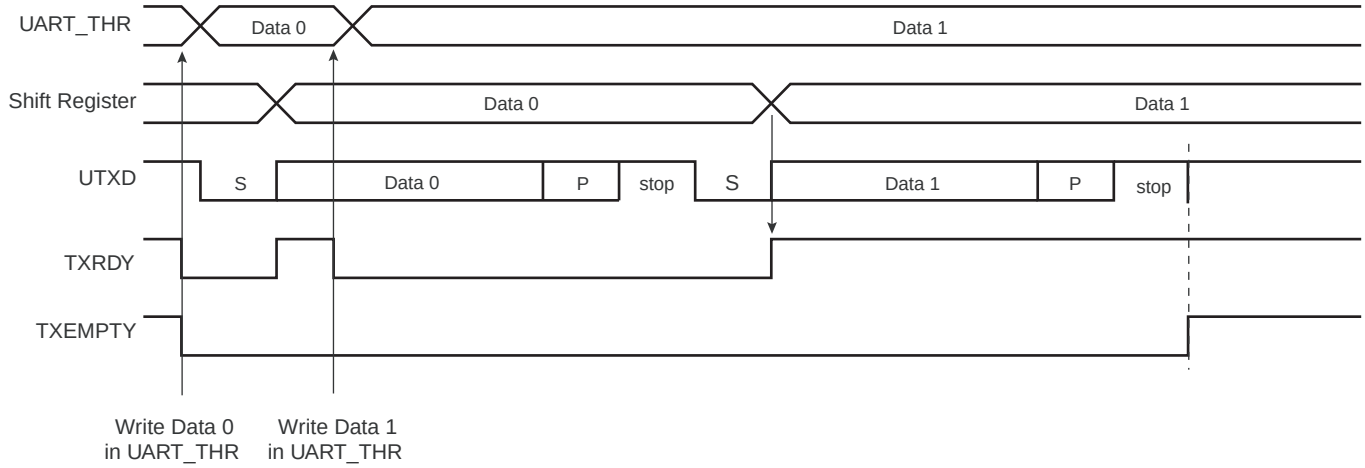



#### 30.5.3.3 Transmitter Control

When the transmitter is enabled, the bit TXRDY (Transmitter Ready) is set in the status register UART_SR. The transmission starts when the programmer writes in the Transmit Holding Register (UART_THR), and after the written character is transferred from UART_THR to the Shift Register. The TXRDY bit remains high until a second character is written in UART_THR. As soon as the first character is completed, the last character written in UART_THR is transferred into the shift register and TXRDY rises again, showing that the holding register is empty.

When both the Shift Register and UART_THR are empty, i.e., all the characters written in UART_THR have been processed, the TXEMPTY bit rises after the last stop bit has been completed.

**Figure 30-10. Transmitter Control**




### 30.5.4 Peripheral DMA Controller

Both the receiver and the transmitter of the UART are connected to a Peripheral DMA Controller (PDC) channel.

The peripheral data controller channels are programmed via registers that are mapped within the UART user interface from the offset 0x100. The status bits are reported in the UART status register (UART_SR) and can generate an interrupt.

The RXRDY bit triggers the PDC channel data transfer of the receiver. This results in a read of the data in UART_RHR. The TXRDY bit triggers the PDC channel data transfer of the transmitter. This results in a write of data in UART_THR.

### 30.5.5 Test Modes

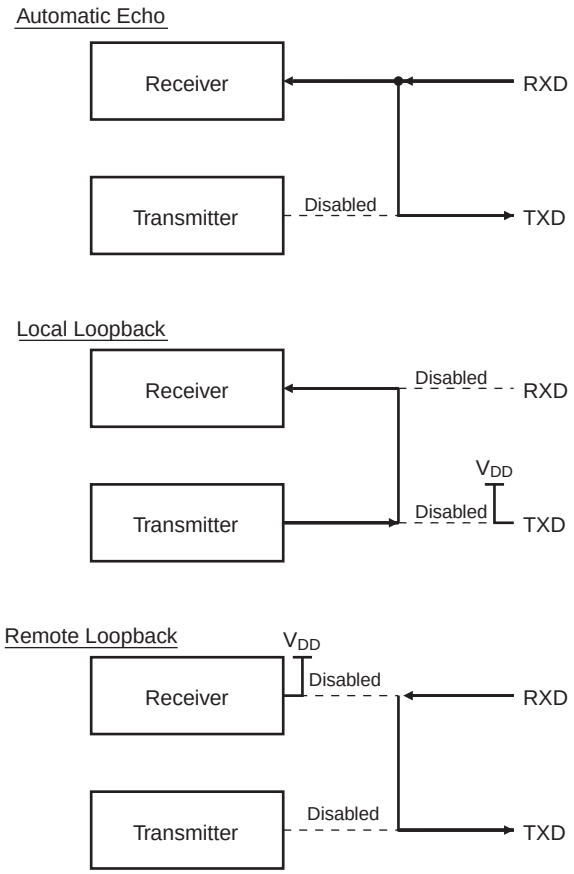
The UART supports three test modes. These modes of operation are programmed by using the field CHMODE (Channel Mode) in the mode register (UART_MR).

The Automatic Echo mode allows bit-by-bit retransmission. When a bit is received on the URXD line, it is sent to the UTXD line. The transmitter operates normally, but has no effect on the UTXD line.

The Local Loopback mode allows the transmitted characters to be received. UTXD and URXD pins are not used and the output of the transmitter is internally connected to the input of the receiver. The URXD pin level has no effect and the UTXD line is held high, as in idle state.

The Remote Loopback mode directly connects the URXD pin to the UTXD line. The transmitter and the receiver are disabled and have no effect. This mode allows a bit-by-bit retransmission.

**Figure 30-11. Test Modes**



## 30.6 Universal Asynchronous Receiver Transmitter (UART) User Interface

**Table 30-3. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x0000 | Control Register | UART_CR | Write-only | – |
| 0x0004 | Mode Register | UART_MR | Read-write | 0x0 |
| 0x0008 | Interrupt Enable Register | UART_IER | Write-only | – |
| 0x000C | Interrupt Disable Register | UART_IDR | Write-only | – |
| 0x0010 | Interrupt Mask Register | UART_IMR | Read-only | 0x0 |
| 0x0014 | Status Register | UART_SR | Read-only | – |
| 0x0018 | Receive Holding Register | UART_RHR | Read-only | 0x0 |
| 0x001C | Transmit Holding Register | UART_THR | Write-only | – |
| 0x0020 | Baud Rate Generator Register | UART_BRGR | Read-write | 0x0 |
| 0x0024 - 0x003C | Reserved | – | – | – |
| 0x004C - 0x00FC | Reserved | – | – | – |
| 0x0100 - 0x0124 | PDC Area | – | – | – |

### 30.6.1 UART Control Register

**Name:** UART_CR

**Addresses:** 0x400E0600 (0), 0x400E0800 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | RSTSTA |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXDIS | TXEN | RXDIS | RXEN | RSTTX | RSTRX | – | – |

- **RSTRX: Reset Receiver**

0 = No effect.

1 = The receiver logic is reset and disabled. If a character is being received, the reception is aborted.

- **RSTTX: Reset Transmitter**

0 = No effect.

1 = The transmitter logic is reset and disabled. If a character is being transmitted, the transmission is aborted.

- **RXEN: Receiver Enable**

0 = No effect.

1 = The receiver is enabled if RXDIS is 0.

- **RXDIS: Receiver Disable**

0 = No effect.

1 = The receiver is disabled. If a character is being processed and RSTRX is not set, the character is completed before the receiver is stopped.

- **TXEN: Transmitter Enable**

0 = No effect.

1 = The transmitter is enabled if TXDIS is 0.

- **TXDIS: Transmitter Disable**

0 = No effect.

1 = The transmitter is disabled. If a character is being processed and a character has been written in the UART_THR and RSTTX is not set, both characters are completed before the transmitter is stopped.

- **RSTSTA: Reset Status Bits**

0 = No effect.

1 = Resets the status bits PARE, FRAME and OVRE in the UART_SR.

### 30.6.2 UART Mode Register

**Name:** UART_MR

**Addresses:** 0x400E0604 (0), 0x400E0804 (1)

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| CHMODE | | – | – | PAR | | | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | – |

- **PAR: Parity Type**

| Value | Name | Description |
|---|---|---|
| 0 | EVEN | Even parity |
| 1 | ODD | Odd parity |
| 2 | SPACE | Space: parity forced to 0 |
| 3 | MARK | Mark: parity forced to 1 |
| 4 | NO | No parity |

- **CHMODE: Channel Mode**

| Value | Name | Description |
|---|---|---|
| 0 | NORMAL | Normal Mode |
| 1 | AUTOMATIC | Automatic Echo |
| 2 | LOCAL_LOOPBACK | Local Loopback |
| 3 | REMOTE_LOOPBACK | Remote Loopback |

### 30.6.3 UART Interrupt Enable Register

**Name:** UART_IER

**Addresses:** 0x400E0608 (0), 0x400E0808 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | RXBUFF | TXBUFE | – | TXEMPTY | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PARE | FRAME | OVRE | ENDTX | ENDRX | – | TXRDY | RXRDY |

- **RXRDY: Enable RXRDY Interrupt**
- **TXRDY: Enable TXRDY Interrupt**
- **ENDRX: Enable End of Receive Transfer Interrupt**
- **ENDTX: Enable End of Transmit Interrupt**
- **OVRE: Enable Overrun Error Interrupt**
- **FRAME: Enable Framing Error Interrupt**
- **PARE: Enable Parity Error Interrupt**
- **TXEMPTY: Enable TXEMPTY Interrupt**
- **TXBUFE: Enable Buffer Empty Interrupt**
- **RXBUFF: Enable Buffer Full Interrupt**

0 = No effect.

1 = Enables the corresponding interrupt.

### 30.6.4 UART Interrupt Disable Register

**Name:** UART_IDR

**Addresses:** 0x400E060C (0), 0x400E080C (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | RXBUFF | TXBUFE | – | TXEMPTY | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PARE | FRAME | OVRE | ENDTX | ENDRX | – | TXRDY | RXRDY |

- **RXRDY: Disable RXRDY Interrupt**
- **TXRDY: Disable TXRDY Interrupt**
- **ENDRX: Disable End of Receive Transfer Interrupt**
- **ENDTX: Disable End of Transmit Interrupt**
- **OVRE: Disable Overrun Error Interrupt**
- **FRAME: Disable Framing Error Interrupt**
- **PARE: Disable Parity Error Interrupt**
- **TXEMPTY: Disable TXEMPTY Interrupt**
- **TXBUFE: Disable Buffer Empty Interrupt**
- **RXBUFF: Disable Buffer Full Interrupt**

0 = No effect.

1 = Disables the corresponding interrupt.

### 30.6.5 UART Interrupt Mask Register

**Name:** UART_IMR

**Addresses:** 0x400E0610 (0), 0x400E0810 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | RXBUFF | TXBUFE | – | TXEMPTY | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PARE | FRAME | OVRE | ENDTX | ENDRX | – | TXRDY | RXRDY |

- **RXRDY: Mask RXRDY Interrupt**
- **TXRDY: Disable TXRDY Interrupt**
- **ENDRX: Mask End of Receive Transfer Interrupt**
- **ENDTX: Mask End of Transmit Interrupt**
- **OVRE: Mask Overrun Error Interrupt**
- **FRAME: Mask Framing Error Interrupt**
- **PARE: Mask Parity Error Interrupt**
- **TXEMPTY: Mask TXEMPTY Interrupt**
- **TXBUFE: Mask TXBUFE Interrupt**
- **RXBUFF: Mask RXBUFF Interrupt**

0 = The corresponding interrupt is disabled.

1 = The corresponding interrupt is enabled.

### 30.6.6 UART Status Register

**Name:** UART_SR

**Addresses:** 0x400E0614 (0), 0x400E0814 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | RXBUFF | TXBUFE | – | TXEMPTY | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PARE | FRAME | OVRE | ENDTX | ENDRX | – | TXRDY | RXRDY |

- **RXRDY: Receiver Ready**

0 = No character has been received since the last read of the UART_RHR or the receiver is disabled.

1 = At least one complete character has been received, transferred to UART_RHR and not yet read.

- **TXRDY: Transmitter Ready**

0 = A character has been written to UART_THR and not yet transferred to the Shift Register, or the transmitter is disabled.

1 = There is no character written to UART_THR not yet transferred to the Shift Register.

- **ENDRX: End of Receiver Transfer**

0 = The End of Transfer signal from the receiver Peripheral Data Controller channel is inactive.

1 = The End of Transfer signal from the receiver Peripheral Data Controller channel is active.

- **ENDTX: End of Transmitter Transfer**

0 = The End of Transfer signal from the transmitter Peripheral Data Controller channel is inactive.

1 = The End of Transfer signal from the transmitter Peripheral Data Controller channel is active.

- **OVRE: Overrun Error**

0 = No overrun error has occurred since the last RSTSTA.

1 = At least one overrun error has occurred since the last RSTSTA.

- **FRAME: Framing Error**

0 = No framing error has occurred since the last RSTSTA.

1 = At least one framing error has occurred since the last RSTSTA.

- **PARE: Parity Error**

0 = No parity error has occurred since the last RSTSTA.

1 = At least one parity error has occurred since the last RSTSTA.

- **TXEMPTY: Transmitter Empty**

0 = There are characters in UART_THR, or characters being processed by the transmitter, or the transmitter is disabled.

1 = There are no characters in UART_THR and there are no characters being processed by the transmitter.

- **TXBUFE: Transmission Buffer Empty**

0 = The buffer empty signal from the transmitter PDC channel is inactive.

1 = The buffer empty signal from the transmitter PDC channel is active.

- **RXBUFF: Receive Buffer Full**

0 = The buffer full signal from the receiver PDC channel is inactive.

1 = The buffer full signal from the receiver PDC channel is active.

### 30.6.7 UART Receiver Holding Register

**Name:** UART_RHR

**Addresses:** 0x400E0618 (0), 0x400E0818 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXCHR | | | | | | | |

- **RXCHR: Received Character**

Last received character if RXRDY is set.

### 30.6.8 UART Transmit Holding Register

**Name:** UART_THR

**Addresses:** 0x400E061C (0), 0x400E081C (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXCHR | | | | | | | |

- **TXCHR: Character to be Transmitted**

Next character to be transmitted after the current character if TXRDY is not set.

### 30.6.9 UART Baud Rate Generator Register

**Name:** UART_BRGR

**Addresses:** 0x400E0620 (0), 0x400E0820 (1)

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CD | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CD | | | | | | | |

- **CD: Clock Divisor**

0 = Baud Rate Clock is disabled

1 to 65,535 = MCK / (CD x 16)

# 31. Universal Synchronous Asynchronous Receiver Transmitter (USART)

## 31.1 Description

The Universal Synchronous Asynchronous Receiver Transceiver (USART) provides one full duplex universal synchronous asynchronous serial link. Data frame format is widely programmable (data length, parity, number of stop bits) to support a maximum of standards. The receiver implements parity error, framing error and overrun error detection. The receiver time-out enables handling variable-length frames and the transmitter timeguard facilitates communications with slow remote devices. Multidrop communications are also supported through address bit handling in reception and transmission.

The USART features three test modes: remote loopback, local loopback and automatic echo.

The USART supports specific operating modes providing interfaces on RS485 and SPI buses, with ISO7816 T = 0 or T = 1 smart card slots and infrared transceivers (ISO7816 only on USART0). The hardware handshaking feature enables an out-of-band flow control by automatic management of the pins RTS and CTS.

The USART supports the connection to the Peripheral DMA Controller, which enables data transfers to the transmitter and from the receiver. The PDC provides chained buffer management without any intervention of the processor.

## 31.2 Embedded Characteristics

- Programmable Baud Rate Generator
- 5- to 9-bit Full-duplex Synchronous or Asynchronous Serial Communications
  - 1, 1.5 or 2 Stop Bits in Asynchronous Mode or 1 or 2 Stop Bits in Synchronous Mode
  - Parity Generation and Error Detection
  - Framing Error Detection, Overrun Error Detection
  - MSB- or LSB-first
  - Optional Break Generation and Detection
  - By 8 or by 16 Over-sampling Receiver Frequency
  - Optional Hardware Handshaking RTS-CTS
  - Receiver Time-out and Transmitter Timeguard
  - Optional Multidrop Mode with Address Generation and Detection
- RS485 with Driver Control Signal
- ISO7816, T = 0 or T = 1 Protocols for Interfacing with Smart Cards (Only on USART0)
  - NACK Handling, Error Counter with Repetition and Iteration Limit
- IrDA Modulation and Demodulation (Only on USART0)
  - Communication at up to 115.2 Kbps
- SPI Mode
  - Master or Slave
  - Serial Clock Programmable Phase and Polarity
  - SPI Serial Clock (SCK) Frequency up to Internal Clock Frequency MCK/6
- Test Modes
  - Remote Loopback, Local Loopback, Automatic Echo
- Supports Connection of Two Peripheral DMA Controller Channels (PDC)
  - Offers Buffer Transfer without Processor Intervention

## 31.3 Block Diagram

**Figure 31-1. USART Block Diagram**

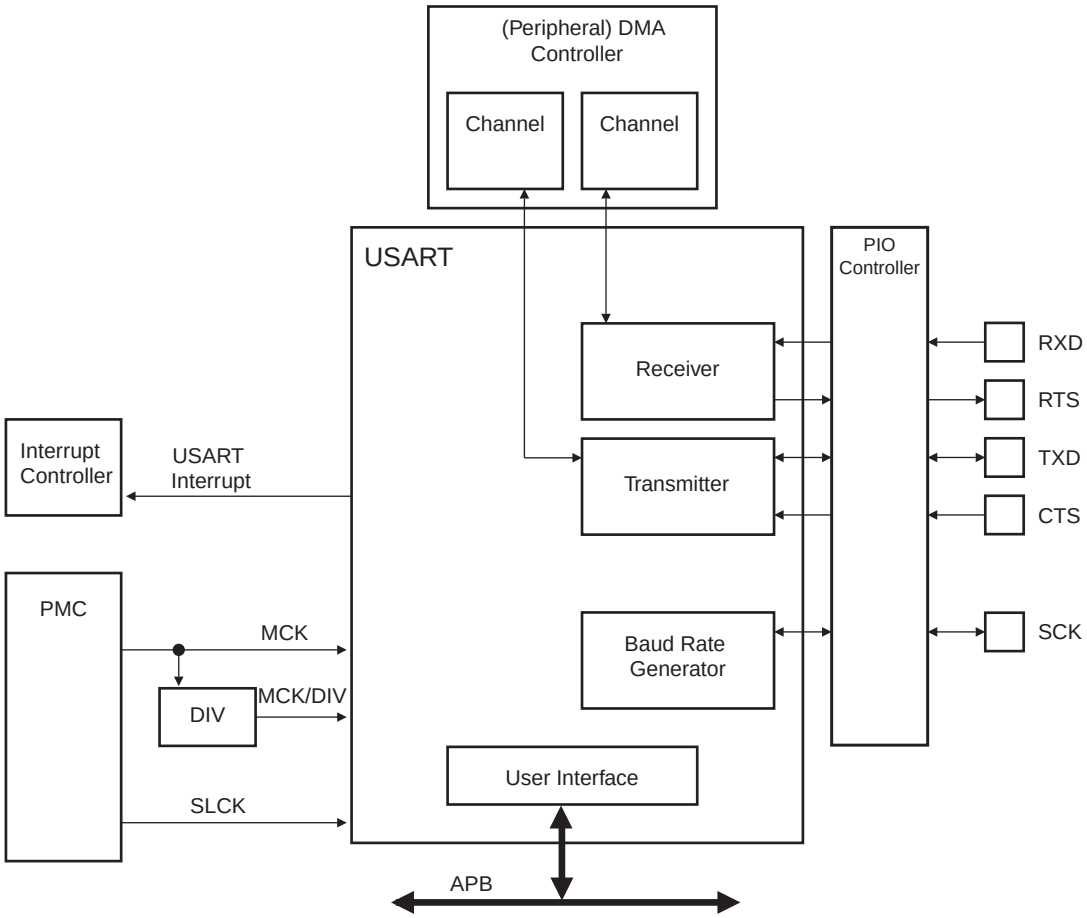
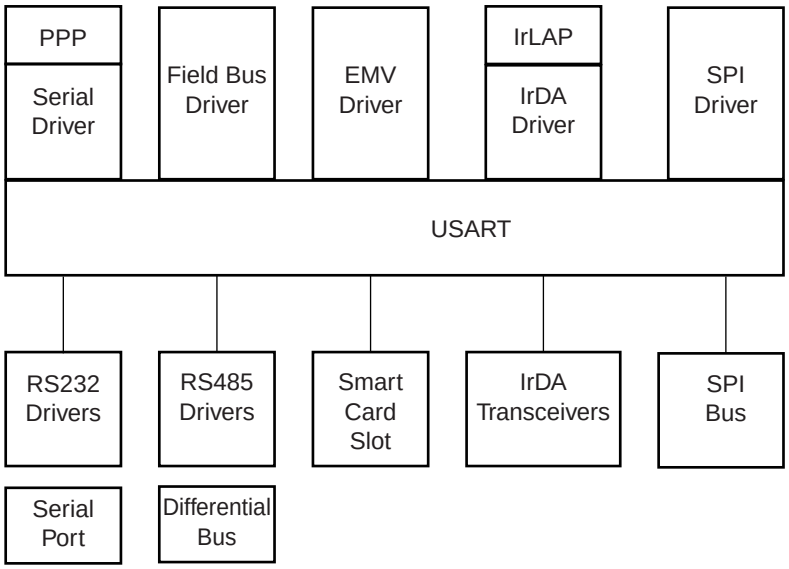



**Table 31-1. SPI Operating Mode**

| PIN | USART | SPI Slave | SPI Master |
|---|---|---|---|
| RXD | RXD | MOSI | MISO |
| TXD | TXD | MISO | MOSI |
| RTS | RTS | – | CS |
| CTS | CTS | CS | – |

## 31.4 Application Block Diagram

**Figure 31-2. Application Block Diagram**




## 31.5 I/O Lines Description

**Table 31-2. I/O Line Description**

| Name | Description | Type | Active Level |
|---|---|---|---|
| SCK | Serial Clock | I/O | |
| TXD | Transmit Serial Data<br>or Master Out Slave In (MOSI) in SPI Master Mode<br>or Master In Slave Out (MISO) in SPI Slave Mode | I/O | |
| RXD | Receive Serial Data<br>or Master In Slave Out (MISO) in SPI Master Mode<br>or Master Out Slave In (MOSI) in SPI Slave Mode | Input | |
| CTS | Clear to Send<br>or Slave Select (NSS) in SPI Slave Mode | Input | Low |
| RTS | Request to Send<br>or Slave Select (NSS) in SPI Master Mode | Output | Low |

## 31.6 Product Dependencies

### 31.6.1 I/O Lines

The pins used for interfacing the USART may be multiplexed with the PIO lines. The programmer must first program the PIO controller to assign the desired USART pins to their peripheral function. If I/O lines of the USART are not used by the application, they can be used for other purposes by the PIO Controller.

To prevent the TXD line from falling when the USART is disabled, the use of an internal pull up is mandatory. If the hardware handshaking feature is used, the internal pull up on TXD must also be enabled.

**Table 31-3. I/O Lines**

| Instance | Signal | I/O Line | Peripheral |
|---|---|---|---|
| USART0 | CTS0 | PA8 | A |
| USART0 | RTS0 | PA7 | A |
| USART0 | RXD0 | PA5 | A |
| USART0 | SCK0 | PA2 | B |
| USART0 | TXD0 | PA6 | A |
| USART1 | CTS1 | PA25 | A |
| USART1 | RTS1 | PA24 | A |
| USART1 | RXD1 | PA21 | A |
| USART1 | SCK1 | PA23 | A |
| USART1 | TXD1 | PA22 | A |

### 31.6.2 Power Management

The USART is not continuously clocked. The programmer must first enable the USART Clock in the Power Management Controller (PMC) before using the USART. However, if the application does not require USART operations, the USART clock can be stopped when not needed and be restarted later. In this case, the USART will resume its operations where it left off.

Configuring the USART does not require the USART clock to be enabled.

### 31.6.3 Interrupt

The USART interrupt line is connected on one of the internal sources of the Interrupt Controller. Using the USART interrupt requires the Interrupt Controller to be programmed first. Note that it is not recommended to use the USART interrupt line in edge sensitive mode.

**Table 31-4. Peripheral IDs**

| Instance | ID |
|---|---|
| USART0 | 14 |
| USART1 | 15 |

## 31.7 Functional Description

The USART is capable of managing several types of serial synchronous or asynchronous communications.

It supports the following communication modes:

- 5- to 9-bit full-duplex asynchronous serial communication
  - MSB- or LSB-first
  - 1, 1.5 or 2 stop bits
  - Parity even, odd, marked, space or none
  - By 8 or by 16 over-sampling receiver frequency
  - Optional hardware handshaking
  - Optional break management
  - Optional multidrop serial communication
- High-speed 5- to 9-bit full-duplex synchronous serial communication
  - MSB- or LSB-first
  - 1 or 2 stop bits
  - Parity even, odd, marked, space or none
  - By 8 or by 16 over-sampling frequency
  - Optional hardware handshaking
  - Optional break management
  - Optional multidrop serial communication
- RS485 with driver control signal
- ISO7816, T0 or T1 protocols for interfacing with smart cards (Only on USART0)
  - NACK handling, error counter with repetition and iteration limit, inverted data.
- InfraRed IrDA Modulation and Demodulation
- SPI Mode
  - Master or Slave
  - Serial Clock Programmable Phase and Polarity
  - SPI Serial Clock (SCK) Frequency up to Internal Clock Frequency MCK/6
- Test modes
  - Remote loopback, local loopback, automatic echo

### 31.7.1 Baud Rate Generator

The Baud Rate Generator provides the bit period clock named the Baud Rate Clock to both the receiver and the transmitter.

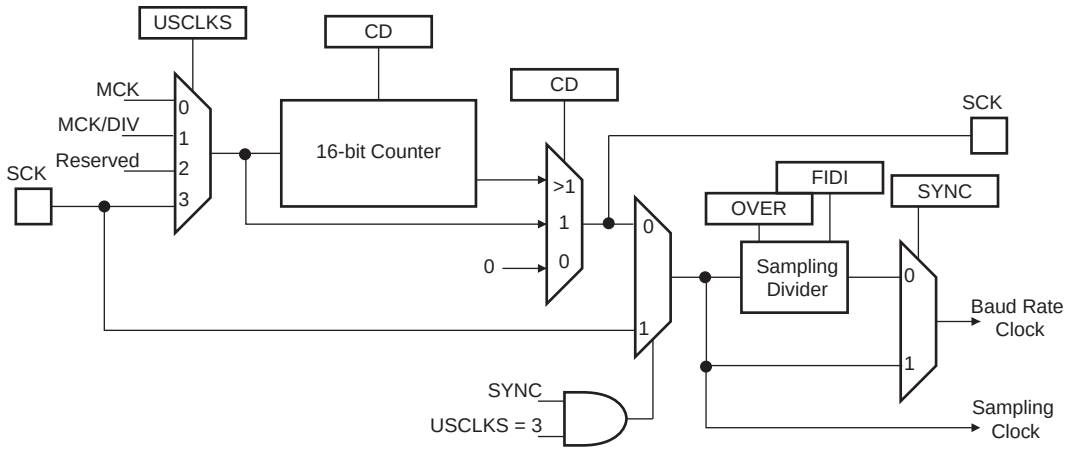
The Baud Rate Generator clock source can be selected by setting the USCLKS field in the Mode Register (US_MR) between:

- the Master Clock MCK
- a division of the Master Clock, the divider being product dependent, but generally set to 8
- the external clock, available on the SCK pin

The Baud Rate Generator is based upon a 16-bit divider, which is programmed with the CD field of the Baud Rate Generator Register (US_BRGR). If CD is programmed to 0, the Baud Rate Generator does not generate any clock. If CD is programmed to 1, the divider is bypassed and becomes inactive.

If the external SCK clock is selected, the duration of the low and high levels of the signal provided on the SCK pin must be longer than a Master Clock (MCK) period. The frequency of the signal provided on SCK must be at least 3 times lower than MCK in USART mode, or 6 in SPI mode.

**Figure 31-3. Baud Rate Generator**




#### 31.7.1.1 Baud Rate in Asynchronous Mode

If the USART is programmed to operate in asynchronous mode, the selected clock is first divided by CD, which is field programmed in the Baud Rate Generator Register (US_BRGR). The resulting clock is provided to the receiver as a sampling clock and then divided by 16 or 8, depending on the programming of the OVER bit in US_MR.

If OVER is set to 1, the receiver sampling is 8 times higher than the baud rate clock. If OVER is cleared, the sampling is performed at 16 times the baud rate clock.

The following formula performs the calculation of the Baud Rate.

$$Baudrate = \frac{SelectedClock}{(8(2-Over)CD)}$$

This gives a maximum baud rate of MCK divided by 8, assuming that MCK is the highest possible clock and that OVER is programmed to 1.

*Baud Rate Calculation Example*

Table 31-5 shows calculations of CD to obtain a baud rate at 38400 bauds for different source clock frequencies. This table also shows the actual resulting baud rate and the error.

**Table 31-5. Baud Rate Example (OVER = 0)**

| Source Clock | Expected Baud Rate | Calculation Result | CD | Actual Baud Rate | Error |
|---|---|---|---|---|---|
| MHz | Bit/s | | | Bit/s | |
| 3 686 400 | 38 400 | 6.00 | 6 | 38 400.00 | 0.00% |
| 4 915 200 | 38 400 | 8.00 | 8 | 38 400.00 | 0.00% |
| 5 000 000 | 38 400 | 8.14 | 8 | 39 062.50 | 1.70% |
| 7 372 800 | 38 400 | 12.00 | 12 | 38 400.00 | 0.00% |
| 8 000 000 | 38 400 | 13.02 | 13 | 38 461.54 | 0.16% |
| 12 000 000 | 38 400 | 19.53 | 20 | 37 500.00 | 2.40% |
| 12 288 000 | 38 400 | 20.00 | 20 | 38 400.00 | 0.00% |
| 14 318 180 | 38 400 | 23.30 | 23 | 38 908.10 | 1.31% |
| 14 745 600 | 38 400 | 24.00 | 24 | 38 400.00 | 0.00% |
| 18 432 000 | 38 400 | 30.00 | 30 | 38 400.00 | 0.00% |
| 24 000 000 | 38 400 | 39.06 | 39 | 38 461.54 | 0.16% |
| 24 576 000 | 38 400 | 40.00 | 40 | 38 400.00 | 0.00% |
| 25 000 000 | 38 400 | 40.69 | 40 | 38 109.76 | 0.76% |
| 32 000 000 | 38 400 | 52.08 | 52 | 38 461.54 | 0.16% |
| 32 768 000 | 38 400 | 53.33 | 53 | 38 641.51 | 0.63% |
| 33 000 000 | 38 400 | 53.71 | 54 | 38 194.44 | 0.54% |
| 40 000 000 | 38 400 | 65.10 | 65 | 38 461.54 | 0.16% |
| 50 000 000 | 38 400 | 81.38 | 81 | 38 580.25 | 0.47% |

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%.

$$Error = 1 - \left(\frac{ExpectedBaudRate}{ActualBaudRate}\right)$$

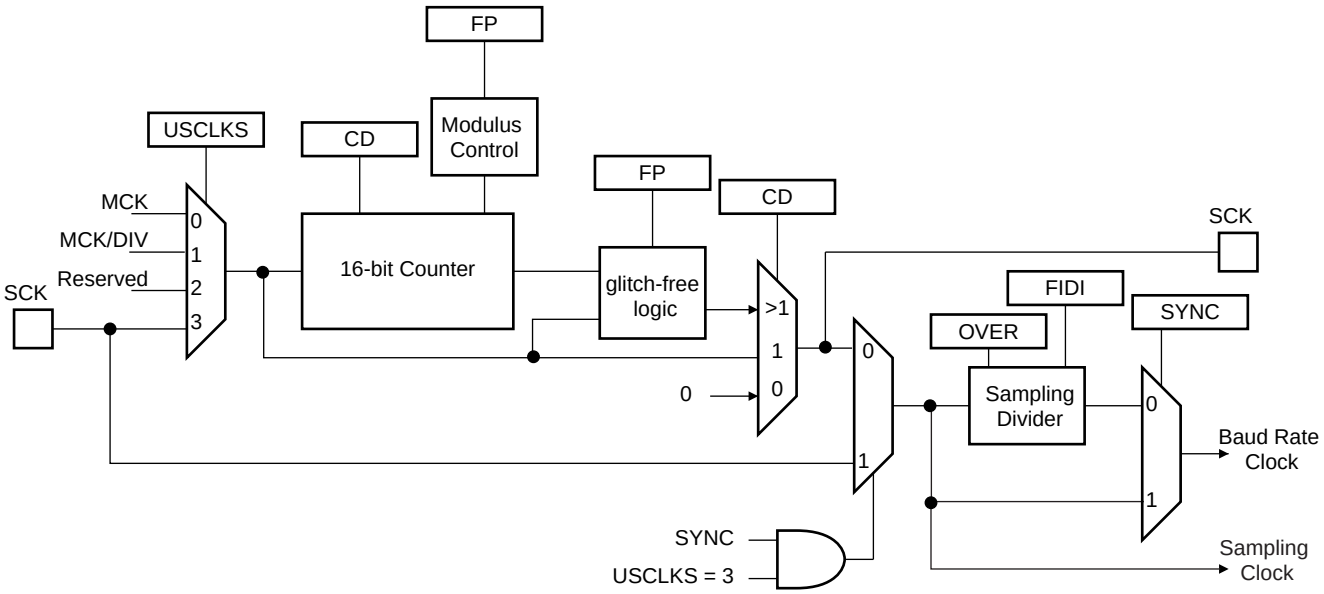
#### 31.7.1.2 Fractional Baud Rate in Asynchronous Mode

The Baud Rate generator previously defined is subject to the following limitation: the output frequency changes by only integer multiples of the reference frequency. An approach to this problem is to integrate a fractional N clock generator that has a high resolution. The generator architecture is modified to obtain Baud Rate changes by a fraction of the reference source clock. This fractional part is programmed with the FP field in the Baud Rate Generator Register (US_BRGR). If FP is not 0, the fractional part is activated. The resolution is one eighth of the clock divider. This feature is only available when using USART normal mode. The fractional Baud Rate is calculated using the following formula:

$$Baudrate = \frac{SelectedClock}{\left(8(2 - Over)\left(CD + \frac{FP}{8}\right)\right)}$$

The modified architecture is presented below:

**Figure 31-4. Fractional Baud Rate Generator**




### 31.7.1.3 Baud Rate in Synchronous Mode or SPI Mode

If the USART is programmed to operate in synchronous mode, the selected clock is simply divided by the field CD in US_BRGR.

$$BaudRate = \frac{SelectedClock}{CD}$$

In synchronous mode, if the external clock is selected (USCLKS = 3), the clock is provided directly by the signal on the USART SCK pin. No division is active. The value written in US_BRGR has no effect. The external clock frequency must be at least 3 times lower than the system clock. In synchronous mode master (USCLKS = 0 or 1, CLK0 set to 1), the receive part limits the SCK maximum frequency to MCK/3 in USART mode, or MCK/6 in SPI mode.

When either the external clock SCK or the internal clock divided (MCK/DIV) is selected, the value programmed in CD must be even if the user has to ensure a 50:50 mark/space ratio on the SCK pin. If the internal clock MCK is selected, the Baud Rate Generator ensures a 50:50 duty cycle on the SCK pin, even if the value programmed in CD is odd.

### 31.7.1.4 Baud Rate in ISO 7816 Mode

The ISO7816 specification defines the bit rate with the following formula:

$$B = \frac{Di}{Fi} \times f$$

where:

- B is the bit rate
- Di is the bit-rate adjustment factor
- Fi is the clock frequency division factor
- f is the ISO7816 clock frequency (Hz)

Di is a binary value encoded on a 4-bit field, named DI, as represented in Table 31-6.

**Table 31-6. 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 31-7.

**Table 31-7. 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 31-8 shows the resulting Fi/Di Ratio, which is the ratio between the ISO7816 clock and the baud rate clock.

**Table 31-8. Possible Values for the Fi/Di Ratio**

| Fi/Di | 372 | 558 | 774 | 1116 | 1488 | 1806 | 512 | 768 | 1024 | 1536 | 2048 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 372 | 558 | 744 | 1116 | 1488 | 1860 | 512 | 768 | 1024 | 1536 | 2048 |
| 2 | 186 | 279 | 372 | 558 | 744 | 930 | 256 | 384 | 512 | 768 | 1024 |
| 4 | 93 | 139.5 | 186 | 279 | 372 | 465 | 128 | 192 | 256 | 384 | 512 |
| 8 | 46.5 | 69.75 | 93 | 139.5 | 186 | 232.5 | 64 | 96 | 128 | 192 | 256 |
| 16 | 23.25 | 34.87 | 46.5 | 69.75 | 93 | 116.2 | 32 | 48 | 64 | 96 | 128 |
| 32 | 11.62 | 17.43 | 23.25 | 34.87 | 46.5 | 58.13 | 16 | 24 | 32 | 48 | 64 |
| 12 | 31 | 46.5 | 62 | 93 | 124 | 155 | 42.66 | 64 | 85.33 | 128 | 170.6 |
| 20 | 18.6 | 27.9 | 37.2 | 55.8 | 74.4 | 93 | 25.6 | 38.4 | 51.2 | 76.8 | 102.4 |

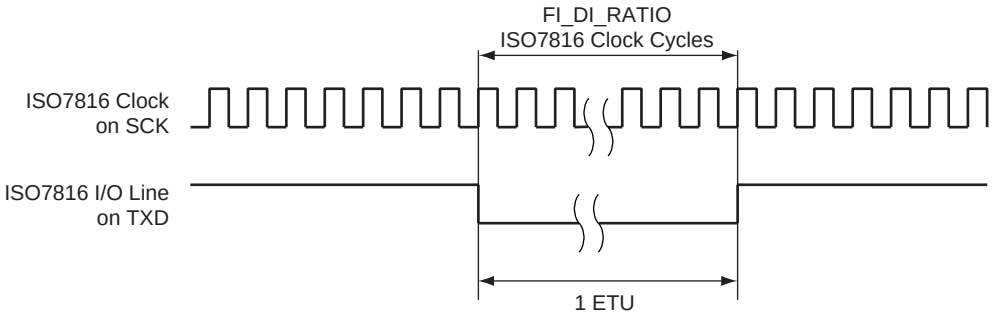
If the USART is configured in ISO7816 Mode, the clock selected by the USCLKS field in the Mode Register (US_MR) is first divided by the value programmed in the field CD in the Baud Rate Generator Register (US_BRGR). The resulting clock can be provided to the SCK pin to feed the smart card clock inputs. This means that the CLKO bit can be set in US_MR.

This clock is then divided by the value programmed in the FI_DI_RATIO field in the FI_DI_Ratio register (US_FIDI). This is performed by the Sampling Divider, which performs a division by up to 2047 in ISO7816 Mode. The non-integer values of the Fi/Di Ratio are not supported and the user must program the FI_DI_RATIO field to a value as close as possible to the expected value.

The FI_DI_RATIO field resets to the value 0x174 (372 in decimal) and is the most common divider between the ISO7816 clock and the bit rate (Fi = 372, Di = 1).

Figure 31-5 shows the relation between the Elementary Time Unit, corresponding to a bit time, and the ISO 7816 clock.

**Figure 31-5. Elementary Time Unit (ETU)**




### 31.7.2 Receiver and Transmitter Control

After reset, the receiver is disabled. The user must enable the receiver by setting the RXEN bit in the Control Register (US_CR). However, the receiver registers can be programmed before the receiver clock is enabled.

After reset, the transmitter is disabled. The user must enable it by setting the TXEN bit in the Control Register (US_CR). However, the transmitter registers can be programmed before being enabled.

The Receiver and the Transmitter can be enabled together or independently.

At any time, the software can perform a reset on the receiver or the transmitter of the USART by setting the corresponding bit, RSTRX and RSTTX respectively, in the Control Register (US_CR). The software resets clear the status flag and reset internal state machines but the user interface configuration registers hold the value configured prior to software reset. Regardless of what the receiver or the transmitter is performing, the communication is immediately stopped.

The user can also independently disable the receiver or the transmitter by setting RXDIS and TXDIS respectively in US_CR. If the receiver is disabled during a character reception, the USART waits until the end of reception of the current character, then the reception is stopped. If the transmitter is disabled while it is operating, the USART waits the end of transmission of both the current character and character being stored in the Transmit Holding Register (US_THR). If a timeguard is programmed, it is handled normally.

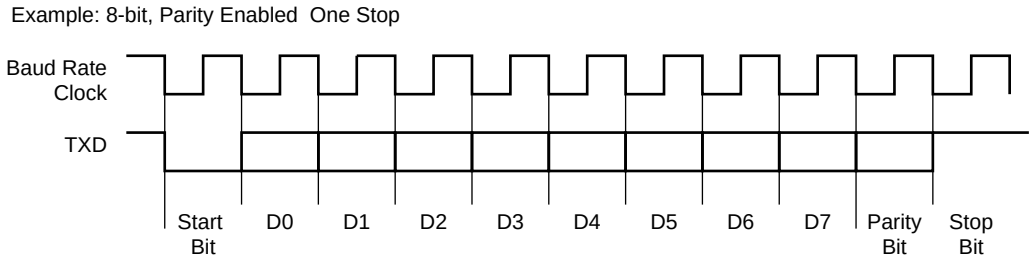
### 31.7.3 Synchronous and Asynchronous Modes

#### 31.7.3.1 Transmitter Operations

The transmitter performs the same in both synchronous and asynchronous operating modes (SYNC = 0 or SYNC = 1). One start bit, up to 9 data bits, one optional parity bit and up to two stop bits are successively shifted out on the TXD pin at each falling edge of the programmed serial clock.

The number of data bits is selected by the CHRL field and the MODE 9 bit in the Mode Register (US_MR). Nine bits are selected by setting the MODE 9 bit regardless of the CHRL field. The parity bit is set according to the PAR field in US_MR. The even, odd, space, marked or none parity bit can be configured. The MSBF field in US_MR configures which data bit is sent first. If written to 1, the most significant bit is sent first. If written to 0, the less significant bit is sent first. The number of stop bits is selected by the NBSTOP field in US_MR. The 1.5 stop bit is supported in asynchronous mode only.

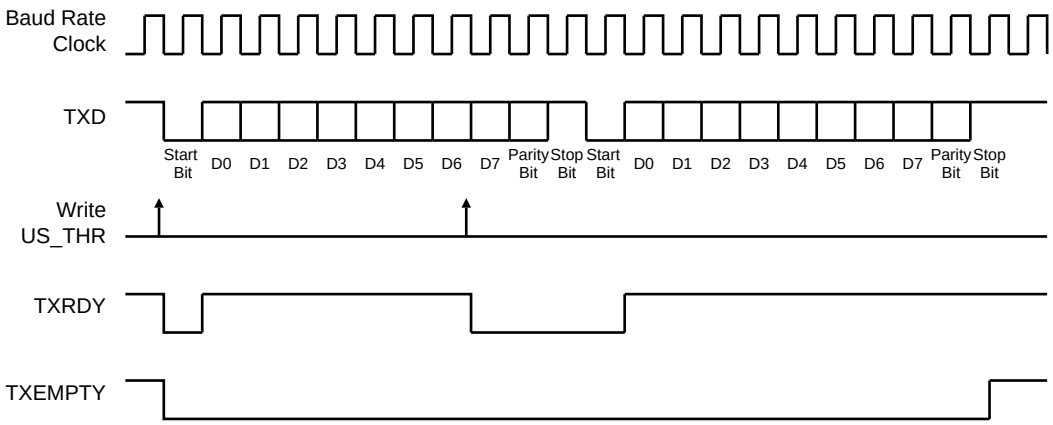
**Figure 31-6. Character Transmit**




The characters are sent by writing in the Transmit Holding Register (US_THR). The transmitter reports two status bits in the Channel Status Register (US_CSR): TXRDY (Transmitter Ready), which indicates that US_THR is empty and TXEMPTY, which indicates that all the characters written in US_THR have been processed. When the current character processing is completed, the last character written in US_THR is transferred into the Shift Register of the transmitter and US_THR becomes empty, thus TXRDY rises.

Both TXRDY and TXEMPTY bits are low when the transmitter is disabled. Writing a character in US_THR while TXRDY is low has no effect and the written character is lost.

**Figure 31-7. Transmitter Status**




### 31.7.3.2 Asynchronous Receiver

If the USART is programmed in asynchronous operating mode (SYNC = 0), the receiver oversamples the RXD input line. The oversampling is either 16 or 8 times the Baud Rate clock, depending on the OVER bit in the Mode Register (US_MR).

The receiver samples the RXD line. If the line is sampled during one half of a bit time to 0, a start bit is detected and data, parity and stop bits are successively sampled on the bit rate clock.

If the oversampling is 16, (OVER to 0), a start is detected at the eighth sample to 0. Then, data bits, parity bit and stop bit are sampled on each 16 sampling clock cycle. If the oversampling is 8 (OVER to 1), a start bit is detected at the fourth sample to 0. Then, data bits, parity bit and stop bit are sampled on each 8 sampling clock cycle.

The number of data bits, first bit sent and parity mode are selected by the same fields and bits as the transmitter, i.e. respectively CHRL, MODE9, MSBF and PAR. For the synchronization mechanism **only**, the number of stop bits has no effect on the receiver as it considers only one stop bit, regardless of the field NBSTOP, so that resynchronization between the receiver and the transmitter can occur. Moreover, as soon as the stop bit is sampled, the receiver starts looking for a new start bit so that resynchronization can also be accomplished when the transmitter is operating with one stop bit.

Figure 31-8 and Figure 31-9 illustrate start detection and character reception when USART operates in asynchronous mode.

**Figure 31-8. Asynchronous Start Detection**

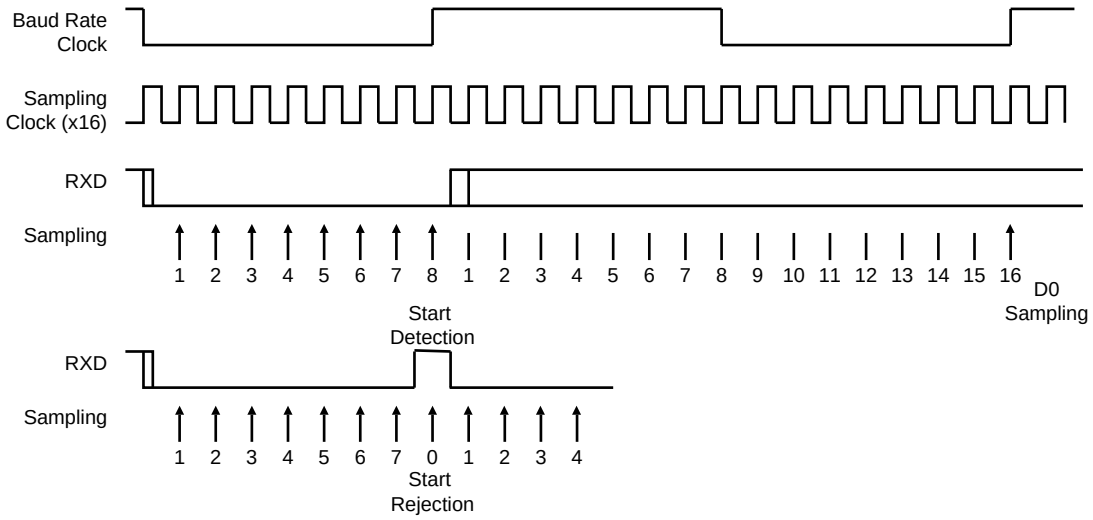
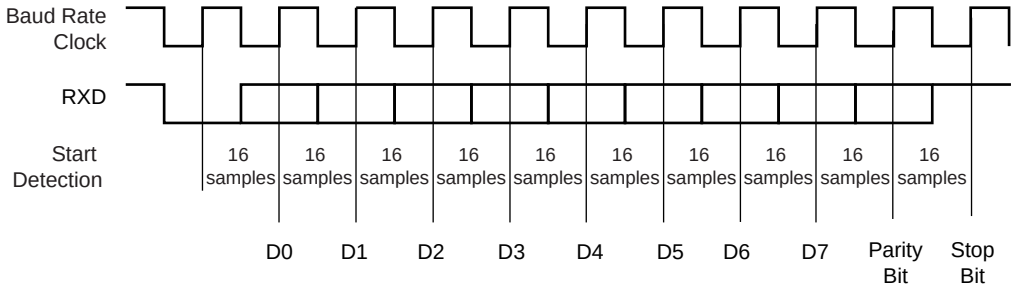



**Figure 31-9. Asynchronous Character Reception**

Example: 8-bit, Parity Enabled




### 31.7.3.3 Synchronous Receiver

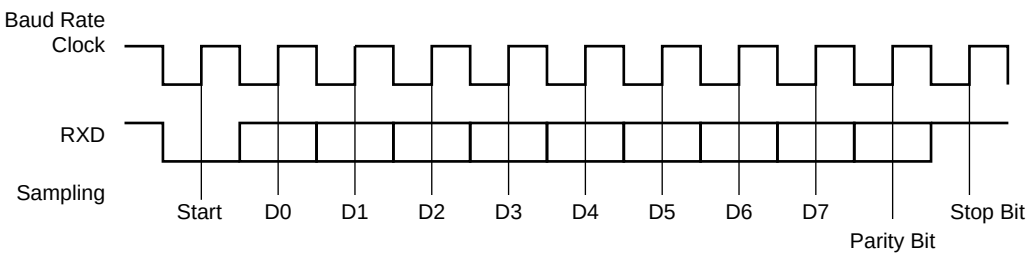
In synchronous mode (SYNC = 1), the receiver samples the RXD signal on each rising edge of the Baud Rate Clock. If a low level is detected, it is considered as a start. All data bits, the parity bit and the stop bits are sampled and the receiver waits for the next start bit. Synchronous mode operations provide a high speed transfer capability.

Configuration fields and bits are the same as in asynchronous mode.

Figure 31-10 illustrates a character reception in synchronous mode.

**Figure 31-10. Synchronous Mode Character Reception**

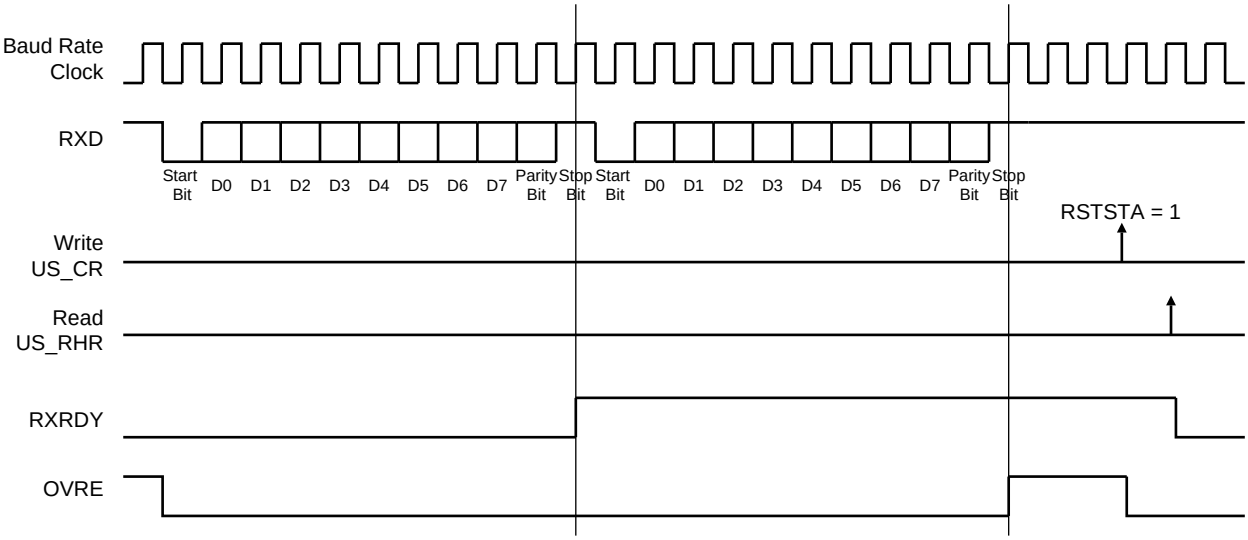
Example: 8-bit, Parity Enabled 1 Stop



### 31.7.3.4 Receiver Operations

When a character reception is completed, it is transferred to the Receive Holding Register (US_RHR) and the RXRDY bit in the Status Register (US_CSR) rises. If a character is completed while the RXRDY is set, the OVRE (Overrun Error) bit is set. The last character is transferred into US_RHR and overwrites the previous one. The OVRE bit is cleared by writing the Control Register (US_CR) with the RSTSTA (Reset Status) bit to 1.

**Figure 31-11. Receiver Status**




### 31.7.3.5 Parity

The USART supports five parity modes selected by programming the PAR field in the Mode Register (US_MR). The PAR field also enables the Multidrop mode, see “Multidrop Mode” on page 546. 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.

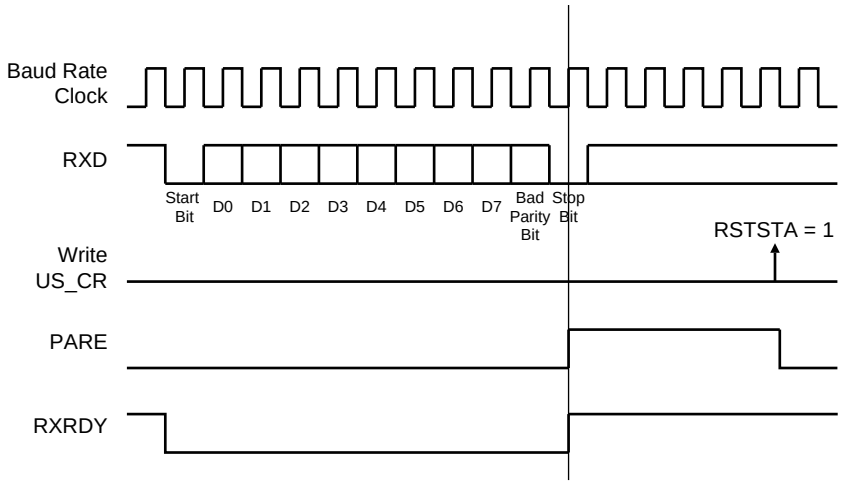
Table 31-9 shows an example of the parity bit for the character 0x41 (character ASCII “A”) depending on the configuration of the USART. Because there are two bits to 1, 1 bit is added when a parity is odd, or 0 is added when a parity is even.

**Table 31-9. Parity Bit Examples**

| Character | Hexa | Binary | Parity Bit | Parity Mode |
|---|---|---|---|---|
| A | 0x41 | 0100 0001 | 1 | Odd |
| A | 0x41 | 0100 0001 | 0 | Even |
| A | 0x41 | 0100 0001 | 1 | Mark |
| A | 0x41 | 0100 0001 | 0 | Space |
| A | 0x41 | 0100 0001 | None | None |

When the receiver detects a parity error, it sets the PARE (Parity Error) bit in the Channel Status Register (US_CSR). The PARE bit can be cleared by writing the Control Register (US_CR) with the RSTSTA bit to 1. Figure 31-12 illustrates the parity bit status setting and clearing.

**Figure 31-12. Parity Error**




### 31.7.3.6 Multidrop Mode

If the PAR field in the Mode Register (US_MR) is programmed to the value 0x6 or 0x07, the USART runs in Multidrop Mode. This mode differentiates the data characters and the address characters. Data is transmitted with the parity bit to 0 and addresses are transmitted with the parity bit to 1.

If the USART is configured in multidrop mode, the receiver sets the PARE parity error bit when the parity bit is high and the transmitter is able to send a character with the parity bit high when the Control Register is written with the SENDA bit to 1.

To handle parity error, the PARE bit is cleared when the Control Register is written with the bit RSTSTA to 1.

The transmitter sends an address byte (parity bit set) when SENDA is written to US_CR. In this case, the next byte written to US_THR is transmitted as an address. Any character written in US_THR without having written the command SENDA is transmitted normally with the parity to 0.

### 31.7.3.7 Transmitter Timeguard

The timeguard feature enables the USART interface with slow remote devices.

The timeguard function enables the transmitter to insert an idle state on the TXD line between two characters. This idle state actually acts as a long stop bit.

The duration of the idle state is programmed in the TG field of the Transmitter Timeguard Register (US_TTGR). When this field is programmed to zero no timeguard is generated. Otherwise, the transmitter holds a high level on TXD after each transmitted byte during the number of bit periods programmed in TG in addition to the number of stop bits.

As illustrated in Figure 31-13, the behavior of TXRDY and TXEMPTY status bits is modified by the programming of a timeguard. TXRDY rises only when the start bit of the next character is sent, and thus remains to 0 during the timeguard transmission if a character has been written in US_THR. TXEMPTY remains low until the timeguard transmission is completed as the timeguard is part of the current character being transmitted.

**Figure 31-13. Timeguard Operations**

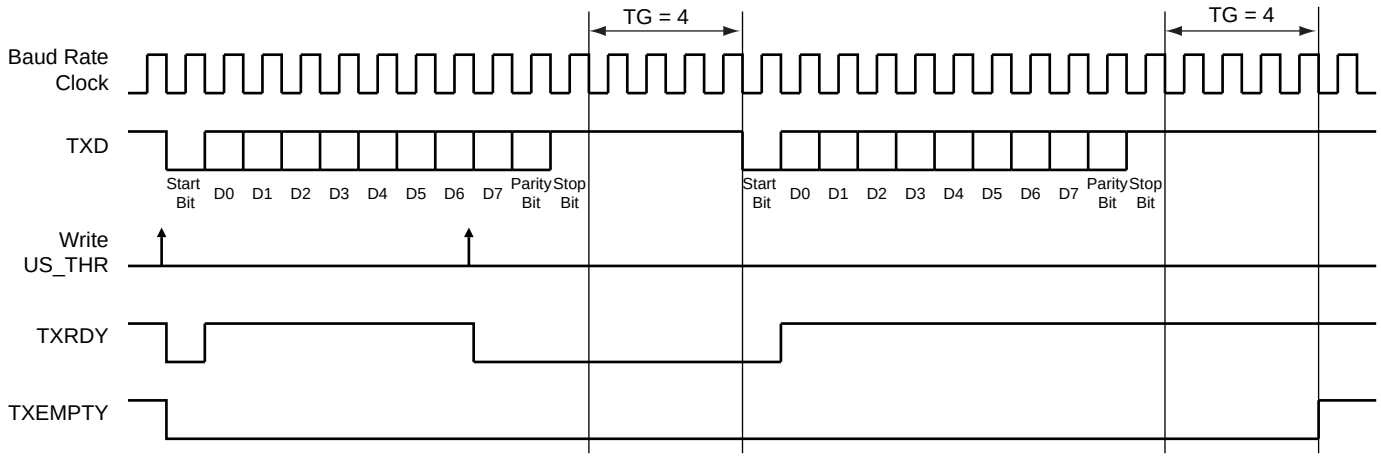



Table 31-10 indicates the maximum length of a timeguard period that the transmitter can handle in relation to the function of the Baud Rate.

**Table 31-10. Maximum Timeguard Length Depending on Baud Rate**

| Baud Rate | Bit time | Timeguard |
|---|---|---|
| Bit/sec | µs | ms |
| 1 200 | 833 | 212.50 |
| 9 600 | 104 | 26.56 |
| 14400 | 69.4 | 17.71 |
| 19200 | 52.1 | 13.28 |
| 28800 | 34.7 | 8.85 |
| 33400 | 29.9 | 7.63 |
| 56000 | 17.9 | 4.55 |
| 57600 | 17.4 | 4.43 |
| 115200 | 8.7 | 2.21 |

#### 31.7.3.8 Receiver Time-out

The Receiver Time-out provides support in handling variable-length frames. This feature detects an idle condition on the RXD line. When a time-out is detected, the bit TIMEOUT in the Channel Status Register (US_CSR) rises and can generate an interrupt, thus indicating to the driver an end of frame.

The time-out delay period (during which the receiver waits for a new character) is programmed in the TO field of the Receiver Time-out Register (US_RTOR). If the TO field is programmed to 0, the Receiver Time-out is disabled and no time-out is detected. The TIMEOUT bit in US_CSR remains to 0. Otherwise, the receiver loads a 16-bit counter with the value programmed in TO. This counter is decremented at each bit period and reloaded each time a new character is received. If the counter reaches 0, the TIMEOUT bit in the Status Register rises. Then, the user can either:

- Stop the counter clock until a new character is received. This is performed by writing the Control Register (US_CR) with the STTTO (Start Time-out) bit to 1. In this case, the idle state on RXD before a new character is received will not provide a time-out. This prevents having to handle an interrupt before a character is received and allows waiting for the next idle state on RXD after a frame is received.
- Obtain an interrupt while no character is received. This is performed by writing US_CR with the RETTO (Reload and Start Time-out) bit to 1. If RETTO is performed, the counter starts counting down immediately

from the value TO. This enables generation of a periodic interrupt so that a user time-out can be handled, for example when no key is pressed on a keyboard.

If STTTO is performed, the counter clock is stopped until a first character is received. The idle state on RXD before the start of the frame does not provide a time-out. This prevents having to obtain a periodic interrupt and enables a wait of the end of frame when the idle state on RXD is detected.

If RETTO is performed, the counter starts counting down immediately from the value TO. This enables generation of a periodic interrupt so that a user time-out can be handled, for example when no key is pressed on a keyboard.

Figure 31-14 shows the block diagram of the Receiver Time-out feature.

**Figure 31-14. Receiver Time-out Block Diagram**

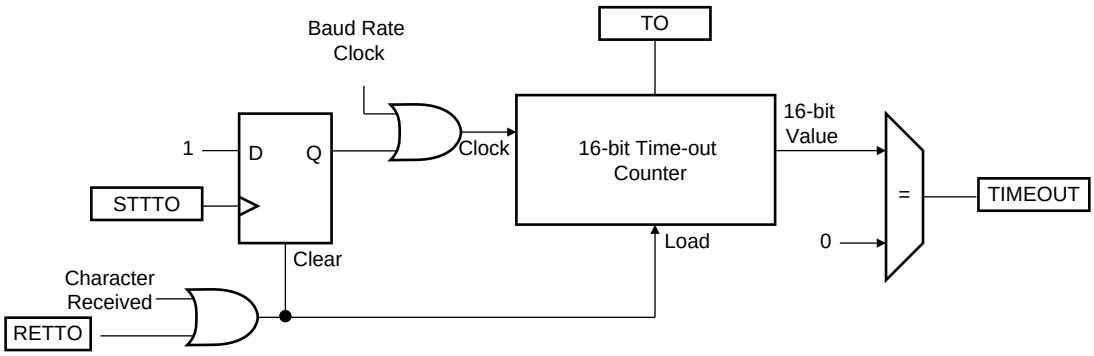



Table 31-11 gives the maximum time-out period for some standard baud rates.

**Table 31-11. Maximum Time-out Period**

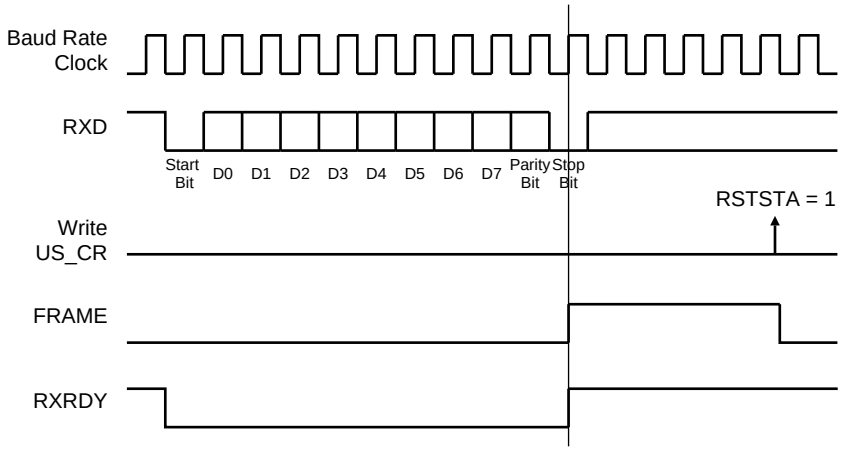
| Baud Rate | Bit Time | Time-out |
|---|---|---|
| bit/sec | µs | ms |
| 600 | 1 667 | 109 225 |
| 1 200 | 833 | 54 613 |
| 2 400 | 417 | 27 306 |
| 4 800 | 208 | 13 653 |
| 9 600 | 104 | 6 827 |
| 14400 | 69 | 4 551 |
| 19200 | 52 | 3 413 |
| 28800 | 35 | 2 276 |
| 33400 | 30 | 1 962 |
| 56000 | 18 | 1 170 |
| 57600 | 17 | 1 138 |
| 200000 | 5 | 328 |

#### 31.7.3.9 Framing Error

The receiver is capable of detecting framing errors. A framing error happens when the stop bit of a received character is detected at level 0. This can occur if the receiver and the transmitter are fully desynchronized.

A framing error is reported on the FRAME bit of the Channel Status Register (US_CSR). The FRAME bit is asserted in the middle of the stop bit as soon as the framing error is detected. It is cleared by writing the Control Register (US_CR) with the RSTSTA bit to 1.

**Figure 31-15. Framing Error Status**




### 31.7.3.10 Transmit Break

The user can request the transmitter to generate a break condition on the TXD line. A break condition drives the TXD line low during at least one complete character. It appears the same as a 0x00 character sent with the parity and the stop bits to 0. However, the transmitter holds the TXD line at least during one character until the user requests the break condition to be removed.

A break is transmitted by writing the Control Register (US_CR) with the STTBRK bit to 1. This can be performed at any time, either while the transmitter is empty (no character in either the Shift Register or in US_THR) or when a character is being transmitted. If a break is requested while a character is being shifted out, the character is first completed before the TXD line is held low.

Once STTBRK command is requested further STTBRK commands are ignored until the end of the break is completed.

The break condition is removed by writing US_CR with the STPBRK bit to 1. If the STPBRK is requested before the end of the minimum break duration (one character, including start, data, parity and stop bits), the transmitter ensures that the break condition completes.

The transmitter considers the break as though it is a character, i.e. the STTBRK and STPBRK commands are taken into account only if the TXRDY bit in US_CSR is to 1 and the start of the break condition clears the TXRDY and TXEMPTY bits as if a character is processed.

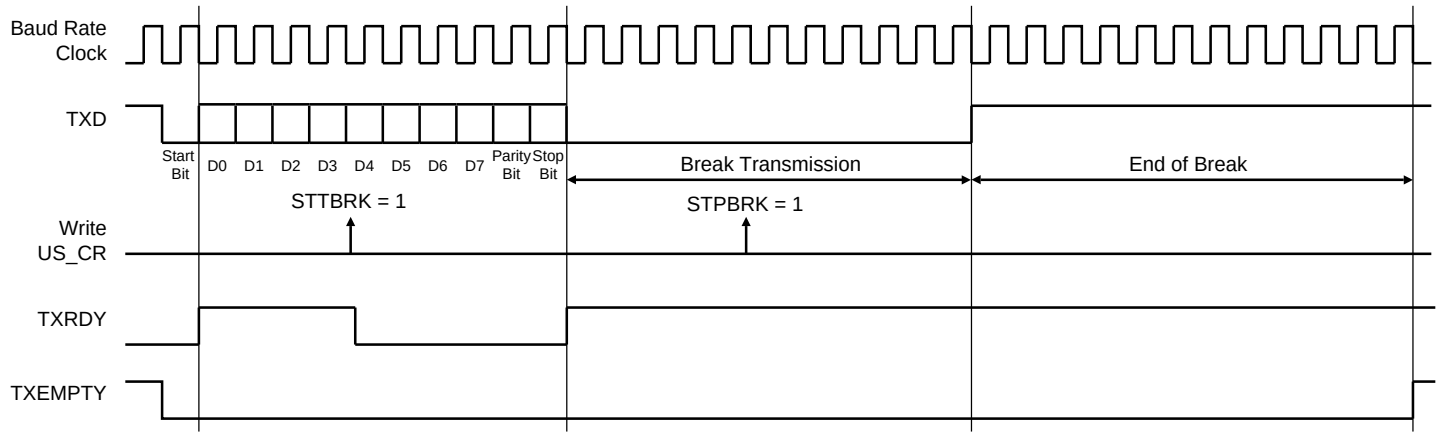
Writing US_CR with both STTBRK and STPBRK bits to 1 can lead to an unpredictable result. All STPBRK commands requested without a previous STTBRK command are ignored. A byte written into the Transmit Holding Register while a break is pending, but not started, is ignored.

After the break condition, the transmitter returns the TXD line to 1 for a minimum of 12 bit times. Thus, the transmitter ensures that the remote receiver detects correctly the end of break and the start of the next character. If the timeguard is programmed with a value higher than 12, the TXD line is held high for the timeguard period.

After holding the TXD line for this period, the transmitter resumes normal operations.

Figure 31-16 illustrates the effect of both the Start Break (STTBRK) and Stop Break (STPBRK) commands on the TXD line.

**Figure 31-16. Break Transmission**




### 31.7.3.11 Receive Break

The receiver detects a break condition when all data, parity and stop bits are low. This corresponds to detecting a framing error with data to 0x00, but FRAME remains low.

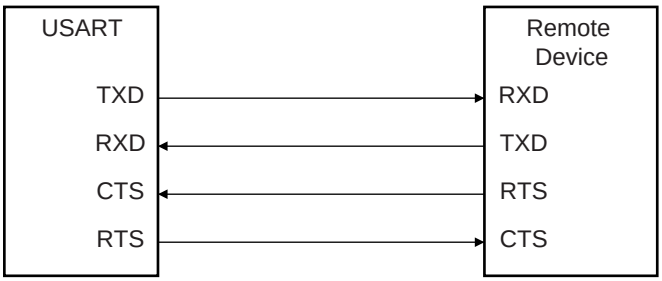
When the low stop bit is detected, the receiver asserts the RXBRK bit in US_CSR. This bit may be cleared by writing the Control Register (US_CR) with the bit RSTSTA to 1.

An end of receive break is detected by a high level for at least 2/16 of a bit period in asynchronous operating mode or one sample at high level in synchronous operating mode. The end of break detection also asserts the RXBRK bit.

### 31.7.3.12 Hardware Handshaking

The USART features a hardware handshaking out-of-band flow control. The RTS and CTS pins are used to connect with the remote device, as shown in Figure 31-17.

**Figure 31-17. Connection with a Remote Device for Hardware Handshaking**




Setting the USART to operate with hardware handshaking is performed by writing the USART_MODE field in the Mode Register (US_MR) to the value 0x2.

The USART behavior when hardware handshaking is enabled is the same as the behavior in standard synchronous or asynchronous mode, except that the receiver drives the RTS pin as described below and the level on the CTS pin modifies the behavior of the transmitter as described below. Using this mode requires using the PDC channel for reception. The transmitter can handle hardware handshaking in any case.

Figure 31-18 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 31-18. Receiver Behavior when Operating with Hardware Handshaking**

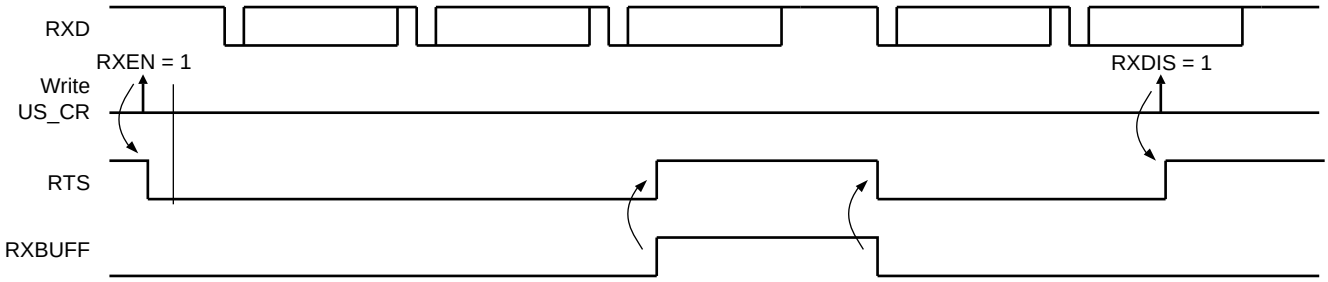
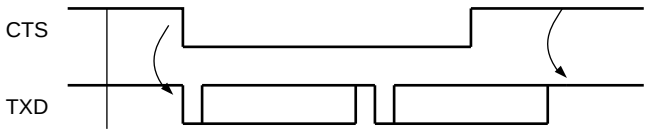



Figure 31-19 shows how the transmitter operates if hardware handshaking is enabled. The CTS pin disables the transmitter. If a character is being processing, the transmitter is disabled only after the completion of the current character and transmission of the next character happens as soon as the pin CTS falls.

**Figure 31-19. Transmitter Behavior when Operating with Hardware Handshaking**




### 31.7.4 ISO7816 Mode

The USART features an ISO7816-compatible operating mode (Only on USART0). This mode permits interfacing with smart cards and Security Access Modules (SAM) communicating through an ISO7816 link. Both T = 0 and T = 1 protocols defined by the ISO7816 specification are supported.

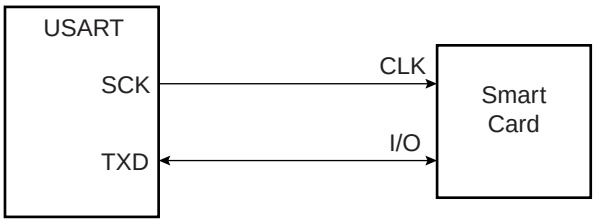
Setting the USART in ISO7816 mode is performed by writing the USART_MODE field in the Mode Register (US_MR) to the value 0x4 for protocol T = 0 and to the value 0x5 for protocol T = 1.

#### 31.7.4.1 ISO7816 Mode Overview

The ISO7816 is a half duplex communication on only one bidirectional line. The baud rate is determined by a division of the clock provided to the remote device (see “Baud Rate Generator” on page 538).

The USART connects to a smart card as shown in Figure 31-20. 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 31-20. Connection of a Smart Card to the USART**




When operating in ISO7816, either in T = 0 or T = 1 modes, the character format is fixed. The configuration is 8 data bits, even parity and 1 or 2 stop bits, regardless of the values programmed in the CHRL, MODE9, PAR and CHMODE fields. MSBF can be used to transmit LSB or MSB first. Parity Bit (PAR) can be used to transmit in normal or inverse mode. Refer to “USART Mode Register” on page 567 and “PAR: Parity Type” on page 568.

The USART cannot operate concurrently in both receiver and transmitter modes as the communication is unidirectional at a time. It has to be configured according to the required mode by enabling or disabling either the receiver or the transmitter as desired. Enabling both the receiver and the transmitter at the same time in ISO7816 mode may lead to unpredictable results.

The ISO7816 specification defines an inverse transmission format. Data bits of the character must be transmitted on the I/O line at their negative value. The USART does not support this format and the user has to perform an exclusive OR on the data before writing it in the Transmit Holding Register (US_THR) or after reading it in the Receive Holding Register (US_RHR).

#### 31.7.4.2 Protocol T = 0

In T = 0 protocol, a character is made up of one start bit, eight data bits, one parity bit and one guard time, which lasts two bit times. The transmitter shifts out the bits and does not drive the I/O line during the guard time.

If no parity error is detected, the I/O line remains to 1 during the guard time and the transmitter can continue with the transmission of the next character, as shown in Figure 31-21.

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 31-22. This error bit is also named NACK, for Non Acknowledge. In this case, the character lasts 1 bit time more, as the guard time length is the same and is added to the error bit time which lasts 1 bit time.

When the USART is the receiver and it detects an error, it does not load the erroneous character in the Receive Holding Register (US_RHR). It appropriately sets the PARE bit in the Status Register (US_SR) so that the software can handle the error.

**Figure 31-21. T = 0 Protocol without Parity Error**

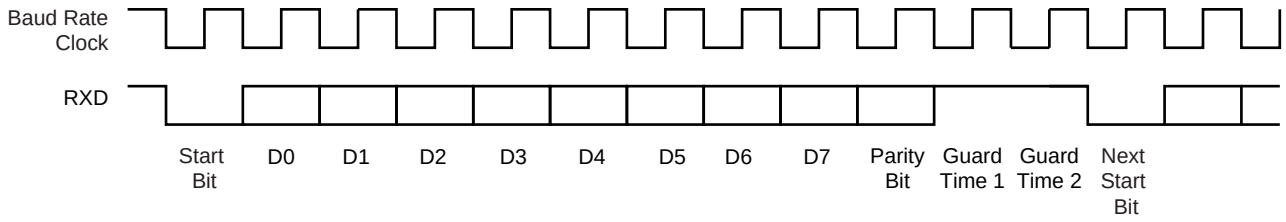
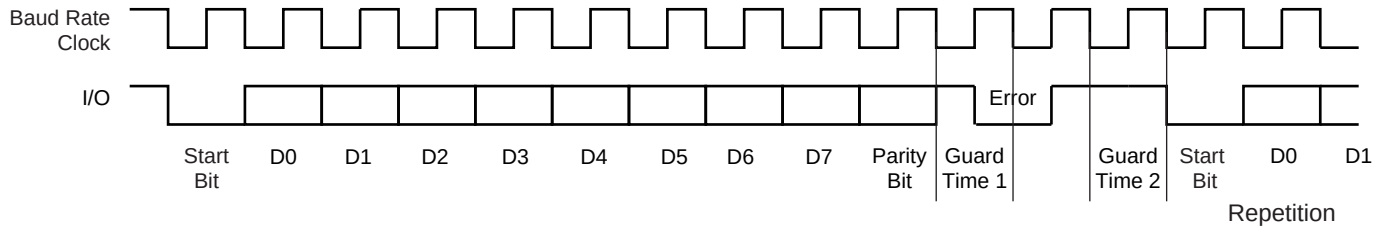



**Figure 31-22. T = 0 Protocol with Parity Error**




*Receive Error Counter*

The USART receiver also records the total number of errors. This can be read in the Number of Error (US_NER) register. The NB_ERRORS field can record up to 255 errors. Reading US_NER automatically clears the NB_ERRORS field.

*Receive NACK Inhibit*

The USART can also be configured to inhibit an error. This can be achieved by setting the INACK bit in the Mode Register (US_MR). If INACK is to 1, no error signal is driven on the I/O line even if a parity bit is detected.

Moreover, if INACK is set, the erroneous received character is stored in the Receive Holding Register, as if no error occurred and the RXRDY bit does rise.

*Transmit Character Repetition*

When the USART is transmitting a character and gets a NACK, it can automatically repeat the character before moving on to the next one. Repetition is enabled by writing the MAX_ITERATION field in the Mode Register (US_MR) at a value higher than 0. Each character can be transmitted up to eight times; the first transmission plus seven repetitions.

If MAX_ITERATION does not equal zero, the USART repeats the character as many times as the value loaded in MAX_ITERATION.

When the USART repetition number reaches MAX_ITERATION, the ITERATION bit is set in the Channel Status Register (US_CSR). If the repetition of the character is acknowledged by the receiver, the repetitions are stopped and the iteration counter is cleared.

The ITERATION bit in US_CSR can be cleared by writing the Control Register with the RSIT bit to 1.

*Disable Successive Receive NACK*

The receiver can limit the number of successive NACKs sent back to the remote transmitter. This is programmed by setting the bit DSNACK in the Mode Register (US_MR). The maximum number of NACK transmitted is programmed in the MAX_ITERATION field. As soon as MAX_ITERATION is reached, the character is considered as correct, an acknowledge is sent on the line and the ITERATION bit in the Channel Status Register is set.

#### 31.7.4.3 Protocol T = 1

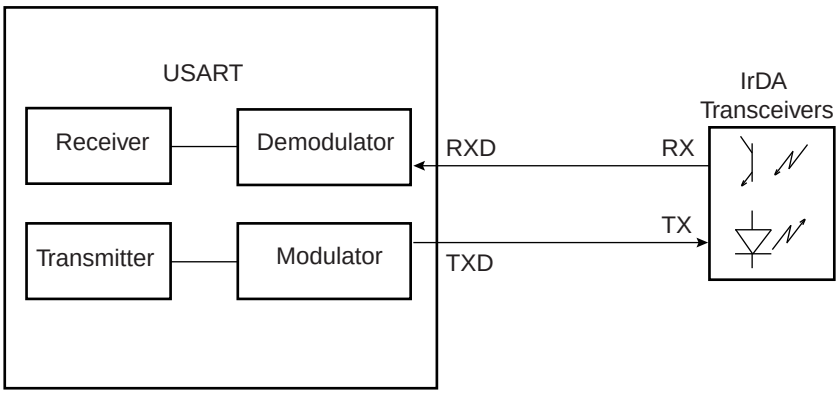
When operating in ISO7816 protocol T = 1, the transmission is similar to an asynchronous format with only one stop bit. The parity is generated when transmitting and checked when receiving. Parity error detection sets the PARE bit in the Channel Status Register (US_CSR).

### 31.7.5 IrDA Mode

The USART features an IrDA mode (Only on USART0) 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 31-23. The modulator and demodulator are compliant with the IrDA specification version 1.1 and support data transfer speeds ranging from 2.4 Kb/s to 115.2 Kb/s.

The USART IrDA mode is enabled by setting the USART_MODE field in the Mode Register (US_MR) to the value 0x8. The IrDA Filter Register (US_IF) allows configuring the demodulator filter. The USART transmitter and receiver operate in a normal asynchronous mode and all parameters are accessible. Note that the modulator and the demodulator are activated.

**Figure 31-23. Connection to IrDA Transceivers**




The receiver and the transmitter must be enabled or disabled according to the direction of the transmission to be managed.

To receive IrDA signals, the following needs to be done:

- Disable TX and Enable RX
- Configure the TXD pin as PIO and set it as an output to 0 (to avoid LED emission). Disable the internal pull-up (better for power consumption).
- Receive data

#### 31.7.5.1 IrDA Modulation

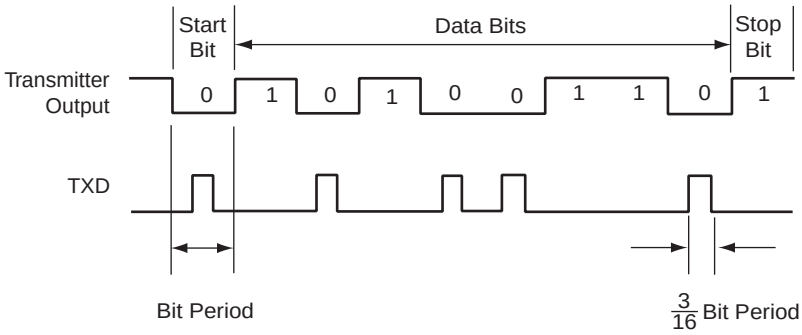
For baud rates up to and including 115.2 Kbits/sec, the RZI modulation scheme is used. “0” is represented by a light pulse of 3/16th of a bit time. Some examples of signal pulse duration are shown in Table 31-12.

**Table 31-12. IrDA Pulse Duration**

| Baud Rate | Pulse Duration (3/16) |
|---|---|
| 2.4 Kb/s | 78.13 µs |
| 9.6 Kb/s | 19.53 µs |
| 19.2 Kb/s | 9.77 µs |
| 38.4 Kb/s | 4.88 µs |
| 57.6 Kb/s | 3.26 µs |
| 115.2 Kb/s | 1.63 µs |

Figure 31-24 shows an example of character transmission.

**Figure 31-24. IrDA Modulation**



#### 31.7.5.2 IrDA Baud Rate

Table 31-13 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 31-13. IrDA Baud Rate Error**

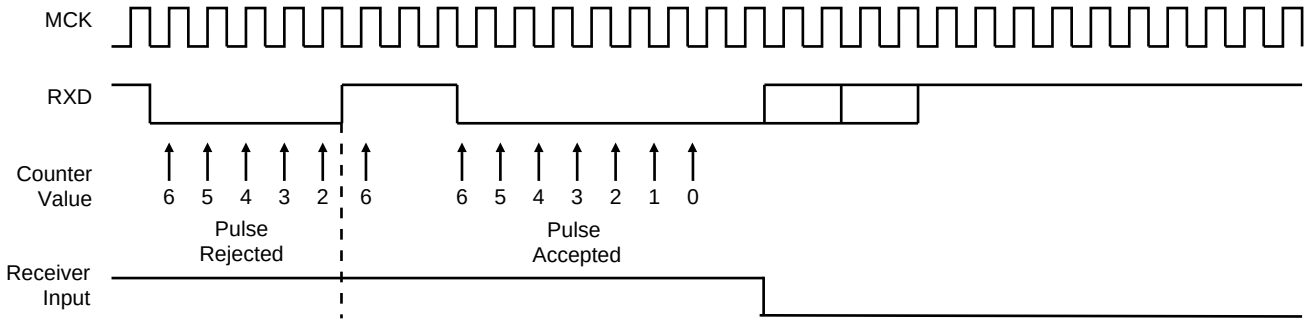
| Peripheral Clock | Baud Rate | CD | Baud Rate Error | Pulse Time |
|---|---|---|---|---|
| 3 686 400 | 115 200 | 2 | 0.00% | 1.63 |
| 20 000 000 | 115 200 | 11 | 1.38% | 1.63 |
| 32 768 000 | 115 200 | 18 | 1.25% | 1.63 |
| 40 000 000 | 115 200 | 22 | 1.38% | 1.63 |
| 3 686 400 | 57 600 | 4 | 0.00% | 3.26 |
| 20 000 000 | 57 600 | 22 | 1.38% | 3.26 |
| 32 768 000 | 57 600 | 36 | 1.25% | 3.26 |
| 40 000 000 | 57 600 | 43 | 0.93% | 3.26 |
| 3 686 400 | 38 400 | 6 | 0.00% | 4.88 |
| 20 000 000 | 38 400 | 33 | 1.38% | 4.88 |
| 32 768 000 | 38 400 | 53 | 0.63% | 4.88 |
| 40 000 000 | 38 400 | 65 | 0.16% | 4.88 |
| 3 686 400 | 19 200 | 12 | 0.00% | 9.77 |
| 20 000 000 | 19 200 | 65 | 0.16% | 9.77 |
| 32 768 000 | 19 200 | 107 | 0.31% | 9.77 |
| 40 000 000 | 19 200 | 130 | 0.16% | 9.77 |
| 3 686 400 | 9 600 | 24 | 0.00% | 19.53 |
| 20 000 000 | 9 600 | 130 | 0.16% | 19.53 |
| 32 768 000 | 9 600 | 213 | 0.16% | 19.53 |
| 40 000 000 | 9 600 | 260 | 0.16% | 19.53 |
| 3 686 400 | 2 400 | 96 | 0.00% | 78.13 |
| 20 000 000 | 2 400 | 521 | 0.03% | 78.13 |
| 32 768 000 | 2 400 | 853 | 0.04% | 78.13 |

#### 31.7.5.3 IrDA Demodulator

The demodulator is based on the IrDA Receive filter comprised of an 8-bit down counter which is loaded with the value programmed in US_IF. When a falling edge is detected on the RXD pin, the Filter Counter starts counting down at the Master Clock (MCK) speed. If a rising edge is detected on the RXD pin, the counter stops and is reloaded with US_IF. If no rising edge is detected when the counter reaches 0, the input of the receiver is driven low during one bit time.

Figure 31-25 illustrates the operations of the IrDA demodulator.

**Figure 31-25. IrDA Demodulator Operations**

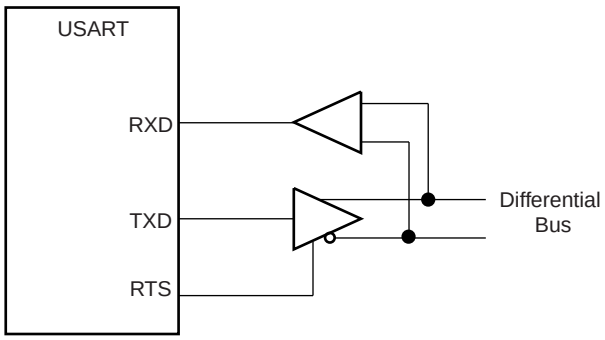



As the IrDA mode uses the same logic as the ISO7816, note that the FI_DI_RATIO field in US_FIDI must be set to a value higher than 0 in order to assure IrDA communications operate correctly.

### 31.7.6 RS485 Mode

The USART features the RS485 mode to enable line driver control. While operating in RS485 mode, the USART behaves as though in asynchronous or synchronous mode and configuration of all the parameters is possible. The difference is that the RTS pin is driven high when the transmitter is operating. The behavior of the RTS pin is controlled by the TXEMPTY bit. A typical connection of the USART to a RS485 bus is shown in Figure 31-26.

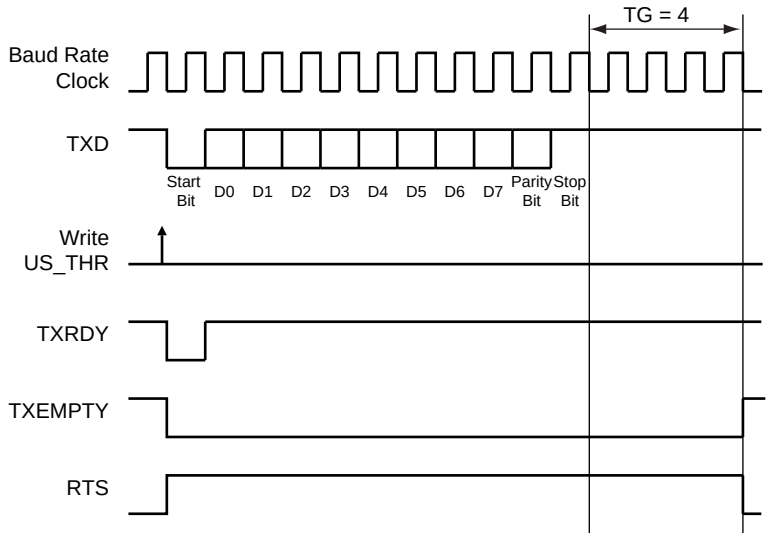
**Figure 31-26. Typical Connection to a RS485 Bus**




The USART is set in RS485 mode by programming the USART_MODE field in the Mode Register (US_MR) to the value 0x1.

The RTS pin is at a level inverse to the TXEMPTY bit. Significantly, the RTS pin remains high when a timeguard is programmed so that the line can remain driven after the last character completion. Figure 31-27 gives an example of the RTS waveform during a character transmission when the timeguard is enabled.

**Figure 31-27. Example of RTS Drive with Timeguard**




### 31.7.7 SPI Mode

The Serial Peripheral Interface (SPI) Mode is a synchronous serial data link that provides communication with external devices in Master or Slave Mode. It also enables communication between processors if an external processor is connected to the system.

The Serial Peripheral Interface is essentially a shift register that serially transmits data bits to other SPIs. During a data transfer, one SPI system acts as the “master” which controls the data flow, while the other devices act as “slaves" which have data shifted into and out by the master. Different CPUs can take turns being masters and one master may simultaneously shift data into multiple slaves. (Multiple Master Protocol is the opposite of Single Master Protocol, where one CPU is always the master while all of the others are always slaves.) However, only one slave may drive its output to write data back to the master at any given time.

A slave device is selected when its NSS signal is asserted by the master. The USART in SPI Master mode can address only one SPI Slave because it can generate only one NSS signal.

The SPI system consists of two data lines and two control lines:

- Master Out Slave In (MOSI): This data line supplies the output data from the master shifted into the input of the slave.
- Master In Slave Out (MISO): This data line supplies the output data from a slave to the input of the master.
- Serial Clock (SCK): This control line is driven by the master and regulates the flow of the data bits. The master may transmit data at a variety of baud rates. The SCK line cycles once for each bit that is transmitted.
- Slave Select (NSS): This control line allows the master to select or deselect the slave.

#### 31.7.7.1 Modes of Operation

The USART can operate in SPI Master Mode or in SPI Slave Mode.

Operation in SPI Master Mode is programmed by writing to 0xE the USART_MODE field in the Mode Register. In this case the SPI lines must be connected as described below:

- the MOSI line is driven by the output pin TXD
- the MISO line drives the input pin RXD
- the SCK line is driven by the output pin SCK
- the NSS line is driven by the output pin RTS

Operation in SPI Slave Mode is programmed by writing to 0xF the USART_MODE field in the Mode Register. In this case the SPI lines must be connected as described below:

- the MOSI line drives the input pin RXD
- the MISO line is driven by the output pin TXD
- the SCK line drives the input pin SCK
- the NSS line drives the input pin CTS

In order to avoid unpredicted behavior, any change of the SPI Mode must be followed by a software reset of the transmitter and of the receiver (except the initial configuration after a hardware reset). (See Section 31.7.7.4).

#### 31.7.7.2 Baud Rate

In SPI Mode, the baudrate generator operates in the same way as in USART synchronous mode: See “Baud Rate in Synchronous Mode or SPI Mode” on page 540. However, there are some restrictions:

In SPI Master Mode:

- the external clock SCK must not be selected (USCLKS ≠ 0x3), and the bit CLKO must be set to “1” in the Mode Register (US_MR), in order to generate correctly the serial clock on the SCK pin.
- to obtain correct behavior of the receiver and the transmitter, the value programmed in CD must be superior or equal to 6.
- if the internal clock divided (MCK/DIV) is selected, the value programmed in CD must be even to ensure a 50:50 mark/space ratio on the SCK pin, this value can be odd if the internal clock is selected (MCK).

In SPI Slave Mode:

- the external clock (SCK) selection is forced regardless of the value of the USCLKS field in the Mode Register (US_MR). Likewise, the value written in US_BRGR has no effect, because the clock is provided directly by the signal on the USART SCK pin.
- to obtain correct behavior of the receiver and the transmitter, the external clock (SCK) frequency must be at least 6 times lower than the system clock.

#### 31.7.7.3 Data Transfer

Up to 9 data bits are successively shifted out on the TXD pin at each rising or falling edge (depending of CPOL and CPHA) of the programmed serial clock. There is no Start bit, no Parity bit and no Stop bit.

The number of data bits is selected by the CHRL field and the MODE 9 bit in the Mode Register (US_MR). The 9 bits are selected by setting the MODE 9 bit regardless of the CHRL field. The MSB data bit is always sent first in SPI Mode (Master or Slave).

Four combinations of polarity and phase are available for data transfers. The clock polarity is programmed with the CPOL bit in the Mode Register. The clock phase is programmed with the CPHA bit. These two parameters determine the edges of the clock signal upon which data is driven and sampled. Each of the two parameters has two possible states, resulting in four possible combinations that are incompatible with one another. Thus, a master/slave pair must use the same parameter pair values to communicate. If multiple slaves are used and fixed in different configurations, the master must reconfigure itself each time it needs to communicate with a different slave.

**Table 31-14. SPI Bus Protocol Mode**

| SPI Bus Protocol Mode | CPOL | CPHA |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 0 | 0 |
| 2 | 1 | 1 |
| 3 | 1 | 0 |

**Figure 31-28. SPI Transfer Format (CPHA=1, 8 bits per transfer)**

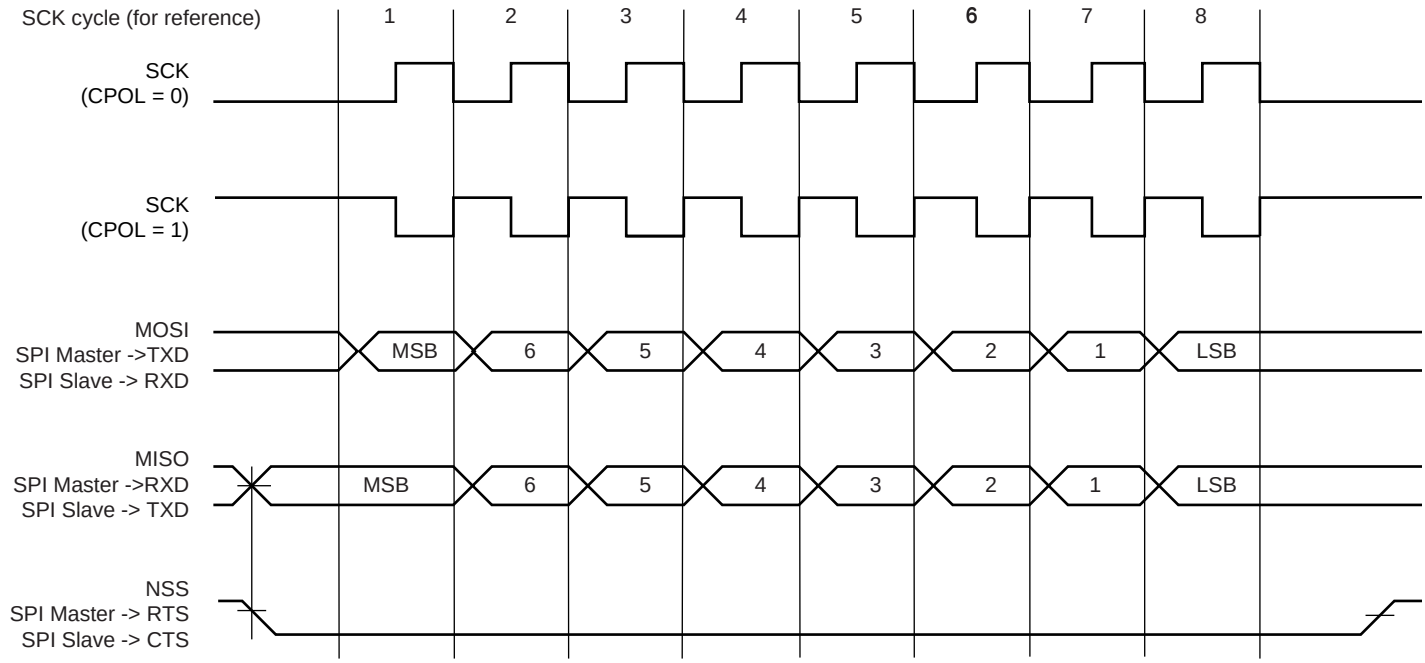
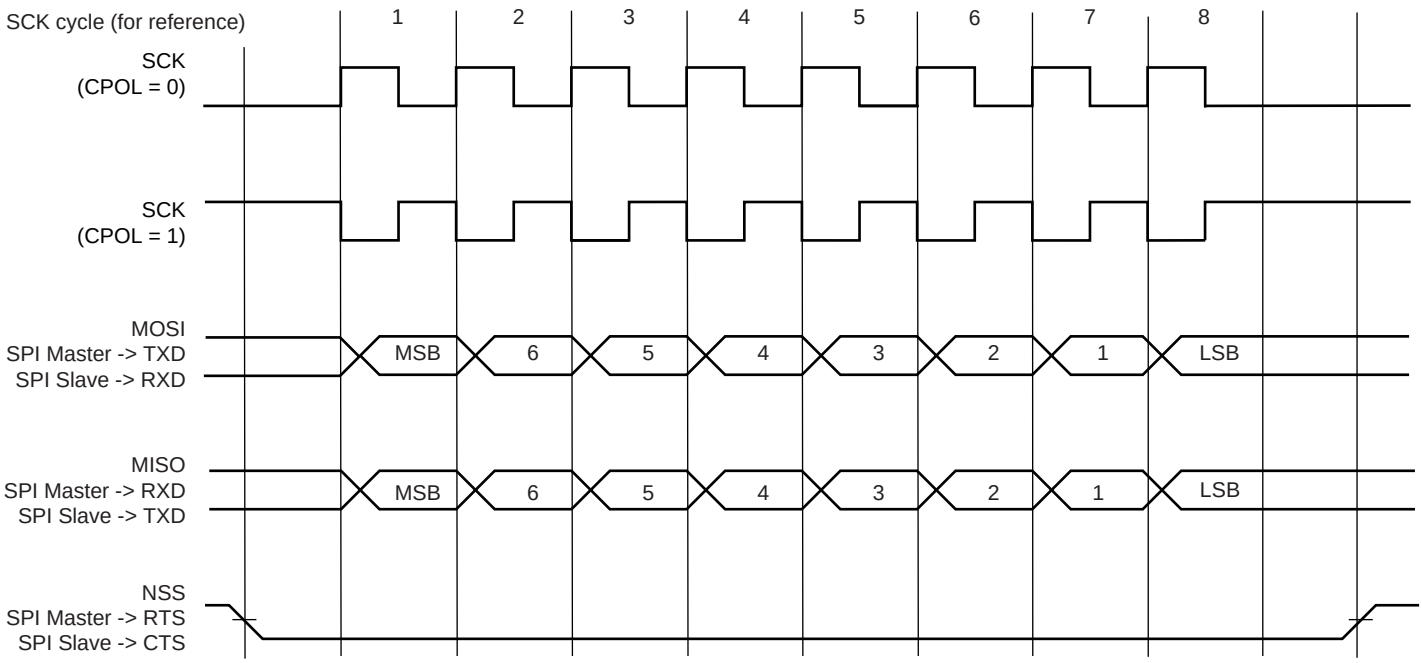



**Figure 31-29. SPI Transfer Format (CPHA=0, 8 bits per transfer)**




### 31.7.7.4 Receiver and Transmitter Control

See “Receiver and Transmitter Control” on page 542.

#### 31.7.7.5 Character Transmission

The characters are sent by writing in the Transmit Holding Register (US_THR). An additional condition for transmitting a character can be added when the USART is configured in SPI master mode. In the USART_MR register, the value configured on INACK field can prevent any character transmission (even if US_THR has been written) while the receiver side is not ready (character not read). When INACK equals 0, the character is transmitted whatever the receiver status. If INACK is set to 1, the transmitter waits for the receiver holding register to be read before transmitting the character (RXRDY flag cleared), thus preventing any overflow (character loss) on the receiver side.

The transmitter reports two status bits in the Channel Status Register (US_CSR): TXRDY (Transmitter Ready), which indicates that US_THR is empty and TXEMPTY, which indicates that all the characters written in US_THR have been processed. When the current character processing is completed, the last character written in US_THR is transferred into the Shift Register of the transmitter and US_THR becomes empty, thus TXRDY rises.

Both TXRDY and TXEMPTY bits are low when the transmitter is disabled. Writing a character in US_THR while TXRDY is low has no effect and the written character is lost.

If the USART is in SPI Slave Mode and if a character must be sent while the Transmit Holding Register (US_THR) is empty, the UNRE (Underrun Error) bit is set. The TXD transmission line stays at high level during all this time. The UNRE bit is cleared by writing the Control Register (US_CR) with the RSTSTA (Reset Status) bit to 1.

In SPI Master Mode, the slave select line (NSS) is asserted at low level 1 Tbit (Time bit) before the transmission of the MSB bit and released at high level 1 Tbit after the transmission of the LSB bit. So, the slave select line (NSS) is always released between each character transmission and a minimum delay of 3 Tbits always inserted. However, in order to address slave devices supporting the CSAAT mode (Chip Select Active After Transfer), the slave select line (NSS) can be forced at low level by writing the Control Register (US_CR) with the RTSEN bit to 1. The slave select line (NSS) can be released at high level only by writing the Control Register (US_CR) with the RTSDIS bit to 1 (for example, when all data have been transferred to the slave device).

In SPI Slave Mode, the transmitter does not require a falling edge of the slave select line (NSS) to initiate a character transmission but only a low level. However, this low level must be present on the slave select line (NSS) at least 1 Tbit before the first serial clock cycle corresponding to the MSB bit.

#### 31.7.7.6 Character Reception

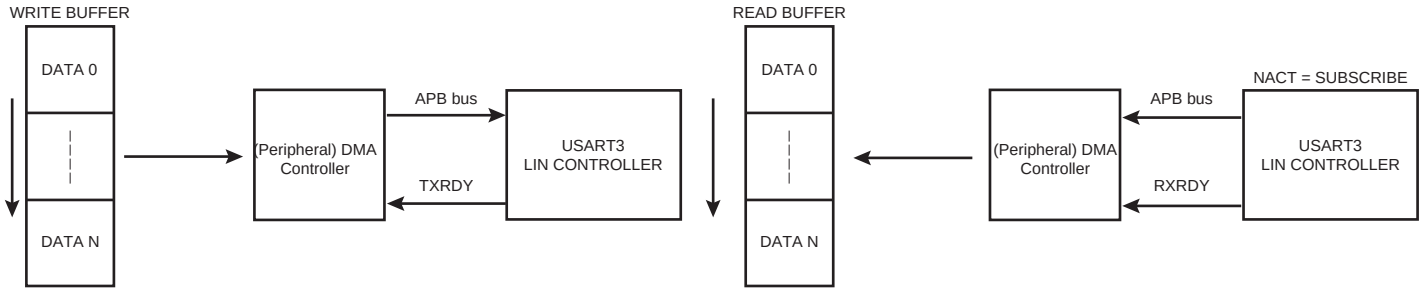
When a character reception is completed, it is transferred to the Receive Holding Register (US_RHR) and the RXRDY bit in the Status Register (US_CSR) rises. If a character is completed while RXRDY is set, the OVRE (Overrun Error) bit is set. The last character is transferred into US_RHR and overwrites the previous one. The OVRE bit is cleared by writing the Control Register (US_CR) with the RSTSTA (Reset Status) bit to 1.

To ensure correct behavior of the receiver in SPI Slave Mode, the master device sending the frame must ensure a minimum delay of 1 Tbit between each character transmission. The receiver does not require a falling edge of the slave select line (NSS) to initiate a character reception but only a low level. However, this low level must be present on the slave select line (NSS) at least 1 Tbit before the first serial clock cycle corresponding to the MSB bit.

#### 31.7.7.7 Receiver Timeout

Because the receiver baudrate clock is active only during data transfers in SPI Mode, a receiver timeout is impossible in this mode, whatever the Time-out value is (field TO) in the Time-out Register (US_RTOR).

### 31.7.8 Test Modes

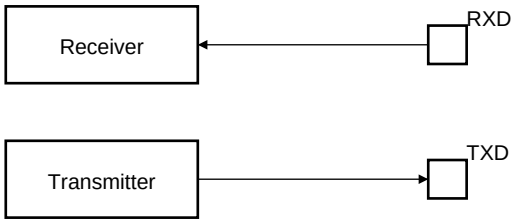



The USART can be programmed to operate in three different test modes. The internal loopback capability allows on-board diagnostics. In the loopback mode the USART interface pins are disconnected or not and reconfigured for loopback internally or externally.

#### 31.7.8.1 Normal Mode

Normal mode connects the RXD pin on the receiver input and the transmitter output on the TXD pin.

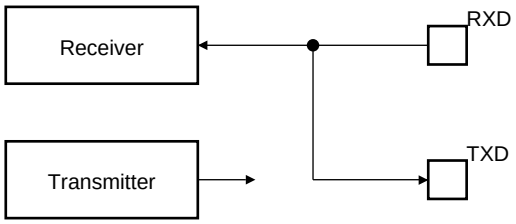
**Figure 31-30. Normal Mode Configuration**




#### 31.7.8.2 Automatic Echo Mode

Automatic echo mode allows bit-by-bit retransmission. When a bit is received on the RXD pin, it is sent to the TXD pin, as shown in Figure 31-31. Programming the transmitter has no effect on the TXD pin. The RXD pin is still connected to the receiver input, thus the receiver remains active.

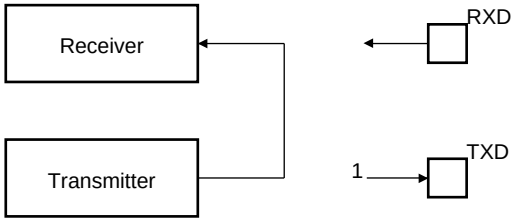
**Figure 31-31. Automatic Echo Mode Configuration**



#### 31.7.8.3 Local Loopback Mode

Local loopback mode connects the output of the transmitter directly to the input of the receiver, as shown in Figure 31-32. The TXD and RXD pins are not used. The RXD pin has no effect on the receiver and the TXD pin is continuously driven high, as in idle state.

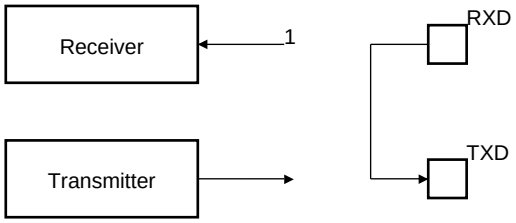
**Figure 31-32. Local Loopback Mode Configuration**




#### 31.7.8.4 Remote Loopback Mode

Remote loopback mode directly connects the RXD pin to the TXD pin, as shown in Figure 31-33. The transmitter and the receiver are disabled and have no effect. This mode allows bit-by-bit retransmission.

**Figure 31-33. Remote Loopback Mode Configuration**




### 31.7.9 Write Protection Registers

To prevent any single software error that may corrupt USART behavior, certain address spaces can be write-protected by setting the WPEN bit in the USART Write Protect Mode Register (US_WPMR).

If a write access to the protected registers is detected, then the WPVS flag in the USART Write Protect Status Register (US_WPSR) is set and the field WPVSRC indicates in which register the write access has been attempted.

The WPVS flag is reset by writing the USART Write Protect Mode Register (US_WPMR) with the appropriate access key, WPKEY.

The protected registers are:

- "USART Mode Register"
- "USART Baud Rate Generator Register"
- "USART Receiver Time-out Register"
- "USART Transmitter Timeguard Register"
- "USART FI DI RATIO Register"
- "USART IrDA FILTER Register"

## 31.8 Universal Synchronous Asynchronous Receiver Transmitter (USART) User Interface

**Table 31-15. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x0000 | Control Register | US_CR | Write-only | – |
| 0x0004 | Mode Register | US_MR | Read-write | – |
| 0x0008 | Interrupt Enable Register | US_IER | Write-only | – |
| 0x000C | Interrupt Disable Register | US_IDR | Write-only | – |
| 0x0010 | Interrupt Mask Register | US_IMR | Read-only | 0x0 |
| 0x0014 | Channel Status Register | US_CSR | Read-only | – |
| 0x0018 | Receiver Holding Register | US_RHR | Read-only | 0x0 |
| 0x001C | Transmitter Holding Register | US_THR | Write-only | – |
| 0x0020 | Baud Rate Generator Register | US_BRGR | Read-write | 0x0 |
| 0x0024 | Receiver Time-out Register | US_RTOR | Read-write | 0x0 |
| 0x0028 | Transmitter Timeguard Register | US_TTGR | Read-write | 0x0 |
| 0x2C - 0x3C | Reserved | – | – | – |
| 0x0040 | FI DI Ratio Register | US_FIDI | Read-write | 0x174 |
| 0x0044 | Number of Errors Register | US_NER | Read-only | – |
| 0x0048 | Reserved | – | – | – |
| 0x004C | IrDA Filter Register | US_IF | Read-write | 0x0 |
| 0xE4 | Write Protect Mode Register | US_WPMR | Read-write | 0x0 |
| 0xE8 | Write Protect Status Register | US_WPSR | Read-only | 0x0 |
| 0x5C - 0xFC | Reserved | – | – | – |
| 0x100 - 0x128 | Reserved for PDC Registers | – | – | – |

### 31.8.1 USART Control Register

**Name:** US_CR

**Addresses:** 0x40024000 (0), 0x40028000 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | RTSDIS/RCS | RTSEN/FCS | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| RETTO | RSTNACK | RSTIT | SENDA | STTTO | STPBRK | STTBRK | RSTSTA |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXDIS | TXEN | RXDIS | RXEN | RSTTX | RSTRX | – | – |

- **RSTRX: Reset Receiver**

0: No effect.

1: Resets the receiver.

- **RSTTX: Reset Transmitter**

0: No effect.

1: Resets the transmitter.

- **RXEN: Receiver Enable**

0: No effect.

1: Enables the receiver, if RXDIS is 0.

- **RXDIS: Receiver Disable**

0: No effect.

1: Disables the receiver.

- **TXEN: Transmitter Enable**

0: No effect.

1: Enables the transmitter if TXDIS is 0.

- **TXDIS: Transmitter Disable**

0: No effect.

1: Disables the transmitter.

- **RSTSTA: Reset Status Bits**

0: No effect.

1: Resets the status bits PARE, FRAME, OVRE, **UNRE** and RXBRK in US_CSR.

- **STTBRK: Start Break**

0: No effect.

1: Starts transmission of a break after the characters present in US_THR and the Transmit Shift Register have been transmitted. No effect if a break is already being transmitted.

- **STPBRK: Stop Break**

0: No effect.

1: Stops transmission of the break after a minimum of one character length and transmits a high level during 12-bit periods. No effect if no break is being transmitted.

- **STTTO: Start Time-out**

0: No effect.

1: Starts waiting for a character before clocking the time-out counter. Resets the status bit TIMEOUT in US_CSR.

- **SENDA: Send Address**

0: No effect.

1: In Multidrop Mode only, the next character written to the US_THR is sent with the address bit set.

- **RSTIT: Reset Iterations**

0: No effect.

1: Resets ITERATION in US_CSR. No effect if the ISO7816 is not enabled.

- **RSTNACK: Reset Non Acknowledge**

0: No effect

1: Resets NACK in US_CSR.

- **RETTO: Rearm Time-out**

0: No effect

1: Restart Time-out

- **RTSEN: Request to Send Enable**

0: No effect.

1: Drives the pin RTS to 0.

- **FCS: Force SPI Chip Select**
  - Applicable if USART operates in SPI Master Mode (USART_MODE = 0xE):

FCS = 0: No effect.

FCS = 1: Forces the Slave Select Line NSS (RTS pin) to 0, even if USART is no transmitting, in order to address SPI slave devices supporting the CSAAT Mode (Chip Select Active After Transfer).

- **RTSDIS: Request to Send Disable**

0: No effect.

1: Drives the pin RTS to 1.

- **RCS: Release SPI Chip Select**
  - Applicable if USART operates in SPI Master Mode (USART_MODE = 0xE):

RCS = 0: No effect.

RCS = 1: Releases the Slave Select Line NSS (RTS pin).

### 31.8.2 USART Mode Register

**Name:** US_MR

**Addresses:** 0x40024004 (0), 0x40028004 (1)

**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/CPOL |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CHMODE | | NBSTOP | | PAR | | | SYNC/CPHA |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CHRL | | USCLKS | | USART_MODE | | | |

This register can only be written if the WPEN bit is cleared in “USART Write Protect Mode Register” on page 583.

- **USART_MODE**

| Value | Name | Description |
|---|---|---|
| 0x0 | NORMAL | Normal mode |
| 0x1 | RS485 | RS485 |
| 0x2 | HW_HANDSHAKING | Hardware Handshaking |
| 0x4 | IS07816_T_0 | IS07816 Protocol: T = 0 |
| 0x6 | IS07816_T_1 | IS07816 Protocol: T = 1 |
| 0x8 | IRDA | IrDA |
| 0xE | SPI_MASTER | SPI Master |
| 0xF | SPI_SLAVE | SPI Slave |

- **USCLKS: Clock Selection**

| Value | Name | Description |
|---|---|---|
| 0 | MCK | Master Clock MCK is selected |
| 1 | DIV | Internal Clock Divided MCK/DIV (DIV=8) is selected |
| 3 | SCK | Serial Clock SLK is selected |

- **CHRL: Character Length.**

| Value | Name | Description |
|---|---|---|
| 0 | 5_BIT | Character length is 5 bits |
| 1 | 6_BIT | Character length is 6 bits |
| 2 | 7_BIT | Character length is 7 bits |
| 3 | 8_BIT | Character length is 8 bits |

- **SYNC: Synchronous Mode Select**

0: USART operates in Asynchronous Mode.

1: USART operates in Synchronous Mode.

- **CPHA: SPI Clock Phase**
  - Applicable if USART operates in SPI Mode (USART_MODE = 0xE or 0xF):

CPHA = 0: Data is changed on the leading edge of SPCK and captured on the following edge of SPCK.

CPHA = 1: Data is captured on the leading edge of SPCK and changed on the following edge of SPCK.

CPHA determines which edge of SPCK causes data to change and which edge causes data to be captured. CPHA is used with CPOL to produce the required clock/data relationship between master and slave devices.

- **PAR: Parity Type**

| Value | Name | Description |
|---|---|---|
| 0 | EVEN | Even parity |
| 1 | ODD | Odd parity |
| 2 | SPACE | Parity forced to 0 (Space) |
| 3 | MARK | Parity forced to 1 (Mark) |
| 4 | NO | No parity |
| 6 | MULTIDROP | Multidrop mode |

- **NBSTOP: Number of Stop Bits**

| Value | Name | Description |
|---|---|---|
| 0 | 1_BIT | 1 stop bit |
| 1 | 1_5_BIT | 1.5 stop bit (SYNC = 0) or reserved (SYNC = 1) |
| 2 | 2_BIT | 2 stop bits |

- **CHMODE: Channel Mode**

| Value | Name | Description |
|---|---|---|
| 0 | NORMAL | Normal Mode |
| 1 | AUTOMATIC | Automatic Echo. Receiver input is connected to the TXD pin. |
| 2 | LOCAL_LOOPBACK | Local Loopback. Transmitter output is connected to the Receiver Input. |
| 3 | REMOTE_LOOPBACK | Remote Loopback. RXD pin is internally connected to the TXD pin. |

- **MSBF: Bit Order**

0: Least Significant Bit is sent/received first.

1: Most Significant Bit is sent/received first.

- **CPOL: SPI Clock Polarity**
  - Applicable if USART operates in SPI Mode (Slave or Master, USART_MODE = 0xE or 0xF):

CPOL = 0: The inactive state value of SPCK is logic level zero.

CPOL = 1: The inactive state value of SPCK is logic level one.

CPOL is used to determine the inactive state value of the serial clock (SPCK). It is used with CPHA to produce the required clock/data relationship between master and slave devices.

- **MODE9: 9-bit Character Length**

0: CHRL defines character length.

1: 9-bit character length.

- **CLKO: Clock Output Select**

0: The USART does not drive the SCK pin.

1: The USART drives the SCK pin if USCLKS does not select the external clock SCK.

- **OVER: Oversampling Mode**

0: 16x Oversampling.

1: 8x Oversampling.

- **INACK: Inhibit Non Acknowledge**

0: The NACK is generated.

1: The NACK is not generated.

Note: In SPI master mode, if INACK = 0 the character transmission starts as soon as a character is written into US_THR register (assuming TXRDY was set). When INACK is 1, an additional condition must be met. The character transmission starts when a character is written and only if RXRDY flag is cleared (Receiver Holding Register has been read).

- **DSNACK: Disable Successive NACK**

0: NACK is sent on the ISO line as soon as a parity error occurs in the received character (unless INACK is set).

1: Successive parity errors are counted up to the value specified in the MAX_ITERATION field. These parity errors generate a NACK on the ISO line. As soon as this value is reached, no additional NACK is sent on the ISO line. The flag ITERATION is asserted.

- **INVDATA: INverted Data**

0: The data field transmitted on TXD line is the same as the one written in US_THR register or the content read in US_RHR is the same as RXD line. Normal mode of operation.

1: The data field transmitted on TXD line is inverted (voltage polarity only) compared to the value written on US_THR register or the content read in US_RHR is inverted compared to what is received on RXD line (or ISO7816 IO line). Inverted Mode of operation, useful for contactless card application. To be used with configuration bit MSBF.

- **MAX_ITERATION**

Defines the maximum number of iterations in mode ISO7816, protocol T= 0.

- **FILTER: Infrared Receive Line Filter**

0: The USART does not filter the receive line.

1: The USART filters the receive line using a three-sample filter (1/16-bit clock) (2 over 3 majority).

### 31.8.3 USART Interrupt Enable Register

**Name:** US_IER

**Addresses:** 0x40024008 (0), 0x40028008 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | CTSIC | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | NACK | RXBUFF | TXBUFE | ITER/UNRE | TXEMPTY | TIMEOUT |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PARE | FRAME | OVRE | ENDTX | ENDRX | RXBRK | TXRDY | RXRDY |

0: No effect

1: Enables the corresponding interrupt.

- **RXRDY: RXRDY Interrupt Enable**
- **TXRDY: TXRDY Interrupt Enable**
- **RXBRK: Receiver Break Interrupt Enable**
- **ENDRX: End of Receive Transfer Interrupt Enable**
- **ENDTX: End of Transmit Interrupt Enable**
- **OVRE: Overrun Error Interrupt Enable**
- **FRAME: Framing Error Interrupt Enable**
- **PARE: Parity Error Interrupt Enable**
- **TIMEOUT: Time-out Interrupt Enable**
- **TXEMPTY: TXEMPTY Interrupt Enable**
- **ITER: Max number of Repetitions Reached**
- **UNRE: SPI Underrun Error**
- **TXBUFE: Buffer Empty Interrupt Enable**
- **RXBUFF: Buffer Full Interrupt Enable**
- **NACK: Non AcknowledgeInterrupt Enable**
- **CTSIC: Clear to Send Input Change Interrupt Enable**

### 31.8.4 USART Interrupt Disable Register

**Name:** US_IDR

**Addresses:** 0x4002400C (0), 0x4002800C (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | CTSIC | – | – | – |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | NACK | RXBUFF | TXBUFE | ITER/UNRE | TXEMPTY | TIMEOUT |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| PARE | FRAME | OVRE | ENDTX | ENDRX | RXBRK | TXRDY | RXRDY |

0: No effect

1: Disables the corresponding interrupt.

- **RXRDY: RXRDY Interrupt Disable**
- **TXRDY: TXRDY Interrupt Disable**
- **RXBRK: Receiver Break Interrupt Disable**
- **ENDRX: End of Receive Transfer Interrupt Disable**
- **ENDTX: End of Transmit Interrupt Disable**
- **OVRE: Overrun Error Interrupt Disable**
- **FRAME: Framing Error Interrupt Disable**
- **PARE: Parity Error Interrupt Disable**
- **TIMEOUT: Time-out Interrupt Disable**
- **TXEMPTY: TXEMPTY Interrupt Disable**
- **ITER: Max number of Repetitions Reached Disable**
- **UNRE: SPI Underrun Error Disable**
- **TXBUFE: Buffer Empty Interrupt Disable**
- **RXBUFF: Buffer Full Interrupt Disable**
- **NACK: Non AcknowledgeInterrupt Disable**
- **CTSIC: Clear to Send Input Change Interrupt Disable**

### 31.8.5 USART Interrupt Mask Register

**Name:** US_IMR

**Addresses:** 0x40024010 (0), 0x40028010 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | CTSIC | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | NACK | RXBUFF | TXBUFE | ITER/UNRE | TXEMPTY | TIMEOUT |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PARE | FRAME | OVRE | ENDTX | ENDRX | RXBRK | TXRDY | RXRDY |

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

- **RXRDY: RXRDY Interrupt Mask**
- **TXRDY: TXRDY Interrupt Mask**
- **RXBRK: Receiver Break Interrupt Mask**
- **ENDRX: End of Receive Transfer Interrupt Mask**
- **ENDTX: End of Transmit Interrupt Mask**
- **OVRE: Overrun Error Interrupt Mask**
- **FRAME: Framing Error Interrupt Mask**
- **PARE: Parity Error Interrupt Mask**
- **TIMEOUT: Time-out Interrupt Mask**
- **TXEMPTY: TXEMPTY Interrupt Mask**
- **ITER: Max number of Repetitions Reached Mask**
- **UNRE: SPI Underrun Error Mask**
- **TXBUFE: Buffer Empty Interrupt Mask**
- **RXBUFF: Buffer Full Interrupt Mask**
- **NACK: Non AcknowledgeInterrupt Mask**
- **CTSIC: Clear to Send Input Change Interrupt Mask**

### 31.8.6 USART Channel Status Register

**Name:** US_CSR

**Addresses:** 0x40024014 (0), 0x40028014 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| CTS | – | – | – | CTSIC | – | – | – |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | NACK | RXBUFF | TXBUFE | ITER/UNRE | TXEMPTY | TIMEOUT |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| PARE | FRAME | OVRE | ENDTX | ENDRX | RXBRK | TXRDY | RXRDY |

- **RXRDY: Receiver Ready**

0: No complete character has been received since the last read of US_RHR or the receiver is disabled. If characters were being received when the receiver was disabled, RXRDY changes to 1 when the receiver is enabled.

1: At least one complete character has been received and US_RHR has not yet been read.

- **TXRDY: Transmitter Ready**

0: A character is in the US_THR waiting to be transferred to the Transmit Shift Register, or an STTBRK command has been requested, or the transmitter is disabled. As soon as the transmitter is enabled, TXRDY becomes 1.

1: There is no character in the US_THR.

- **RXBRK: Break Received/End of Break**

0: No Break received or End of Break detected since the last RSTSTA.

1: Break Received or End of Break detected since the last RSTSTA.

- **ENDRX: End of Receiver Transfer**

0: The End of Transfer signal from the Receive PDC channel is inactive.

1: The End of Transfer signal from the Receive PDC channel is active.

- **ENDTX: End of Transmitter Transfer**

0: The End of Transfer signal from the Transmit PDC channel is inactive.

1: The End of Transfer signal from the Transmit PDC channel is active.

- **OVRE: Overrun Error**

0: No overrun error has occurred since the last RSTSTA.

1: At least one overrun error has occurred since the last RSTSTA.

- **FRAME: Framing Error**

0: No stop bit has been detected low since the last RSTSTA.

1: At least one stop bit has been detected low since the last RSTSTA.

- **PARE: Parity Error**

0: No parity error has been detected since the last RSTSTA.

1: At least one parity error has been detected since the last RSTSTA.

- **TIMEOUT: Receiver Time-out**

0: There has not been a time-out since the last Start Time-out command (STTTO in US_CR) or the Time-out Register is 0.

1: There has been a time-out since the last Start Time-out command (STTTO in US_CR).

- **TXEMPTY: Transmitter Empty**

0: There are characters in either US_THR or the Transmit Shift Register, or the transmitter is disabled.

1: There are no characters in US_THR, nor in the Transmit Shift Register.

- **ITER: Max number of Repetitions Reached**

0: Maximum number of repetitions has not been reached since the last RSTSTA.

1: Maximum number of repetitions has been reached since the last RSTSTA.

- **UNRE: SPI Underrun Error**

– Applicable if USART operates in SPI Slave Mode (USART_MODE = 0xF):

UNRE = 0: No SPI underrun error has occurred since the last RSTSTA.

UNRE = 1: At least one SPI underrun error has occurred since the last RSTSTA.

- **TXBUFE: Transmission Buffer Empty**

0: The signal Buffer Empty from the Transmit PDC channel is inactive.

1: The signal Buffer Empty from the Transmit PDC channel is active.

- **RXBUFF: Reception Buffer Full**

0: The signal Buffer Full from the Receive PDC channel is inactive.

1: The signal Buffer Full from the Receive PDC channel is active.

- **NACK: Non AcknowledgeInterrupt**

0: Non Acknowledge has not been detected since the last RSTNACK.

1: At least one Non Acknowledge has been detected since the last RSTNACK.

- **CTSIC: Clear to Send Input Change Flag**

0: No input change has been detected on the CTS pin since the last read of US_CSR.

1: At least one input change has been detected on the CTS pin since the last read of US_CSR.

- **CTS: Image of CTS Input**

0: CTS is set to 0.

1: CTS is set to 1.

### 31.8.7 USART Receive Holding Register

**Name:** US_RHR

**Addresses:** 0x40024018 (0), 0x40028018 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| RXSYNH | – | – | – | – | – | – | RXCHR |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| RXCHR | | | | | | | |

- **RXCHR: Received Character**

Last character received if RXRDY is set.

- **RXSYNH: Received Sync**

0: Last Character received is a Data.

1: Last Character received is a Command.

### 31.8.8 USART Transmit Holding Register

**Name:** US_THR

**Addresses:** 0x4002401C (0), 0x4002801C (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| TXSYNH | – | – | – | – | – | – | TXCHR |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| TXCHR | | | | | | | |

- **TXCHR: Character to be Transmitted**

Next character to be transmitted after the current character if TXRDY is not set.

- **TXSYNH: Sync Field to be transmitted**

0: The next character sent is encoded as a data. Start Frame Delimiter is DATA SYNC.

1: The next character sent is encoded as a command. Start Frame Delimiter is COMMAND SYNC.

### 31.8.9 USART Baud Rate Generator Register

**Name:** US_BRGR

**Addresses:** 0x40024020 (0), 0x40028020 (1)

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | FP | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CD | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CD | | | | | | | |

This register can only be written if the WPEN bit is cleared in “USART Write Protect Mode Register” on page 583.

- **CD: Clock Divider**

| CD | USART_MODE ≠ ISO7816 | | | USART_MODE = ISO7816 |
|---|---|---|---|---|
| | SYNC = 0 | | SYNC = 1 or USART_MODE = SPI (Master or Slave) | |
| | OVER = 0 | OVER = 1 | | |
| 0 | Baud Rate Clock Disabled | | | |
| 1 to 65535 | Baud Rate = Selected Clock/(16*CD) | Baud Rate = Selected Clock/(8*CD) | Baud Rate = Selected Clock /CD | Baud Rate = Selected Clock/(FI_DI_RATIO*CD) |

- **FP: Fractional Part**

0: Fractional divider is disabled.

1 - 7: Baudrate resolution, defined by FP x 1/8.

### 31.8.10 USART Receiver Time-out Register

**Name:** US_RTOR

**Addresses:** 0x40024024 (0), 0x40028024 (1)

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| TO | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| TO | | | | | | | |

This register can only be written if the WPEN bit is cleared in “USART Write Protect Mode Register” on page 583.

- **TO: Time-out Value**

0: The Receiver Time-out is disabled.

1 - 65535: The Receiver Time-out is enabled and the Time-out delay is TO x Bit Period.

### 31.8.11 USART Transmitter Timeguard Register

**Name:** US_TTGR

**Addresses:** 0x40024028 (0), 0x40028028 (1)

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TG | | | | | | | |

This register can only be written if the WPEN bit is cleared in “USART Write Protect Mode Register” on page 583.

- **TG: Timeguard Value**

0: The Transmitter Timeguard is disabled.

1 - 255: The Transmitter timeguard is enabled and the timeguard delay is TG x Bit Period.

### 31.8.12 USART FI DI RATIO Register

**Name:** US_FIDI

**Addresses:** 0x40024040 (0), 0x40028040 (1)

**Access:** Read-write

**Reset Value:** 0x174

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | FI_DI_RATIO | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| FI_DI_RATIO | | | | | | | |

This register can only be written if the WPEN bit is cleared in “USART Write Protect Mode Register” on page 583.

- **FI_DI_RATIO: FI Over DI Ratio Value**

0: If ISO7816 mode is selected, the Baud Rate Generator generates no signal.

1 - 2047: If ISO7816 mode is selected, the Baud Rate is the clock provided on SCK divided by FI_DI_RATIO.

### 31.8.13 USART Number of Errors Register

**Name:** US_NER

**Addresses:** 0x40024044 (0), 0x40028044 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| NB_ERRORS | | | | | | | |

- **NB_ERRORS: Number of Errors**

Total number of errors that occurred during an ISO7816 transfer. This register automatically clears when read.

### 31.8.14 USART IrDA FILTER Register

**Name:** US_IF

**Addresses:** 0x4002404C (0), 0x4002804C (1)

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IRDA_FILTER | | | | | | | |

This register can only be written if the WPEN bit is cleared in “USART Write Protect Mode Register” on page 583.

- **IRDA_FILTER: IrDA Filter**

Sets the filter of the IrDA demodulator.

### 31.8.15 USART Write Protect Mode Register

**Name:** US_WPMR

**Addresses:** 0x400240E4 (0), 0x400280E4 (1)

**Access:** Read-write

**Reset:** See Table 31-15

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| WPKEY | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| WPKEY | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| — | — | — | — | — | — | — | WPEN |

- **WPEN: Write Protect Enable**

0 = Disables the Write Protect if WPKEY corresponds to 0x555341 (“USA” in ASCII).

1 = Enables the Write Protect if WPKEY corresponds to 0x555341 (“USA” in ASCII).

Protects the registers:

- “USART Mode Register” on page 567
- “USART Baud Rate Generator Register” on page 577
- “USART Receiver Time-out Register” on page 578
- “USART Transmitter Timeguard Register” on page 579
- “USART FI DI RATIO Register” on page 580
- “USART IrDA FILTER Register” on page 582

- **WPKEY: Write Protect KEY**

Should be written at value 0x555341 (“USA” in ASCII). Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

### 31.8.16 USART Write Protect Status Register

**Name:** US_WPSR

**Addresses:** 0x400240E8 (0), 0x400280E8 (1)

**Access:** Read-only

**Reset:** See Table 31-15

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| WPVSRC | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| WPVSRC | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | WPVS |

- **WPVS: Write Protect Violation Status**

0 = No Write Protect Violation has occurred since the last read of the US_WPSR register.

1 = A Write Protect Violation has occurred since the last read of the US_WPSR register. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protect Violation Source**

When WPVS is active, this field indicates the write-protected register (through address offset or code) in which a write access has been attempted.

Note: Reading US_WPSR automatically clears all fields.

# 32. Timer Counter (TC)

## 32.1 Description

A Timer Counter (TC) module includes three identical TC channels. The number of implemented TC modules is device-specific.

Each TC channel can be independently programmed to perform a wide range of functions including frequency measurement, event counting, interval measurement, pulse generation, delay timing and pulse width modulation.

Each channel has three external clock inputs, five internal clock inputs and two multi-purpose input/output signals which can be configured by the user. Each channel drives an internal interrupt signal which can be programmed to generate processor interrupts.

The TC embeds a quadrature decoder (QDEC) connected in front of the timers and driven by TIOA0, TIOB0 and TIOB1 inputs. When enabled, the QDEC performs the input lines filtering, decoding of quadrature signals and connects to the timers/counters in order to read the position and speed of the motor through the user interface.

The TC block has two global registers which act upon all TC channels:

- Block Control Register (TC_BCR)—allows channels to be started simultaneously with the same instruction
- Block Mode Register (TC_BMR)—defines the external clock inputs for each channel, allowing them to be chained

## 32.2 Embedded Characteristics

- Total number of TC channels: 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
  - Quadrature decoder
  - 2-bit gray up/down count for stepper motor
- Each channel is user-configurable and contains:
  - Three external clock inputs
  - Five Internal clock inputs
  - Two multi-purpose input/output signals acting as trigger event
- Internal interrupt signal
- Register Write Protection

## 32.3 Block Diagram

**Table 32-1. Timer Counter Clock Assignment**

| Name | Definition |
|---|---|
| TIMER_CLOCK1 | MCK/2 |
| TIMER_CLOCK2 | MCK/8 |
| TIMER_CLOCK3 | MCK/32 |
| TIMER_CLOCK4 | MCK/128 |
| TIMER_CLOCK5 | SLCK |

Note: 1. When SLCK is selected for Peripheral Clock (CSS = 0 in PMC Master Clock Register), SLCK input is equivalent to Peripheral Clock.

**Figure 32-1. Timer Counter Block Diagram**

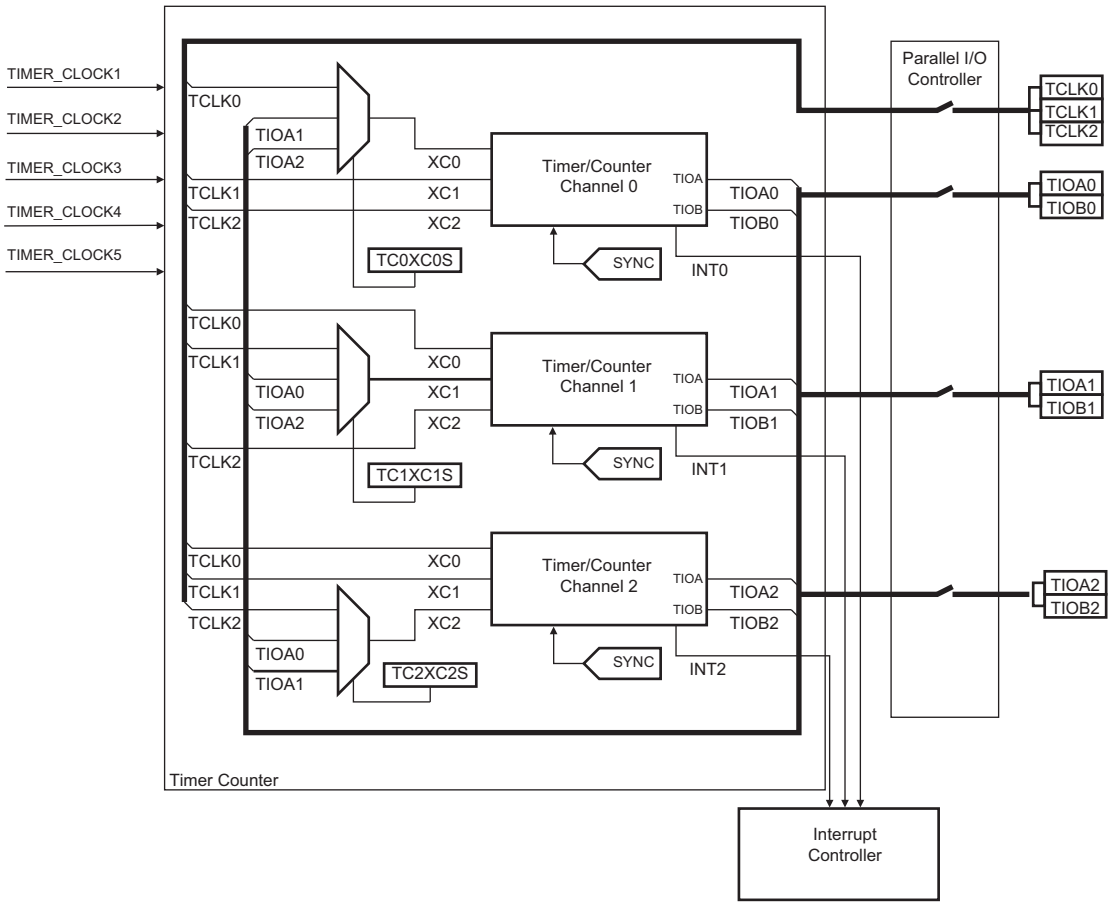


**Table 32-2. Signal Name Description**

| Block/Channel | Signal Name | Description |
|---|---|---|
| Channel Signal | XC0, XC1, XC2 | External Clock Inputs |
| | TIOA | Capture Mode: Timer Counter Input<br>Waveform Mode: Timer Counter Output |
| | TIOB | Capture Mode: Timer Counter Input<br>Waveform Mode: Timer Counter Input/Output |
| | INT | Interrupt Signal Output (internal signal) |
| | SYNC | Synchronization Input Signal (from configuration register) |

## 32.4 Pin Name List

**Table 32-3. TC Pin List**

| Pin Name | Description | Type |
|---|---|---|
| TCLK0–TCLK2 | External Clock Input | Input |
| TIOA0–TIOA2 | I/O Line A | I/O |
| TIOB0–TIOB2 | I/O Line B | I/O |

## 32.5 Product Dependencies

### 32.5.1 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 32-4. I/O Lines**

| Instance | Signal | I/O Line | Peripheral |
|---|---|---|---|
| TC0 | TCLK0 | PA4 | B |
| TC0 | TCLK1 | PA28 | B |
| TC0 | TCLK2 | PA29 | B |
| TC0 | TIOA0 | PA0 | B |
| TC0 | TIOA1 | PA15 | B |
| TC0 | TIOA2 | PA26 | B |
| TC0 | TIOB0 | PA1 | B |
| TC0 | TIOB1 | PA16 | B |
| TC0 | TIOB2 | PA27 | B |
| TC1 | TCLK3 | PC25 | B |
| TC1 | TCLK4 | PC28 | B |
| TC1 | TCLK5 | PC31 | B |
| TC1 | TIOA3 | PC23 | B |
| TC1 | TIOA4 | PC26 | B |
| TC1 | TIOA5 | PC29 | B |
| TC1 | TIOB3 | PC24 | B |
| TC1 | TIOB4 | PC27 | B |
| TC1 | TIOB5 | PC30 | B |

### 32.5.2 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.

### 32.5.3 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 32-5. Peripheral IDs**

| Instance | ID |
|---|---|
| TC0 | 23 |
| TC1 | 24 |

## 32.6 Functional Description

### 32.6.1 Description

All channels of the Timer Counter are independent and identical in operation except when the QDEC is enabled. The registers for channel programming are listed in Table 32-6 “Register Mapping”.

### 32.6.2 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.

### 32.6.3 Clock Selection

At block level, input clock signals of each channel can either be connected to the external inputs TCLK0, TCLK1 or TCLK2, or be connected to the internal I/O signals TIOA0, TIOA1 or TIOA2 for chaining by programming the TC Block Mode Register (TC_BMR). See Figure 32-2.

Each channel can independently select an internal or external clock source for its counter:

- External clock signals(1): XC0, XC1 or XC2
- Internal clock signals: MCK/2, MCK/8, MCK/32, MCK/128, SLCK

This selection is made by the TCCLKS bits in the TC Channel Mode Register (TC_CMR).

The selected clock can be inverted with the CLKI bit in the TC_CMR. This allows counting on the opposite edges of the clock.

The burst function allows the clock to be validated when an external signal is high. The BURST parameter in the TC_CMR defines this signal (none, XC0, XC1, XC2). See Figure 32-3.

Note: 1. In all cases, if an external clock is used, the duration of each of its levels must be longer than the peripheral clock period. The external clock frequency must be at least 2.5 times lower than the peripheral clock.

**Figure 32-2. Clock Chaining Selection**

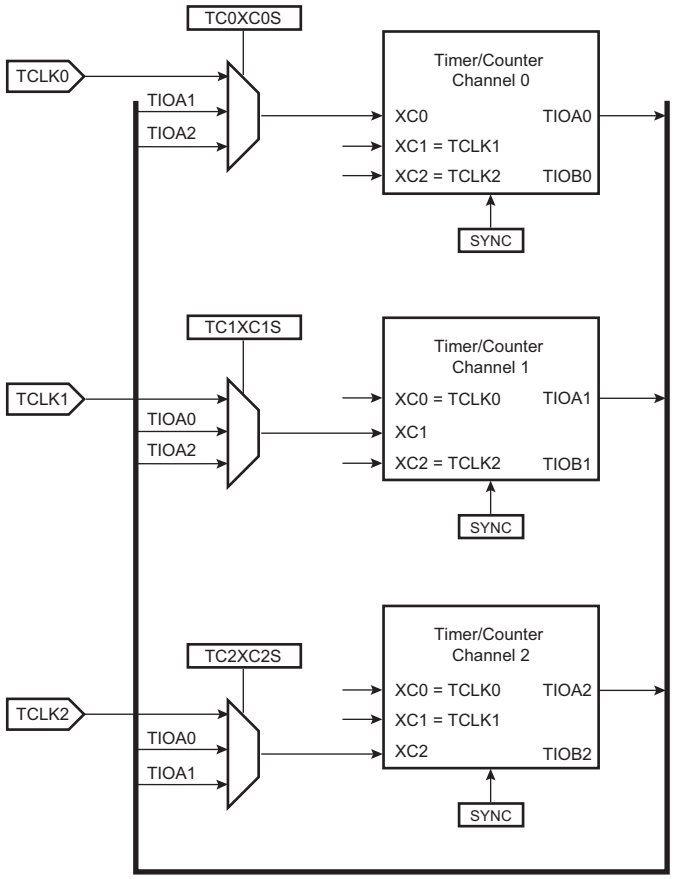
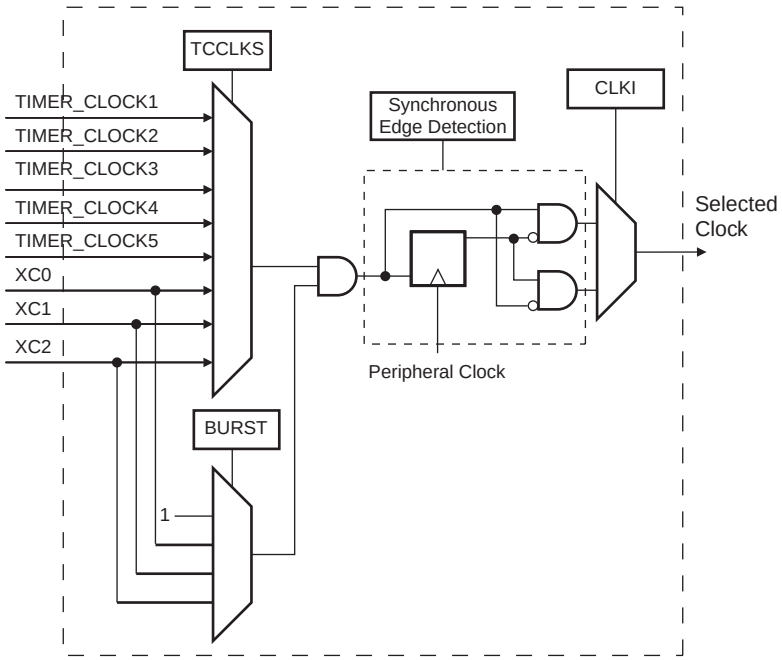



**Figure 32-3. Clock Selection**

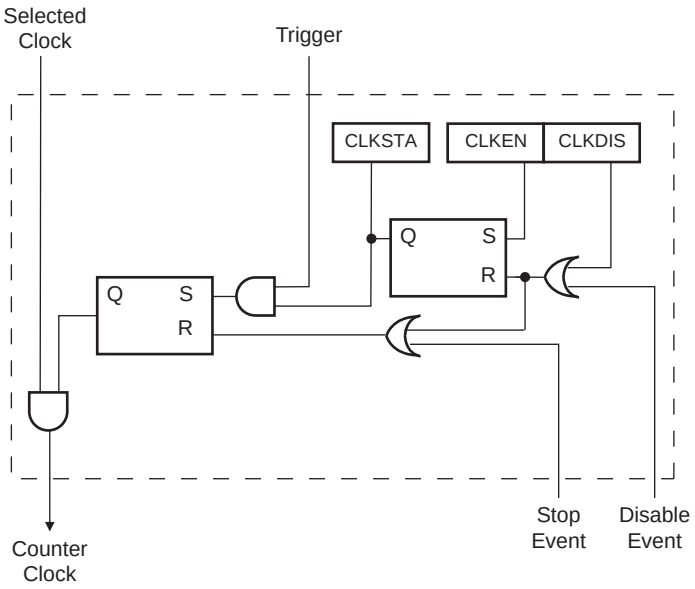


### 32.6.4 Clock Control

The clock of each counter can be controlled in two different ways: it can be enabled/disabled and started/stopped. See Figure 32-4.

- 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 32-4. Clock Control**




### 32.6.5 Operating Modes

Each channel can operate independently in two different modes:

- Capture mode provides measurement on signals.
- Waveform mode provides wave generation.

The TC operating mode is programmed with the WAVE bit in the TC_CMR.

In Capture mode, TIOA and TIOB are configured as inputs.

In Waveform mode, TIOA is always configured to be an output and TIOB is an output if it is not selected to be the external trigger.

### 32.6.6 Trigger

A trigger resets the counter and starts the counter clock. Three types of triggers are common to both modes, and a fourth external trigger is available to each mode.

Regardless of the trigger used, it will be taken into account at the following active edge of the selected clock. This means that the counter value can be read differently from zero just after a trigger, especially when a low frequency signal is selected as the clock.

The following triggers are common to both modes:

- Software Trigger: Each channel has a software trigger, available by setting SWTRG in TC_CCR.
- SYNC: Each channel has a synchronization signal SYNC. When asserted, this signal has the same effect as a software trigger. The SYNC signals of all channels are asserted simultaneously by writing TC_BCR (Block Control) with SYNC set.
- Compare RC Trigger: RC is implemented in each channel and can provide a trigger when the counter value matches the RC value if CPCTRG is set in the TC_CMR.

The channel can also be configured to have an external trigger. In Capture mode, the external trigger signal can be selected between TIOA and TIOB. In Waveform mode, an external event can be programmed on one of the following signals: TIOB, XC0, XC1 or XC2. This external event can then be programmed to perform a trigger by setting bit ENETRG in the TC_CMR.

If an external trigger is used, the duration of the pulses must be longer than the peripheral clock period in order to be detected.

### 32.6.7 Capture Mode

Capture mode is entered by clearing the WAVE bit in the TC_CMR.

Capture mode allows the TC channel to perform measurements such as pulse timing, frequency, period, duty cycle and phase on TIOA and TIOB signals which are considered as inputs.

Figure 32-5 shows the configuration of the TC channel when programmed in Capture mode.

### 32.6.8 Capture Registers A and B

Registers A and B (RA and RB) are used as capture registers. They can be loaded with the counter value when a programmable event occurs on the signal TIOA.

The LDRA field in the TC_CMR defines the TIOA selected edge for the loading of register A, and the LDRB field defines the TIOA selected edge for the loading of Register B.

RA is loaded only if it has not been loaded since the last trigger or if RB has been loaded since the last loading of RA.

RB is loaded only if RA has been loaded since the last trigger or the last loading of RB.

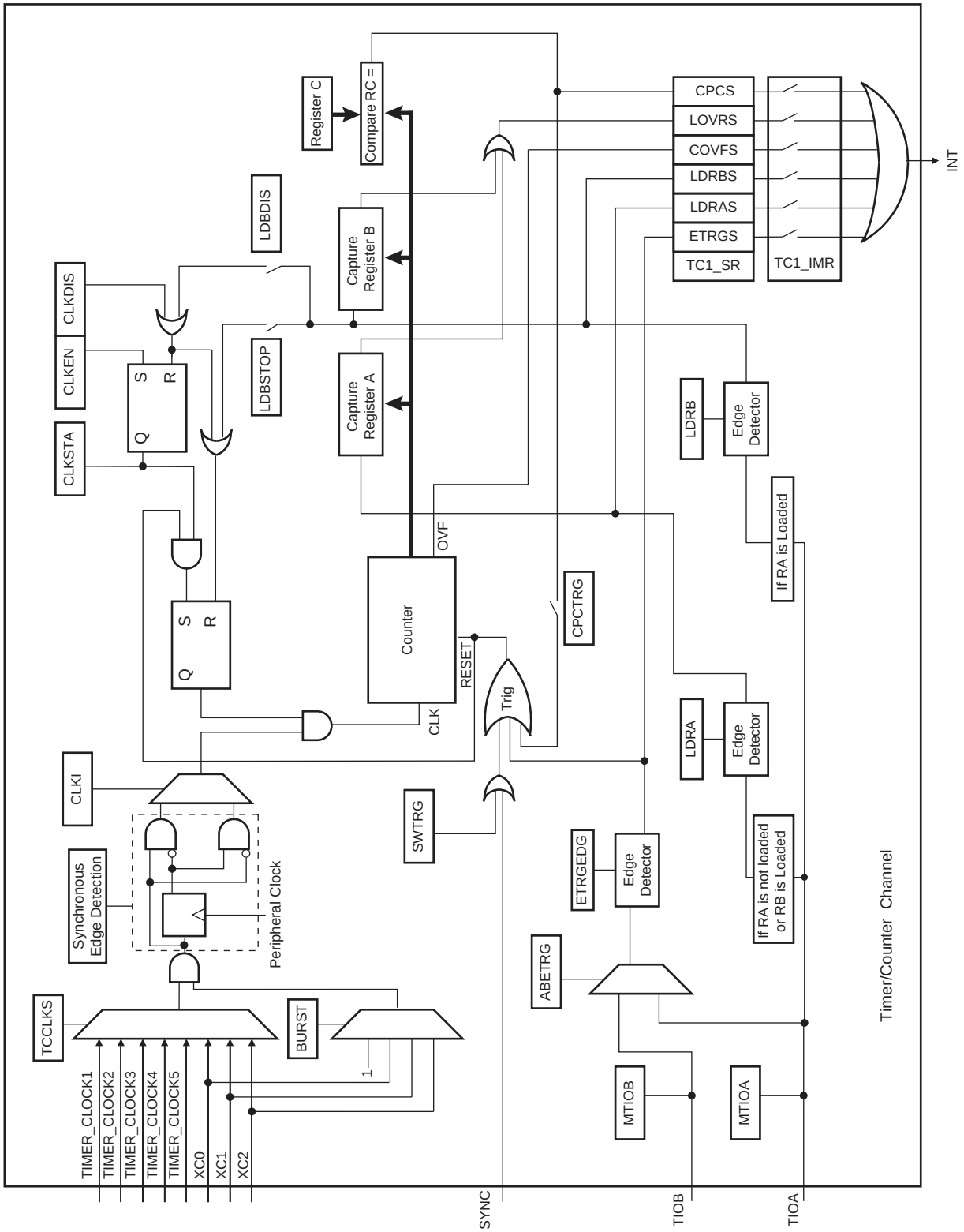
Loading RA or RB before the read of the last value loaded sets the Overrun Error Flag (LOVRS bit) in the TC_SR. In this case, the old value is overwritten.

### 32.6.9 Trigger Conditions

In addition to the SYNC signal, the software trigger and the RC compare trigger, an external trigger can be defined.

The ABETRG bit in the TC_CMR selects TIOA or TIOB input signal as an external trigger. The External Trigger Edge Selection parameter (ETRGEDG field in TC_CMR) defines the edge (rising, falling, or both) detected to generate an external trigger. If ETRGEDG = 0 (none), the external trigger is disabled.

**Figure 32-5. Capture Mode**



### 32.6.10 Waveform Mode

Waveform mode is entered by setting the TC_CMRx.WAVE bit.

In Waveform mode, the TC channel generates one or two PWM signals with the same frequency and independently programmable duty cycles, or generates different types of one-shot or repetitive pulses.

In this mode, TIOA is configured as an output and TIOB is defined as an output if it is not used as an external event (EEVT parameter in TC_CMR).

Figure 32-6 shows the configuration of the TC channel when programmed in Waveform operating mode.

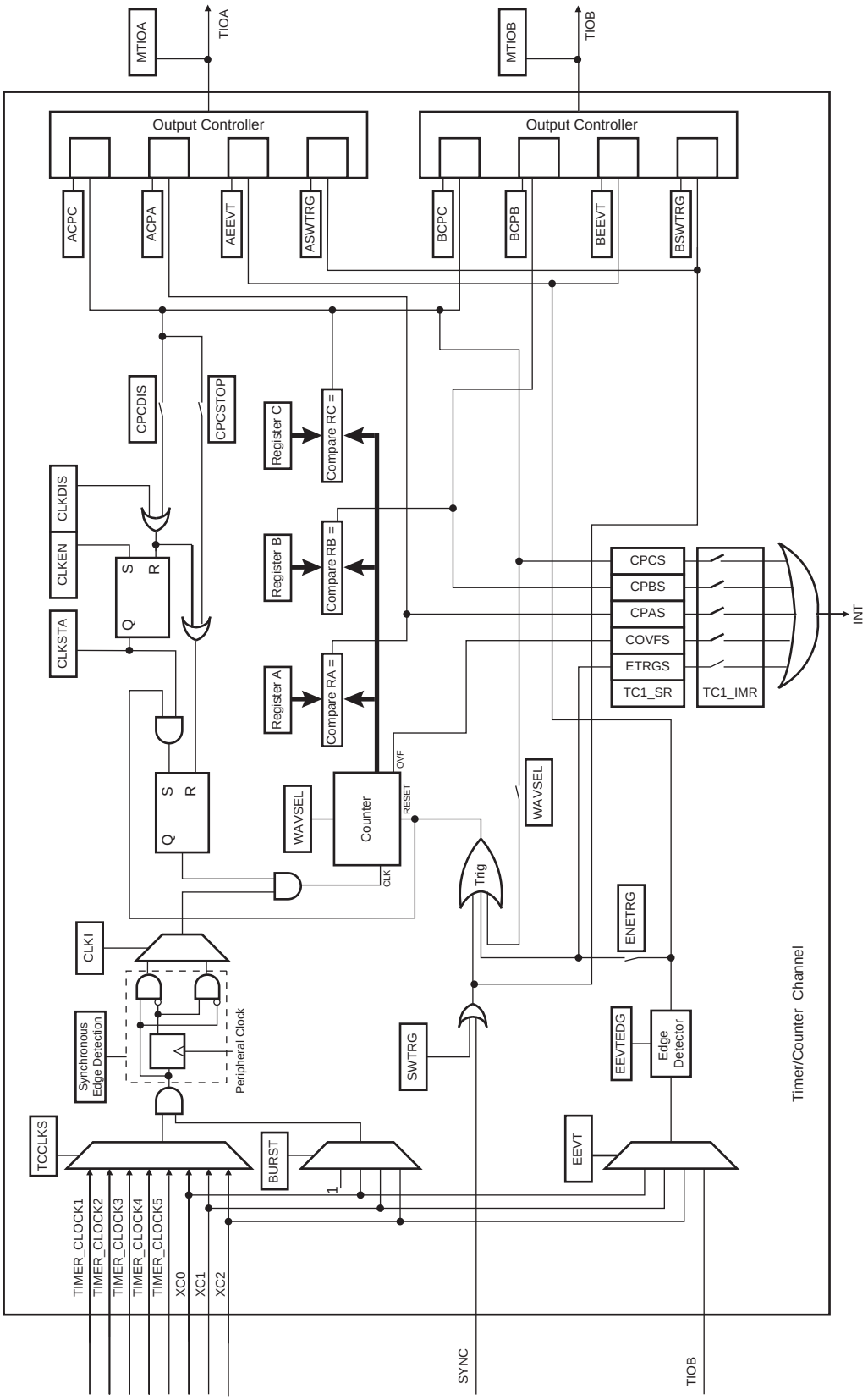
### 32.6.11 Waveform Selection

Depending on the WAVSEL parameter in TC_CMR, the behavior of TC_CV varies.

With any selection, TC_RA, TC_RB and TC_RC can all be used as compare registers.

RA Compare is used to control the TIOA output, RB Compare is used to control the TIOB output (if correctly configured) and RC Compare is used to control TIOA and/or TIOB outputs.

**Figure 32-6. Waveform Mode**



### 32.6.11.1 WAVSEL = 00

When WAVSEL = 00, the value of TC_CV is incremented from 0 to $2^{16}$-1. Once $2^{16}$-1 has been reached, the value of TC_CV is reset. Incrementation of TC_CV starts again and the cycle continues. See Figure 32-7.

An external event trigger or a software trigger can reset the value of TC_CV. It is important to note that the trigger may occur at any time. See Figure 32-8.

RC Compare cannot be programmed to generate a trigger in this configuration. At the same time, RC Compare can stop the counter clock (CPCSTOP = 1 in TC_CMR) and/or disable the counter clock (CPCDIS = 1 in TC_CMR).

**Figure 32-7. WAVSEL = 00 without Trigger**

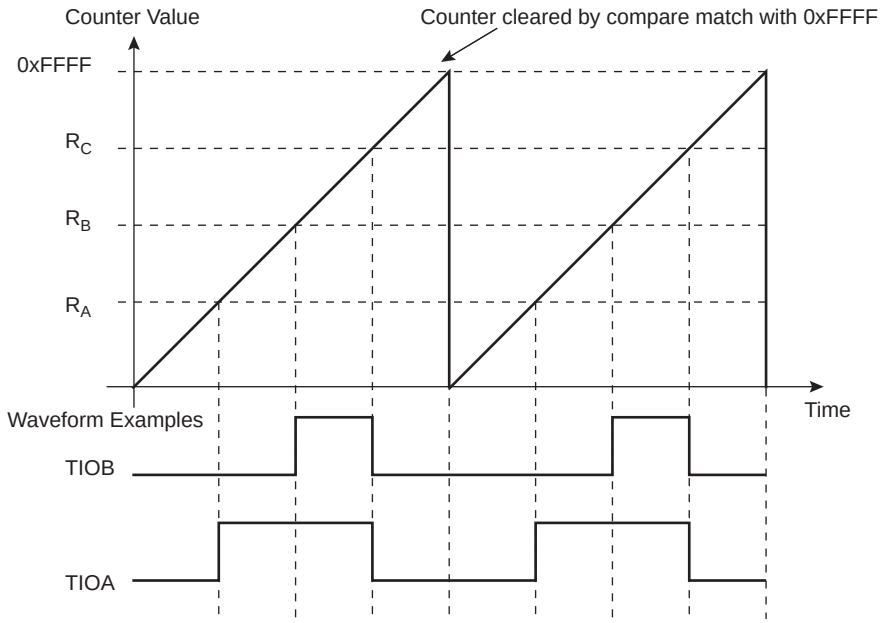
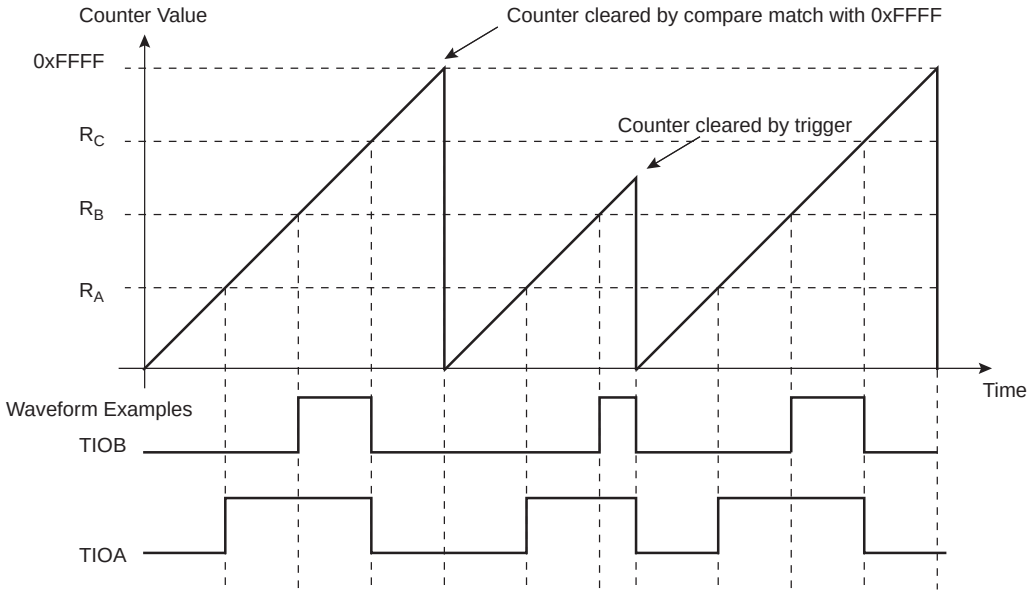



**Figure 32-8. WAVSEL = 00 with Trigger**



### 32.6.11.2 WAVSEL = 10

When WAVSEL = 10, the value of TC_CV is incremented from 0 to the value of RC, then automatically reset on a RC Compare. Once the value of TC_CV has been reset, it is then incremented and so on. See Figure 32-9.

It is important to note that TC_CV can be reset at any time by an external event or a software trigger if both are programmed correctly. See Figure 32-10.

In addition, RC Compare can stop the counter clock (CPCSTOP = 1 in TC_CMR) and/or disable the counter clock (CPCDIS = 1 in TC_CMR).

**Figure 32-9. WAVSEL = 10 without Trigger**

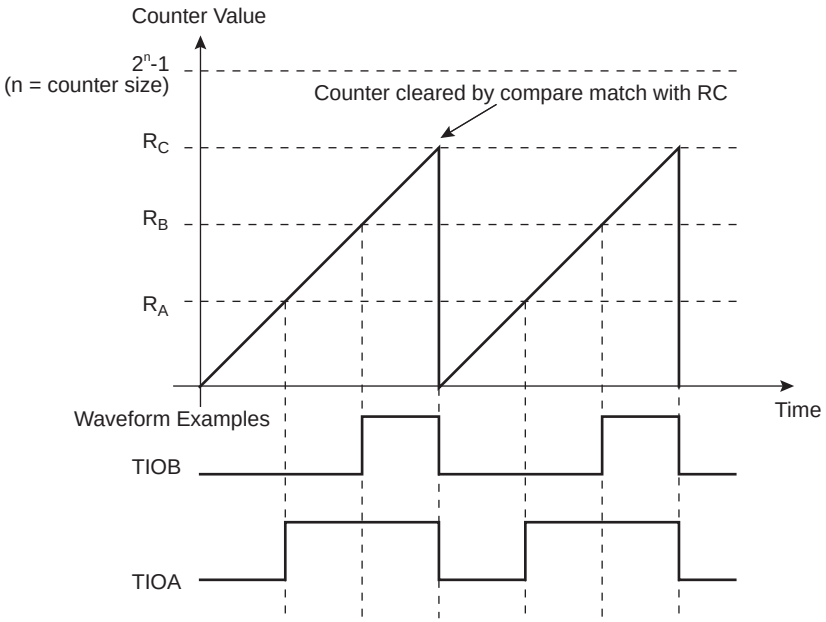
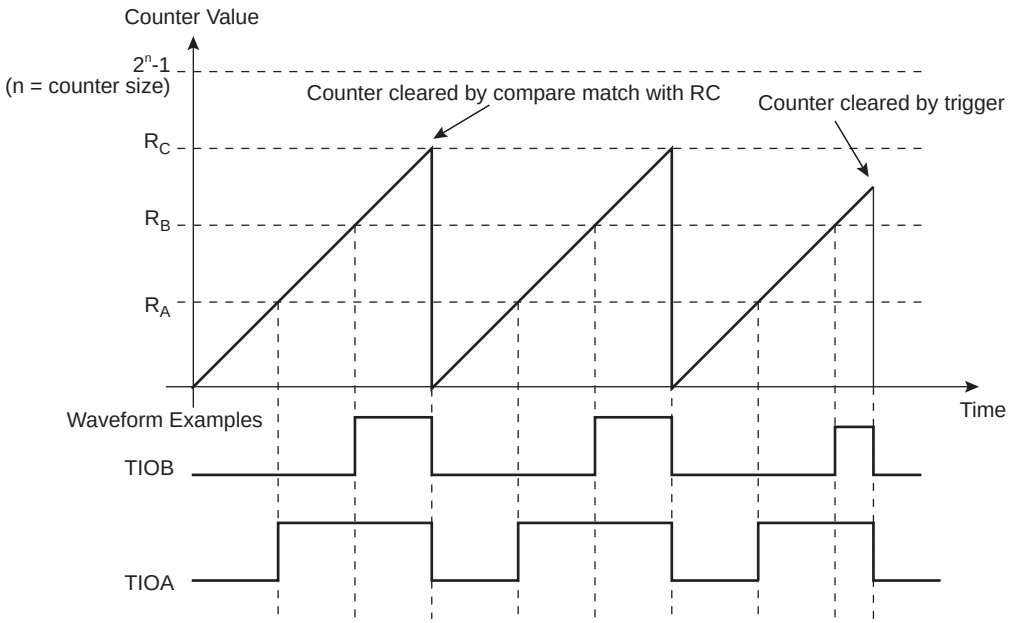



**Figure 32-10. WAVSEL = 10 with Trigger**



### 32.6.11.3 WAVSEL = 01

When WAVSEL = 01, the value of TC_CV is incremented from 0 to $2^{16}$-1 . Once $2^{16}$-1 is reached, the value of TC_CV is decremented to 0, then re-incremented to $2^{16}$-1 and so on. See Figure 32-11.

A trigger such as an external event or a software trigger can modify TC_CV at any time. If a trigger occurs while TC_CV is incrementing, TC_CV then decrements. If a trigger is received while TC_CV is decrementing, TC_CV then increments. See Figure 32-12.

RC Compare cannot be programmed to generate a trigger in this configuration.

At the same time, RC Compare can stop the counter clock (CPCSTOP = 1) and/or disable the counter clock (CPCDIS = 1).

**Figure 32-11. WAVSEL = 01 without Trigger**

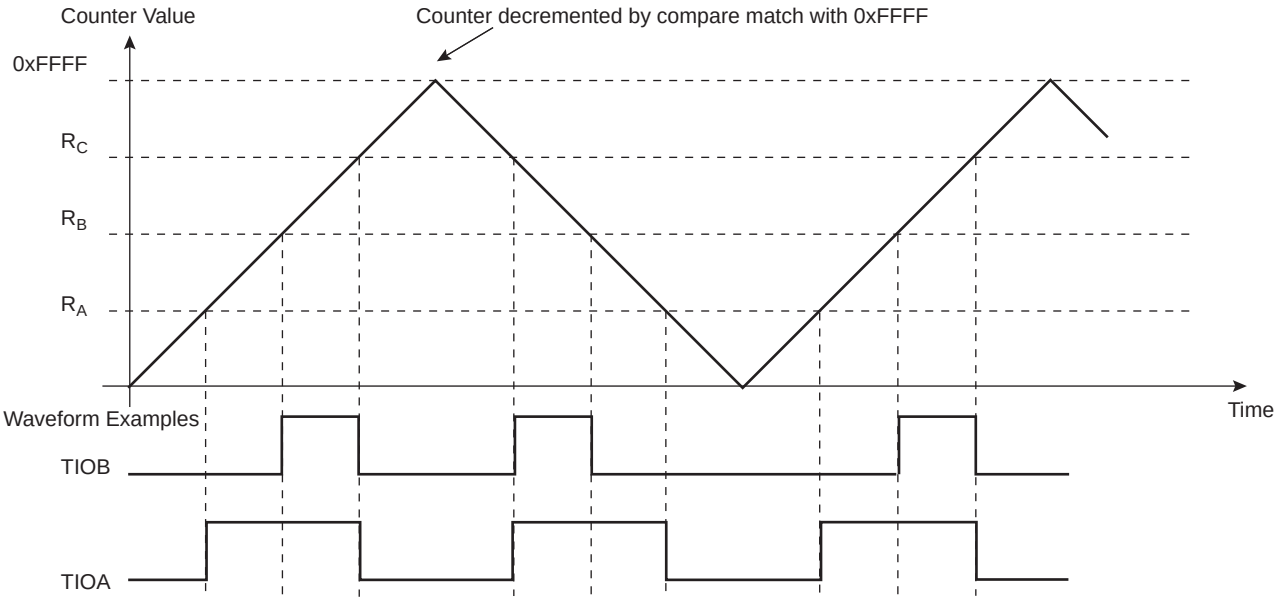
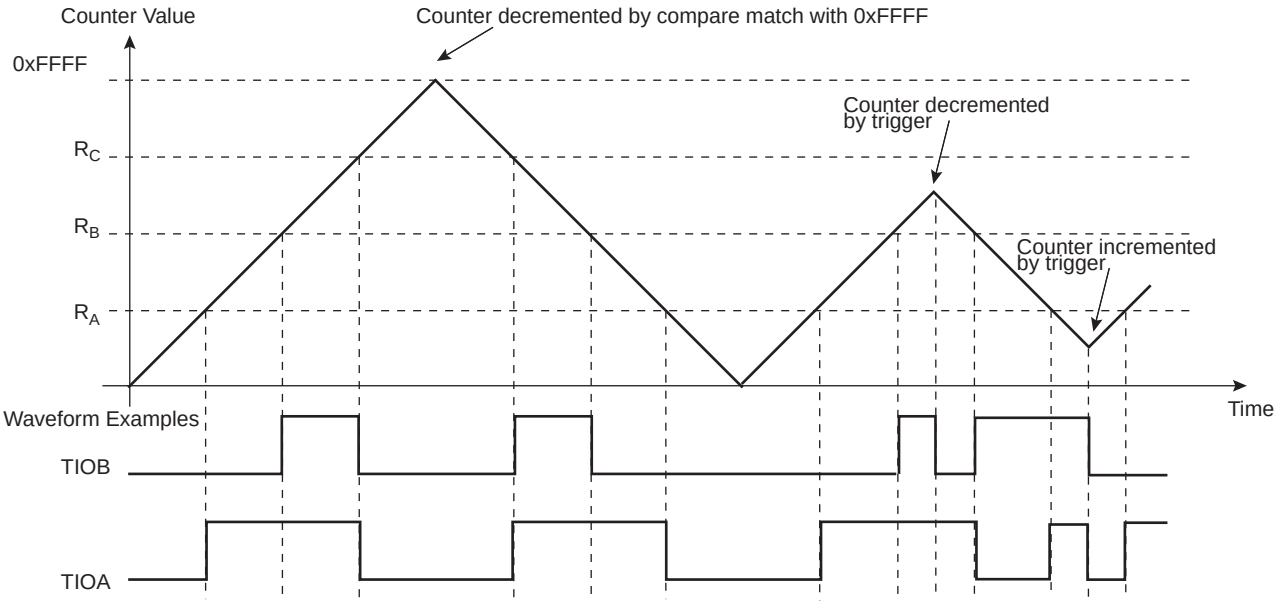



**Figure 32-12. WAVSEL = 01 with Trigger**



### 32.6.11.4 WAVSEL = 11

When WAVSEL = 11, the value of TC_CV is incremented from 0 to RC. Once RC is reached, the value of TC_CV is decremented to 0, then re-incremented to RC and so on. See Figure 32-13.

A trigger such as an external event or a software trigger can modify TC_CV at any time. If a trigger occurs while TC_CV is incrementing, TC_CV then decrements. If a trigger is received while TC_CV is decrementing, TC_CV then increments. See Figure 32-14.

RC Compare can stop the counter clock (CPCSTOP = 1) and/or disable the counter clock (CPCDIS = 1).

**Figure 32-13. WAVSEL = 11 without Trigger**

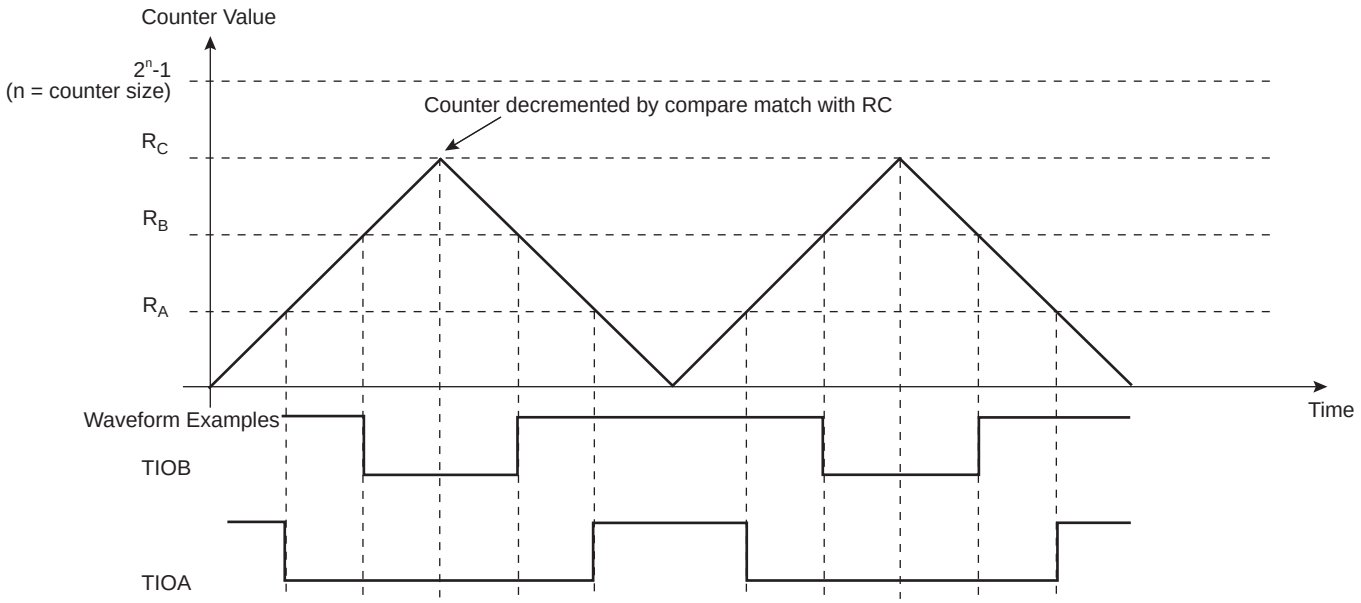
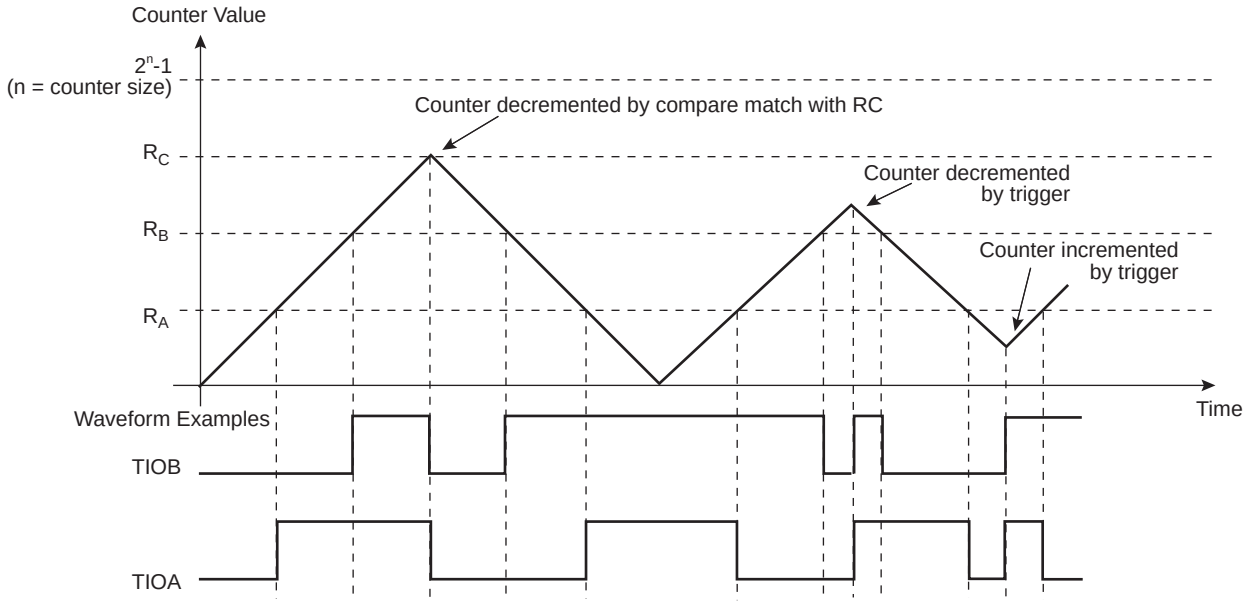



**Figure 32-14. WAVSEL = 11 with Trigger**



### 32.6.12 External Event/Trigger Conditions

An external event can be programmed to be detected on one of the clock sources (XC0, XC1, XC2) or TIOB. The external event selected can then be used as a trigger.

The EEVT parameter in TC_CMR selects the external trigger. The EEVTEDG parameter defines the trigger edge for each of the possible external triggers (rising, falling or both). If EEVTEDG is cleared (none), no external event is defined.

If TIOB is defined as an external event signal (EEVT = 0), TIOB is no longer used as an output and the compare register B is not used to generate waveforms and subsequently no IRQs. In this case the TC channel can only generate a waveform on TIOA.

When an external event is defined, it can be used as a trigger by setting bit ENETRG in the TC_CMR.

As in Capture mode, the SYNC signal and the software trigger are also available as triggers. RC Compare can also be used as a trigger depending on the parameter WAVSEL.

### 32.6.13 Output Controller

The output controller defines the output level changes on TIOA and TIOB following an event. TIOB control is used only if TIOB is defined as output (not as an external event).

The following events control TIOA and TIOB: software trigger, external event and RC compare. RA compare controls TIOA and RB compare controls TIOB. Each of these events can be programmed to set, clear or toggle the output as defined in the corresponding parameter in TC_CMR.

### 32.6.14 Quadrature Decoder

#### 32.6.14.1 Description

The quadrature decoder (QDEC) is driven by TIOA0, TIOB0, TIOB1 input pins and drives the timer/counter of channel 0 and 1. Channel 2 can be used as a time base in case of speed measurement requirements (refer to Figure 32-15).

When writing a 0 to bit QDEN of the TC_BMR, the QDEC is bypassed and the IO pins are directly routed to the timer counter function. See

TIOA0 and TIOB0 are to be driven by the two dedicated quadrature signals from a rotary sensor mounted on the shaft of the off-chip motor.

A third signal from the rotary sensor can be processed through pin TIOB1 and is typically dedicated to be driven by an index signal if it is provided by the sensor. This signal is not required to decode the quadrature signals PHA, PHB.

Field TCCLKS of TC_CMRx must be configured to select XC0 input (i.e., 0x101). Field TC0XC0S has no effect as soon as the QDEC is enabled.

Either speed or position/revolution can be measured. Position channel 0 accumulates the edges of PHA, PHB input signals giving a high accuracy on motor position whereas channel 1 accumulates the index pulses of the sensor, therefore the number of rotations. Concatenation of both values provides a high level of precision on motion system position.

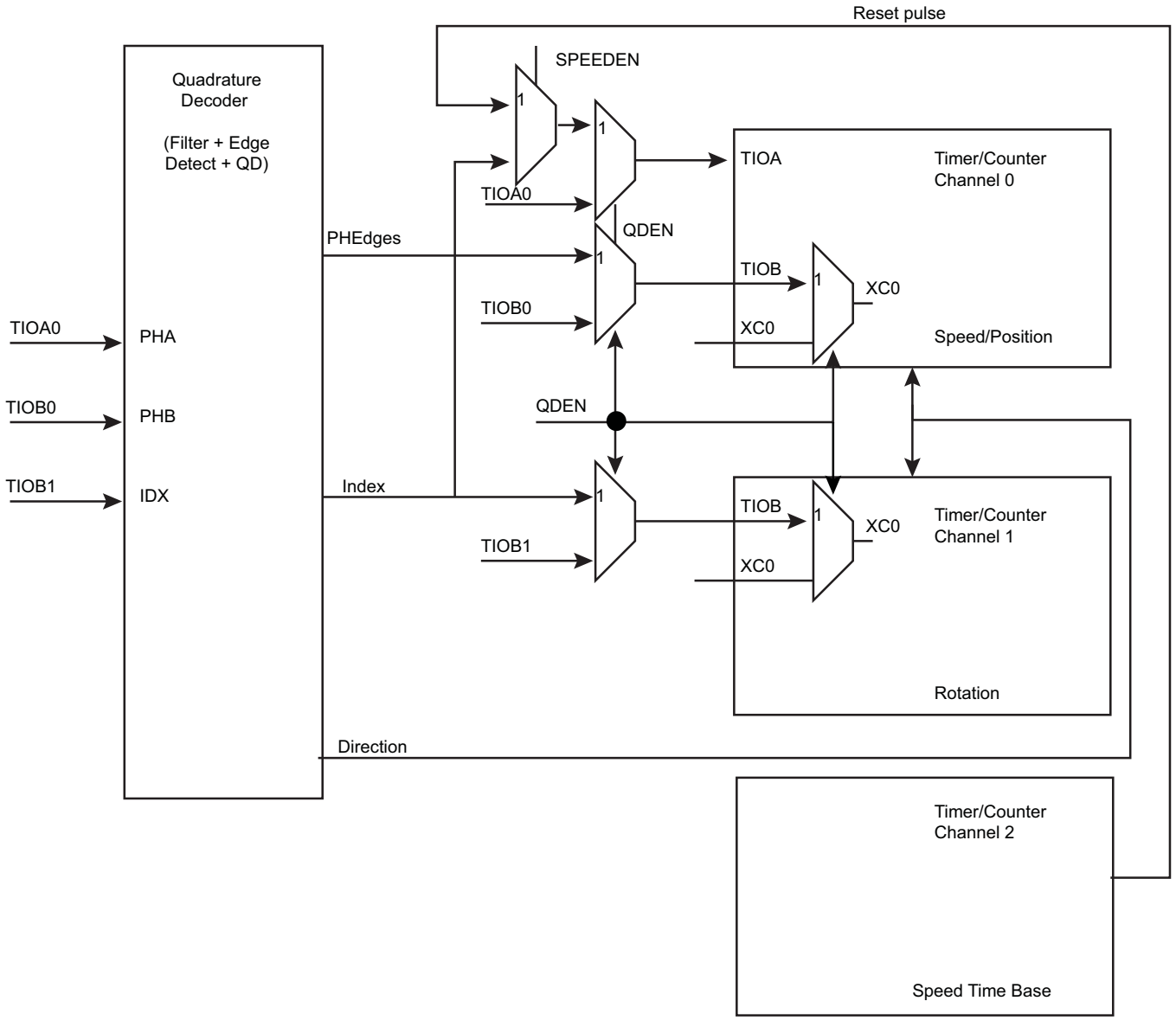
In Speed mode, position cannot be measured but revolution can be measured.

Inputs from the rotary sensor can be filtered prior to down-stream processing. Accommodation of input polarity, phase definition and other factors are configurable.

Interruptions can be generated on different events.

A compare function (using TC_RC) is available on channel 0 (speed/position) or channel 1 (rotation) and can generate an interrupt by means of the CPCS flag in the TC_SRx.

**Figure 32-15. Predefined Connection of the Quadrature Decoder with Timer Counters**



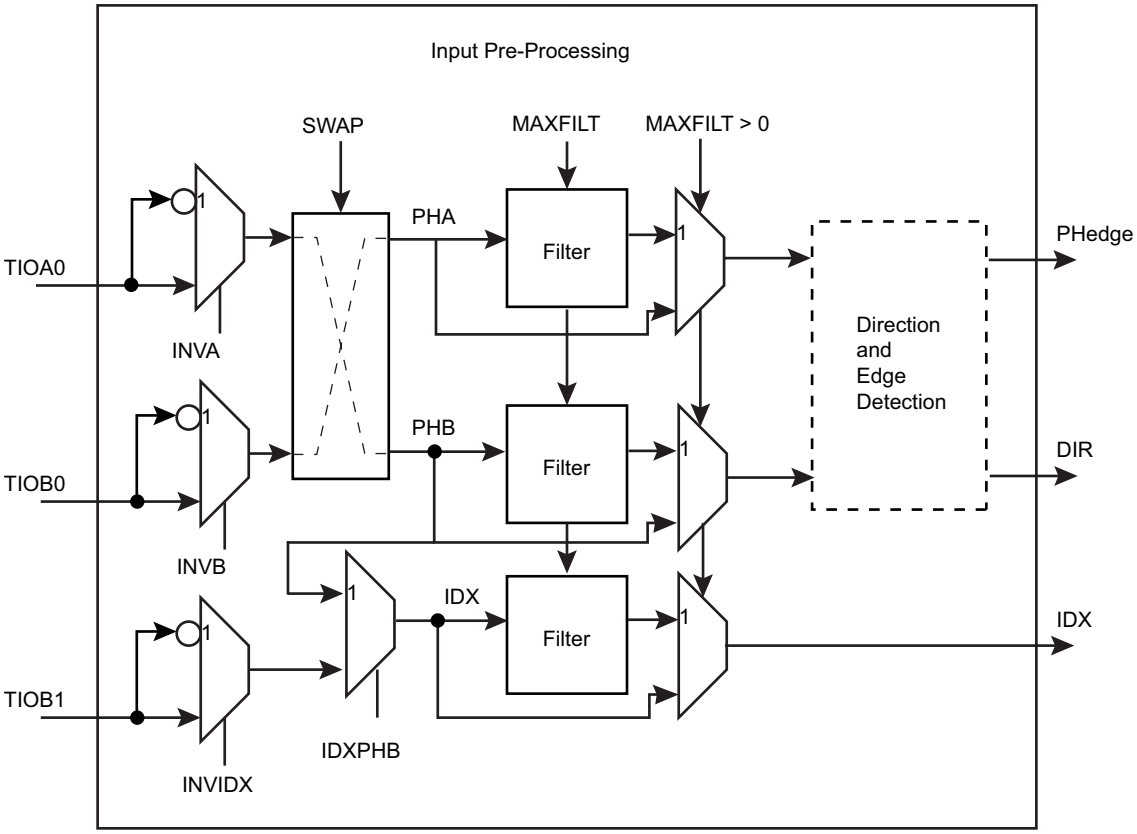

### 32.6.14.2 Input Pre-processing

Input pre-processing consists of capabilities to take into account rotary sensor factors such as polarities and phase definition followed by configurable digital filtering.

Each input can be negated and swapping PHA, PHB is also configurable.

The MAXFILT field in the TC_BMR is used to configure a minimum duration for which the pulse is stated as valid. When the filter is active, pulses with a duration lower than MAXFILT +1 $\times$ $t_{peripheral\ clock}$ ns are not passed to down-stream logic.

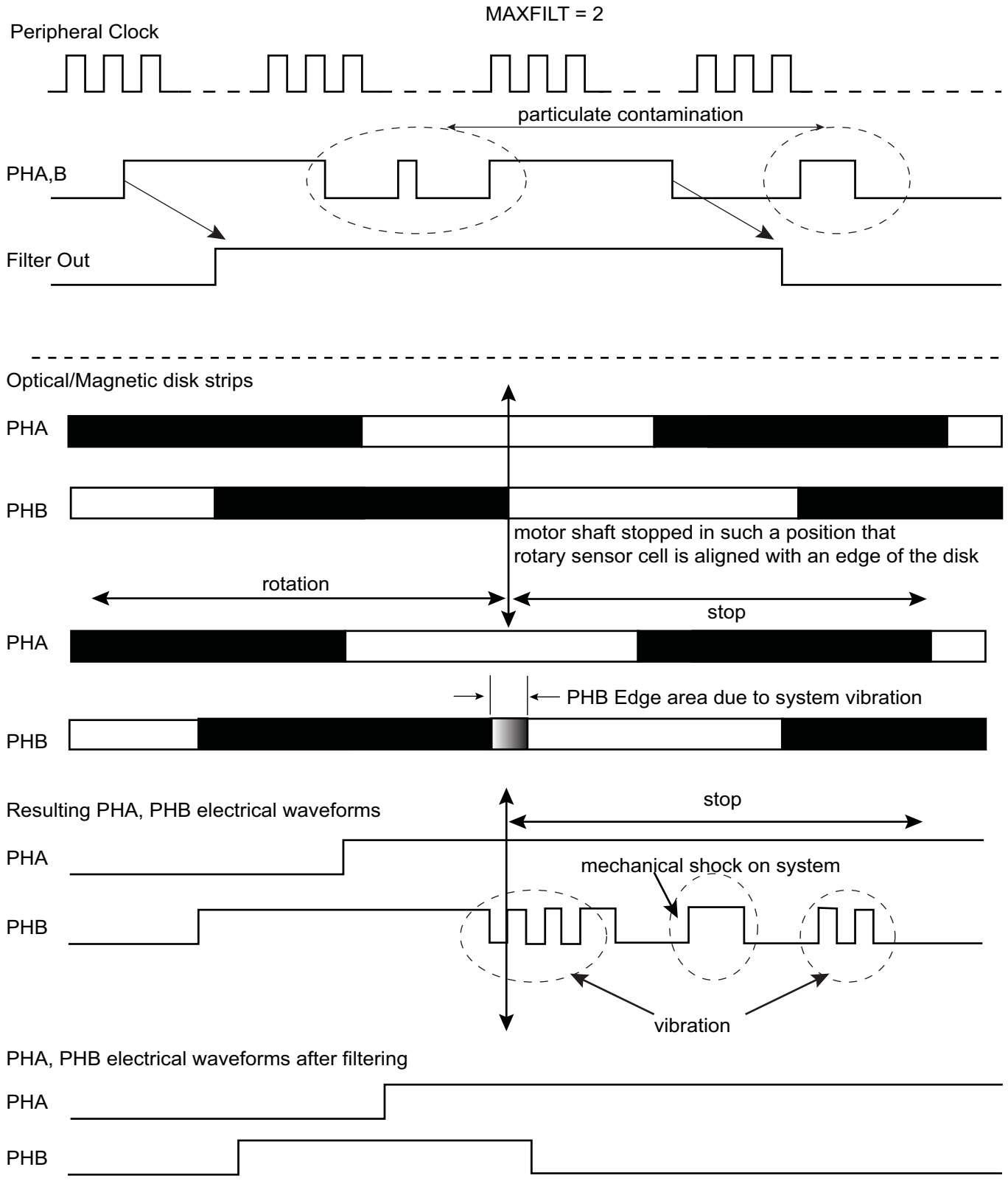
**Figure 32-16. Input Stage**




Input filtering can efficiently remove spurious pulses that might be generated by the presence of particulate contamination on the optical or magnetic disk of the rotary sensor.

Spurious pulses can also occur in environments with high levels of electro-magnetic interference. Or, simply if vibration occurs even when rotation is fully stopped and the shaft of the motor is in such a position that the beginning of one of the reflective or magnetic bars on the rotary sensor disk is aligned with the light or magnetic (Hall) receiver cell of the rotary sensor. Any vibration can make the PHA, PHB signals toggle for a short duration.

**Figure 32-17. Filtering Examples**



### 32.6.14.3 Direction Status and Change Detection

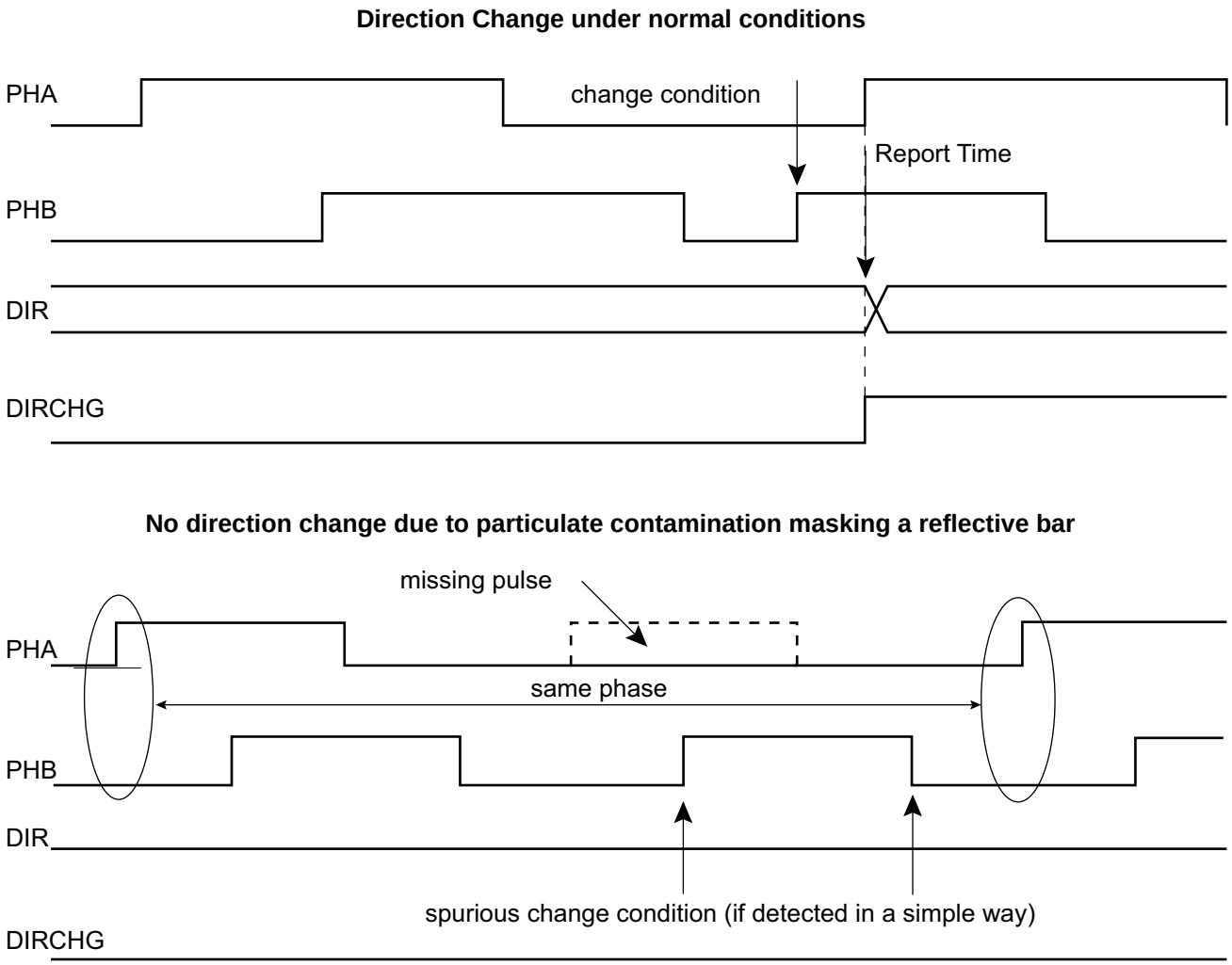
After filtering, the quadrature signals are analyzed to extract the rotation direction and edges of the two quadrature signals detected in order to be counted by timer/counter logic downstream.

The direction status can be directly read at anytime in the TC_QISR. The polarity of the direction flag status depends on the configuration written in TC_BMR. INVA, INVB, INVIDX, SWAP modify the polarity of DIR flag.

Any change in rotation direction is reported in the TC_QISR and can generate an interrupt.

The direction change condition is reported as soon as two consecutive edges on a phase signal have sampled the same value on the other phase signal and there is an edge on the other signal. The two consecutive edges of one phase signal sampling the same value on other phase signal is not sufficient to declare a direction change, for the reason that particulate contamination may mask one or more reflective bars on the optical or magnetic disk of the sensor. Refer to Figure 32-18 for waveforms.

**Figure 32-18. Rotation Change Detection**

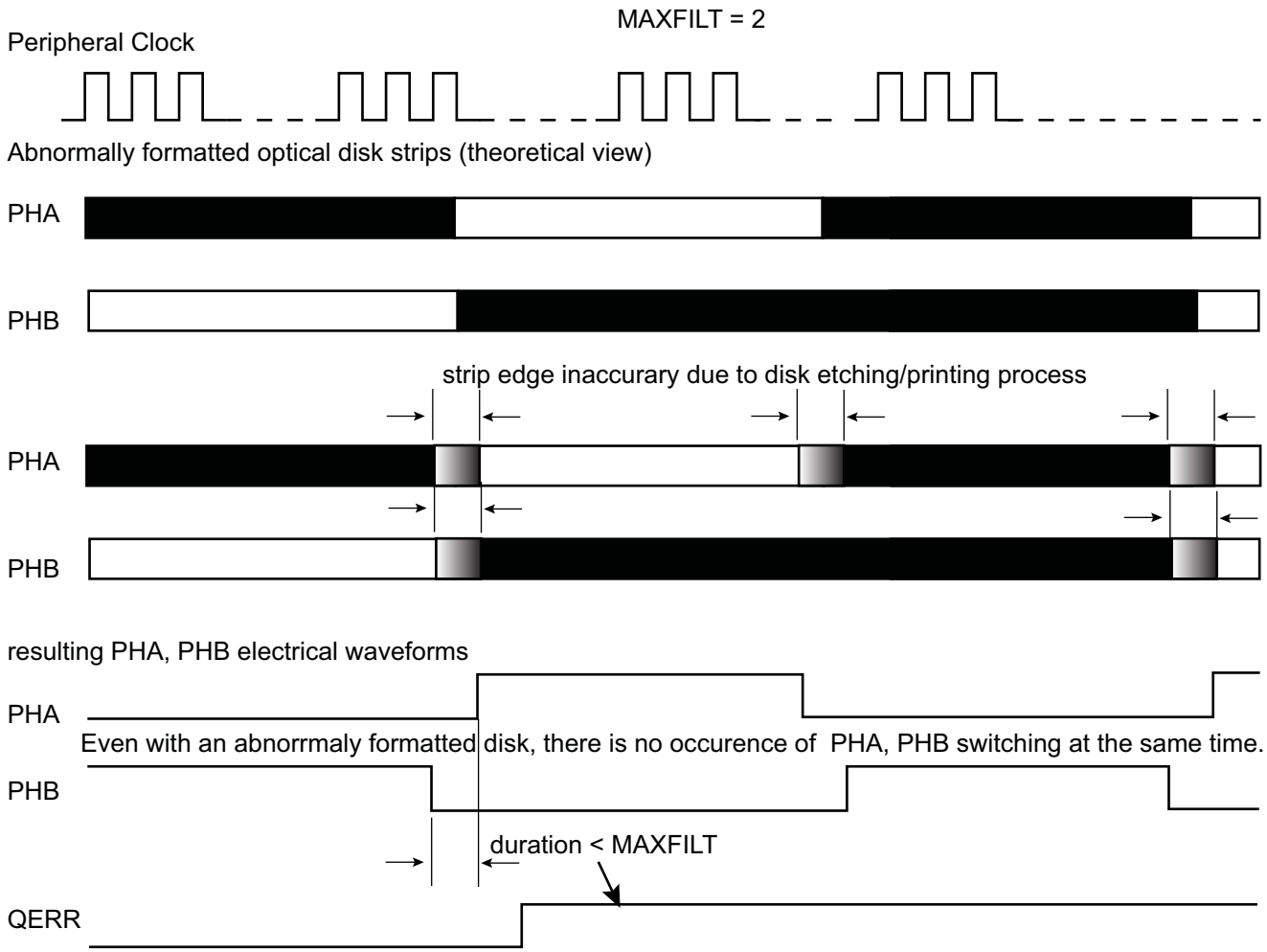



The direction change detection is disabled when QDTRANS is set in the TC_BMR. In this case, the DIR flag report must not be used.

A quadrature error is also reported by the QDEC via the QERR flag in the TC_QISR. This error is reported if the time difference between two edges on PHA, PHB is lower than a predefined value. This predefined value is

configurable and corresponds to (MAXFILT + 1) × $t_{peripheral\ clock}$ ns. After being filtered there is no reason to have two edges closer than (MAXFILT + 1) × $t_{peripheral\ clock}$ ns under normal mode of operation.

**Figure 32-19. Quadrature Error Detection**




MAXFILT must be tuned according to several factors such as the peripheral clock frequency, type of rotary sensor and rotation speed to be achieved.

### 32.6.14.4 Position and Rotation Measurement

When the POSEN bit is set in the TC_BMR, the motor axis position is processed on channel 0 (by means of the PHA, PHB edge detections) and the number of motor revolutions are recorded on channel 1 if the IDX signal is provided on the TIOB1 input. The position measurement can be read in the TC_CV0 register and the rotation measurement can be read in the TC_CV1 register.

Channel 0 and 1 must be configured in Capture mode (TC_CMR0.WAVE = 0). ‘Rising edge’ must be selected as the External Trigger Edge (TC_CMR.ETRGEDG = 0x01) and ‘TIOA’ must be selected as the External Trigger (TC_CMR.ABETRG = 0x1).

In parallel, the number of edges are accumulated on timer/counter channel 0 and can be read on the TC_CV0 register.

Therefore, the accurate position can be read on both TC_CV registers and concatenated to form a 32-bit word.

The timer/counter channel 0 is cleared for each increment of IDX count value.

Depending on the quadrature signals, the direction is decoded and allows to count up or down in timer/counter channels 0 and 1. The direction status is reported on TC_QISR.

#### 32.6.14.5 Speed Measurement

When SPEEDEN is set in the TC_BMR, the speed measure is enabled on channel 0.

A time base must be defined on channel 2 by writing the TC_RC2 period register. Channel 2 must be configured in Waveform mode (WAVE bit set) in TC_CMR2. The WAVSEL field must be defined with 0x10 to clear the counter by comparison and matching with TC_RC value. Field ACPC must be defined at 0x11 to toggle TIOA output.

This time base is automatically fed back to TIOA of channel 0 when QDEN and SPEEDEN are set.

Channel 0 must be configured in Capture mode (WAVE = 0 in TC_CMR0). The ABETRG bit of TC_CMR0 must be configured at 1 to select TIOA as a trigger for this channel.

EDGTRG must be set to 0x01, to clear the counter on a rising edge of the TIOA signal and field LDRA must be set accordingly to 0x01, to load TC_RA0 at the same time as the counter is cleared (LDRB must be set to 0x01). As a consequence, at the end of each time base period the differentiation required for the speed calculation is performed.

The process must be started by configuring bits CLKEN and SWTRG in the TC_CCR.

The speed can be read on field RA in TC_RA0.

Channel 1 can still be used to count the number of revolutions of the motor.

### 32.6.15 2-bit Gray Up/Down Counter for Stepper Motor

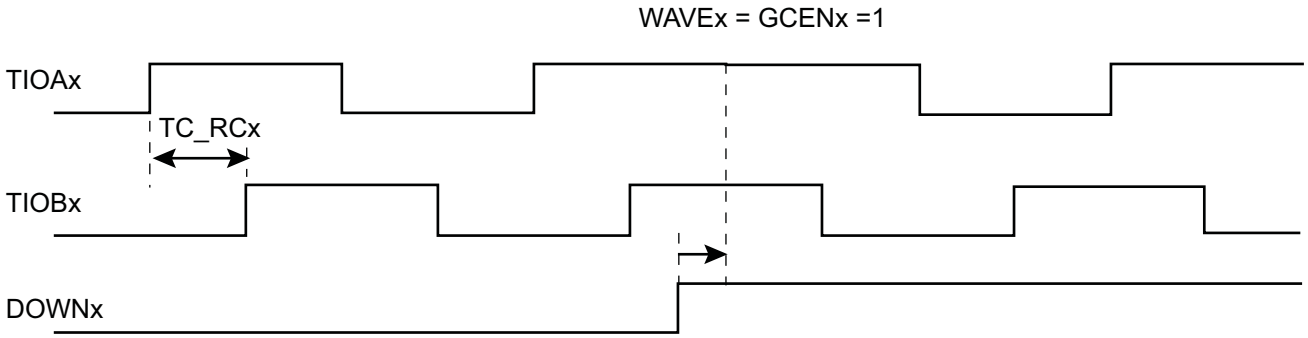
Each channel can be independently configured to generate a 2-bit gray count waveform on corresponding TIOA, TIOB outputs by means of the GCEN bit in TC_SMMRx.

Up or Down count can be defined by writing bit DOWN in TC_SMMRx.

It is mandatory to configure the channel in Waveform mode in the TC_CMR.

The period of the counters can be programmed in TC_RCx.

**Figure 32-20. 2-bit Gray Up/Down Counter**



### 32.6.16 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

## 32.7 Timer Counter (TC) User Interface

**Table 32-6. Register Mapping**

| Offset[1] | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x00 + channel * 0x40 + 0x00 | Channel Control Register | TC_CCR | Write-only | – |
| 0x00 + channel * 0x40 + 0x04 | Channel Mode Register | TC_CMR | Read/Write | 0 |
| 0x00 + channel * 0x40 + 0x08 | Stepper Motor Mode Register | TC_SMMR | Read/Write | 0 |
| 0x00 + channel * 0x40 + 0x0C | Reserved | – | – | – |
| 0x00 + channel * 0x40 + 0x10 | Counter Value | TC_CV | Read-only | 0 |
| 0x00 + channel * 0x40 + 0x14 | Register A | TC_RA | Read/Write[2] | 0 |
| 0x00 + channel * 0x40 + 0x18 | Register B | TC_RB | Read/Write[2] | 0 |
| 0x00 + channel * 0x40 + 0x1C | Register C | TC_RC | Read/Write | 0 |
| 0x00 + channel * 0x40 + 0x20 | Status Register | TC_SR | Read-only | 0 |
| 0x00 + channel * 0x40 + 0x24 | Interrupt Enable Register | TC_IER | Write-only | – |
| 0x00 + channel * 0x40 + 0x28 | Interrupt Disable Register | TC_IDR | Write-only | – |
| 0x00 + channel * 0x40 + 0x2C | Interrupt Mask Register | TC_IMR | Read-only | 0 |
| 0xC0 | Block Control Register | TC_BCR | Write-only | – |
| 0xC4 | Block Mode Register | TC_BMR | Read/Write | 0 |
| 0xC8 | QDEC Interrupt Enable Register | TC_QIER | Write-only | – |
| 0xCC | QDEC Interrupt Disable Register | TC_QIDR | Write-only | – |
| 0xD0 | QDEC Interrupt Mask Register | TC_QIMR | Read-only | 0 |
| 0xD4 | QDEC Interrupt Status Register | TC_QISR | Read-only | 0 |
| 0xD8 | Reserved | – | – | – |
| 0xE4 | Write Protection Mode Register | TC_WPMR | Read/Write | 0 |
| 0xE8–0xFC | Reserved | – | – | – |

Notes: 1. Channel index ranges from 0 to 2.
2. Read-only if TC_CMRx.WAVE = 0

### 32.7.1 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.

### 32.7.2 TC Channel Mode Register: Capture Mode

**Name:** TC_CMRx [x=0..2] (CAPTURE_MODE)

**Address:** 0x40010004 (0)[0], 0x40010044 (0)[1], 0x40010084 (0)[2], 0x40014004 (1)[0], 0x40014044 (1)[1], 0x40014084 (1)[2]

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | LDRB | | LDRA | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| WAVE | CPCTRG | – | – | – | ABETRG | ETRGEDG | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| LDBDIS | LDBSTOP | BURST | | CLKI | TCCLKS | | |

This register can only be written if the WPEN bit is cleared in the TC Write Protection Mode Register.

- **TCCLKS: Clock Selection**

| Value | Name | Description |
|---|---|---|
| 0 | TIMER_CLOCK1 | Clock selected: internal MCK/2 clock signal (from PMC) |
| 1 | TIMER_CLOCK2 | Clock selected: internal MCK/8 clock signal (from PMC) |
| 2 | TIMER_CLOCK3 | Clock selected: internal MCK/32 clock signal (from PMC) |
| 3 | TIMER_CLOCK4 | Clock selected: internal MCK/128 clock signal (from PMC) |
| 4 | TIMER_CLOCK5 | Clock selected: internal SLCK clock signal (from PMC) |
| 5 | XC0 | Clock selected: XC0 |
| 6 | XC1 | Clock selected: XC1 |
| 7 | XC2 | Clock selected: XC2 |

- **CLKI: Clock Invert**

0: Counter is incremented on rising edge of the clock.

1: Counter is incremented on falling edge of the clock.

- **BURST: Burst Signal Selection**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | The clock is not gated by an external signal. |
| 1 | XC0 | XC0 is ANDed with the selected clock. |
| 2 | XC1 | XC1 is ANDed with the selected clock. |
| 3 | XC2 | XC2 is ANDed with the selected clock. |

- **LDBSTOP: Counter Clock Stopped with RB Loading**

0: Counter clock is not stopped when RB loading occurs.

1: Counter clock is stopped when RB loading occurs.

- **LDBDIS: Counter Clock Disable with RB Loading**

0: Counter clock is not disabled when RB loading occurs.

1: Counter clock is disabled when RB loading occurs.

- **ETRGEDG: External Trigger Edge Selection**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | The clock is not gated by an external signal. |
| 1 | RISING | Rising edge |
| 2 | FALLING | Falling edge |
| 3 | EDGE | Each edge |

- **ABETRG: TIOA or TIOB External Trigger Selection**

0: TIOB is used as an external trigger.

1: TIOA is used as an external trigger.

- **CPCTRG: RC Compare Trigger Enable**

0: RC Compare has no effect on the counter and its clock.

1: RC Compare resets the counter and starts the counter clock.

- **WAVE: Waveform Mode**

0: Capture mode is enabled.

1: Capture mode is disabled (Waveform mode is enabled).

- **LDRA: RA Loading Edge Selection**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | None |
| 1 | RISING | Rising edge of TIOA |
| 2 | FALLING | Falling edge of TIOA |
| 3 | EDGE | Each edge of TIOA |

- **LDRB: RB Loading Edge Selection**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | None |
| 1 | RISING | Rising edge of TIOA |
| 2 | FALLING | Falling edge of TIOA |
| 3 | EDGE | Each edge of TIOA |

### 32.7.3 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 SLCK clock signal (from PMC) |
| 5 | XC0 | Clock selected: XC0 |
| 6 | XC1 | Clock selected: XC1 |
| 7 | XC2 | Clock selected: XC2 |

- **CLKI: Clock Invert**

0: Counter is incremented on rising edge of the clock.

1: Counter is incremented on falling edge of the clock.

- **BURST: Burst Signal Selection**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | The clock is not gated by an external signal. |
| 1 | XC0 | XC0 is ANDed with the selected clock. |
| 2 | XC1 | XC1 is ANDed with the selected clock. |
| 3 | XC2 | XC2 is ANDed with the selected clock. |

- **CPCSTOP: Counter Clock Stopped with RC Compare**

0: Counter clock is not stopped when counter reaches RC.

1: Counter clock is stopped when counter reaches RC.

- **CPCDIS: Counter Clock Disable with RC Compare**

0: Counter clock is not disabled when counter reaches RC.

1: Counter clock is disabled when counter reaches RC.

- **EEVTEDG: External Event Edge Selection**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | None |
| 1 | RISING | Rising edge |
| 2 | FALLING | Falling edge |
| 3 | EDGE | Each edge |

- **EEVT: External Event Selection**

Signal selected as external event.

| Value | Name | Description | TIOB Direction |
|---|---|---|---|
| 0 | TIOB | TIOB(1) | Input |
| 1 | XC0 | XC0 | Output |
| 2 | XC1 | XC1 | Output |
| 3 | XC2 | XC2 | Output |

Note: 1. If TIOB is chosen as the external event signal, it is configured as an input and no longer generates waveforms and subsequently no IRQs.

- **ENETRG: External Event Trigger Enable**

0: The external event has no effect on the counter and its clock.

1: The external event resets the counter and starts the counter clock.

Note: Whatever the value programmed in ENETRG, the selected external event only controls the TIOA output and TIOB if not used as input (trigger event input or other input used).

- **WAVSEL: Waveform Selection**

| Value | Name | Description |
|---|---|---|
| 0 | UP | UP mode without automatic trigger on RC Compare |
| 1 | UPDOWN | UPDOWN mode without automatic trigger on RC Compare |
| 2 | UP_RC | UP mode with automatic trigger on RC Compare |
| 3 | UPDOWN_RC | UPDOWN mode with automatic trigger on RC Compare |

- **WAVE: Waveform Mode**

0: Waveform mode is disabled (Capture mode is enabled).

1: Waveform mode is enabled.

- **ACPA: RA Compare Effect on TIOA**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | None |
| 1 | SET | Set |
| 2 | CLEAR | Clear |
| 3 | TOGGLE | Toggle |

- **ACPC: RC Compare Effect on TIOA**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | None |
| 1 | SET | Set |
| 2 | CLEAR | Clear |
| 3 | TOGGLE | Toggle |

- **AEEVT: External Event Effect on TIOA**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | None |
| 1 | SET | Set |
| 2 | CLEAR | Clear |
| 3 | TOGGLE | Toggle |

- **ASWTRG: Software Trigger Effect on TIOA**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | None |
| 1 | SET | Set |
| 2 | CLEAR | Clear |
| 3 | TOGGLE | Toggle |

- **BCPB: RB Compare Effect on TIOB**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | None |
| 1 | SET | Set |
| 2 | CLEAR | Clear |
| 3 | TOGGLE | Toggle |

- **BCPC: RC Compare Effect on TIOB**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | None |
| 1 | SET | Set |
| 2 | CLEAR | Clear |
| 3 | TOGGLE | Toggle |

- **BEEVT: External Event Effect on TIOB**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | None |
| 1 | SET | Set |
| 2 | CLEAR | Clear |
| 3 | TOGGLE | Toggle |

- **BSWTRG: Software Trigger Effect on TIOB**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | None |
| 1 | SET | Set |
| 2 | CLEAR | Clear |
| 3 | TOGGLE | Toggle |

### 32.7.4 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.

### 32.7.5 TC Counter Value Register

**Name:** TC_CVx [x=0..2]

**Address:** 0x40010010 (0)[0], 0x40010050 (0)[1], 0x40010090 (0)[2], 0x40014010 (1)[0], 0x40014050 (1)[1], 0x40014090 (1)[2]

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| CV | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| CV | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CV | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CV | | | | | | | |

- **CV: Counter Value**

CV contains the counter value in real time.

IMPORTANT: For 16-bit channels, CV field size is limited to register bits 15:0.

### 32.7.6 TC Register A

**Name:** TC_RAx [x=0..2]

**Address:** 0x40010014 (0)[0], 0x40010054 (0)[1], 0x40010094 (0)[2], 0x40014014 (1)[0], 0x40014054 (1)[1], 0x40014094 (1)[2]

**Access:** Read-only if TC_CMRx.WAVE = 0, Read/Write if TC_CMRx.WAVE = 1

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RA | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| RA | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| RA | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| RA | | | | | | | |

This register can only be written if the WPEN bit is cleared in the TC Write Protection Mode Register.

- **RA: Register A**

RA contains the Register A value in real time.

IMPORTANT: For 16-bit channels, RA field size is limited to register bits 15:0.

### 32.7.7 TC Register B

**Name:** TC_RBx [x=0..2]

**Address:** 0x40010018 (0)[0], 0x40010058 (0)[1], 0x40010098 (0)[2], 0x40014018 (1)[0], 0x40014058 (1)[1], 0x40014098 (1)[2]

**Access:** Read-only if TC_CMRx.WAVE = 0, Read/Write if TC_CMRx.WAVE = 1

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RB | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| RB | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| RB | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| RB | | | | | | | |

This register can only be written if the WPEN bit is cleared in the TC Write Protection Mode Register.

- **RB: Register B**

RB contains the Register B value in real time.

IMPORTANT: For 16-bit channels, RB field size is limited to register bits 15:0.

### 32.7.8 TC Register C

**Name:** TC_RCx [x=0..2]

**Address:** 0x4001001C (0)[0], 0x4001005C (0)[1], 0x4001009C (0)[2], 0x4001401C (1)[0], 0x4001405C (1)[1], 0x4001409C (1)[2]

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RC | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| RC | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| RC | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| RC | | | | | | | |

This register can only be written if the WPEN bit is cleared in the TC Write Protection Mode Register.

- **RC: Register C**

RC contains the Register C value in real time.

IMPORTANT: For 16-bit channels, RC field size is limited to register bits 15:0.

### 32.7.9 TC Status Register

**Name:** TC_SRx [x=0..2]

**Address:** 0x40010020 (0)[0], 0x40010060 (0)[1], 0x400100A0 (0)[2], 0x40014020 (1)[0], 0x40014060 (1)[1], 0x400140A0 (1)[2]

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | MTIOB | MTIOA | CLKSTA |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ETRGS | LDRBS | LDRAS | CPCS | CPBS | CPAS | LOVRS | COVFS |

- **COVFS: Counter Overflow Status (cleared on read)**

0: No counter overflow has occurred since the last read of the Status Register.

1: A counter overflow has occurred since the last read of the Status Register.

- **LOVRS: Load Overrun Status (cleared on read)**

0: Load overrun has not occurred since the last read of the Status Register or TC_CMRx.WAVE = 1.

1: RA or RB have been loaded at least twice without any read of the corresponding register since the last read of the Status Register, if TC_CMRx.WAVE = 0.

- **CPAS: RA Compare Status (cleared on read)**

0: RA Compare has not occurred since the last read of the Status Register or TC_CMRx.WAVE = 0.

1: RA Compare has occurred since the last read of the Status Register, if TC_CMRx.WAVE = 1.

- **CPBS: RB Compare Status (cleared on read)**

0: RB Compare has not occurred since the last read of the Status Register or TC_CMRx.WAVE = 0.

1: RB Compare has occurred since the last read of the Status Register, if TC_CMRx.WAVE = 1.

- **CPCS: RC Compare Status (cleared on read)**

0: RC Compare has not occurred since the last read of the Status Register.

1: RC Compare has occurred since the last read of the Status Register.

- **LDRAS: RA Loading Status (cleared on read)**

0: RA Load has not occurred since the last read of the Status Register or TC_CMRx.WAVE = 1.

1: RA Load has occurred since the last read of the Status Register, if TC_CMRx.WAVE = 0.

- **LDRBS: RB Loading Status (cleared on read)**

0: RB Load has not occurred since the last read of the Status Register or TC_CMRx.WAVE = 1.

1: RB Load has occurred since the last read of the Status Register, if TC_CMRx.WAVE = 0.

- **ETRGS: External Trigger Status (cleared on read)**

0: External trigger has not occurred since the last read of the Status Register.

1: External trigger has occurred since the last read of the Status Register.

- **CLKSTA: Clock Enabling Status**

0: Clock is disabled.

1: Clock is enabled.

- **MTIOA: TIOA Mirror**

0: TIOA is low. If TC_CMRx.WAVE = 0, this means that TIOA pin is low. If TC_CMRx.WAVE = 1, this means that TIOA is driven low.

1: TIOA is high. If TC_CMRx.WAVE = 0, this means that TIOA pin is high. If TC_CMRx.WAVE = 1, this means that TIOA is driven high.

- **MTIOB: TIOB Mirror**

0: TIOB is low. If TC_CMRx.WAVE = 0, this means that TIOB pin is low. If TC_CMRx.WAVE = 1, this means that TIOB is driven low.

1: TIOB is high. If TC_CMRx.WAVE = 0, this means that TIOB pin is high. If TC_CMRx.WAVE = 1, this means that TIOB is driven high.

### 32.7.10 TC Interrupt Enable Register

**Name:** TC_IERx [x=0..2]

**Address:** 0x40010024 (0)[0], 0x40010064 (0)[1], 0x400100A4 (0)[2], 0x40014024 (1)[0], 0x40014064 (1)[1], 0x400140A4 (1)[2]

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ETRGS | LDRBS | LDRAS | CPCS | CPBS | CPAS | LOVRS | COVFS |

- **COVFS: Counter Overflow**

0: No effect.

1: Enables the Counter Overflow Interrupt.

- **LOVRS: Load Overrun**

0: No effect.

1: Enables the Load Overrun Interrupt.

- **CPAS: RA Compare**

0: No effect.

1: Enables the RA Compare Interrupt.

- **CPBS: RB Compare**

0: No effect.

1: Enables the RB Compare Interrupt.

- **CPCS: RC Compare**

0: No effect.

1: Enables the RC Compare Interrupt.

- **LDRAS: RA Loading**

0: No effect.

1: Enables the RA Load Interrupt.

- **LDRBS: RB Loading**

0: No effect.

1: Enables the RB Load Interrupt.

- **ETRGS: External Trigger**

0: No effect.

1: Enables the External Trigger Interrupt.

### 32.7.11 TC Interrupt Disable Register

**Name:** TC_IDRx [x=0..2]

**Address:** 0x40010028 (0)[0], 0x40010068 (0)[1], 0x400100A8 (0)[2], 0x40014028 (1)[0], 0x40014068 (1)[1], 0x400140A8 (1)[2]

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ETRGS | LDRBS | LDRAS | CPCS | CPBS | CPAS | LOVRS | COVFS |

- **COVFS: Counter Overflow**

0: No effect.

1: Disables the Counter Overflow Interrupt.

- **LOVRS: Load Overrun**

0: No effect.

1: Disables the Load Overrun Interrupt (if TC_CMRx.WAVE = 0).

- **CPAS: RA Compare**

0: No effect.

1: Disables the RA Compare Interrupt (if TC_CMRx.WAVE = 1).

- **CPBS: RB Compare**

0: No effect.

1: Disables the RB Compare Interrupt (if TC_CMRx.WAVE = 1).

- **CPCS: RC Compare**

0: No effect.

1: Disables the RC Compare Interrupt.

- **LDRAS: RA Loading**

0: No effect.

1: Disables the RA Load Interrupt (if TC_CMRx.WAVE = 0).

- **LDRBS: RB Loading**

0: No effect.

1: Disables the RB Load Interrupt (if TC_CMRx.WAVE = 0).

- **ETRGS: External Trigger**

0: No effect.

1: Disables the External Trigger Interrupt.

### 32.7.12 TC Interrupt Mask Register

**Name:** TC_IMRx [x=0..2]

**Address:** 0x4001002C (0)[0], 0x4001006C (0)[1], 0x400100AC (0)[2], 0x4001402C (1)[0], 0x4001406C (1)[1], 0x400140AC (1)[2]

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ETRGS | LDRBS | LDRAS | CPCS | CPBS | CPAS | LOVRS | COVFS |

- **COVFS: Counter Overflow**

0: The Counter Overflow Interrupt is disabled.

1: The Counter Overflow Interrupt is enabled.

- **LOVRS: Load Overrun**

0: The Load Overrun Interrupt is disabled.

1: The Load Overrun Interrupt is enabled.

- **CPAS: RA Compare**

0: The RA Compare Interrupt is disabled.

1: The RA Compare Interrupt is enabled.

- **CPBS: RB Compare**

0: The RB Compare Interrupt is disabled.

1: The RB Compare Interrupt is enabled.

- **CPCS: RC Compare**

0: The RC Compare Interrupt is disabled.

1: The RC Compare Interrupt is enabled.

- **LDRAS: RA Loading**

0: The Load RA Interrupt is disabled.

1: The Load RA Interrupt is enabled.

- **LDRBS: RB Loading**

0: The Load RB Interrupt is disabled.

1: The Load RB Interrupt is enabled.

- **ETRGS: External Trigger**

0: The External Trigger Interrupt is disabled.

1: The External Trigger Interrupt is enabled.

### 32.7.13 TC Block Control Register

**Name:** TC_BCR

**Address:** 0x400100C0 (0), 0x400140C0 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | SYNC |

- **SYNC: Synchro Command**

0: No effect.

1: Asserts the SYNC signal which generates a software trigger simultaneously for each of the channels.

### 32.7.14 TC Block Mode Register

**Name:** TC_BMR

**Address:** 0x400100C4 (0), 0x400140C4 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | MAXFILT | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| MAXFILT | | | | – | – | IDXPHB | SWAP |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| INVIDX | INVB | INVA | EDGPHA | QDTRANS | SPEEDEN | POSEN | QDEN |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | TC2XC2S | | TC1XC1S | | TC0XC0S | |

This register can only be written if the WPEN bit is cleared in the TC Write Protection Mode Register.

- **TC0XC0S: External Clock Signal 0 Selection**

| Value | Name | Description |
|---|---|---|
| 0 | TCLK0 | Signal connected to XC0: TCLK0 |
| 1 | – | Reserved |
| 2 | TIOA1 | Signal connected to XC0: TIOA1 |
| 3 | TIOA2 | Signal connected to XC0: TIOA2 |

- **TC1XC1S: External Clock Signal 1 Selection**

| Value | Name | Description |
|---|---|---|
| 0 | TCLK1 | Signal connected to XC1: TCLK1 |
| 1 | – | Reserved |
| 2 | TIOA0 | Signal connected to XC1: TIOA0 |
| 3 | TIOA2 | Signal connected to XC1: TIOA2 |

- **TC2XC2S: External Clock Signal 2 Selection**

| Value | Name | Description |
|---|---|---|
| 0 | TCLK2 | Signal connected to XC2: TCLK2 |
| 1 | – | Reserved |
| 2 | TIOA0 | Signal connected to XC2: TIOA0 |
| 3 | TIOA1 | Signal connected to XC2: TIOA1 |

- **QDEN: Quadrature Decoder Enabled**

0: Disabled.

1: Enables the QDEC (filter, edge detection and quadrature decoding).

Quadrature decoding (direction change) can be disabled using QDTRANS bit.

One of the POSEN or SPEEDEN bits must be also enabled.

- **POSEN: Position Enabled**

0: Disable position.

1: Enables the position measure on channel 0 and 1.

- **SPEEDEN: Speed Enabled**

0: Disabled.

1: Enables the speed measure on channel 0, the time base being provided by channel 2.

- **QDTRANS: Quadrature Decoding Transparent**

0: Full quadrature decoding logic is active (direction change detected).

1: Quadrature decoding logic is inactive (direction change inactive) but input filtering and edge detection are performed.

- **EDGPHA: Edge on PHA Count Mode**

0: Edges are detected on PHA only.

1: Edges are detected on both PHA and PHB.

- **INVA: Inverted PHA**

0: PHA (TIOA0) is directly driving the QDEC.

1: PHA is inverted before driving the QDEC.

- **INVB: Inverted PHB**

0: PHB (TIOB0) is directly driving the QDEC.

1: PHB is inverted before driving the QDEC.

- **INVIDX: Inverted Index**

0: IDX (TIOA1) is directly driving the QDEC.

1: IDX is inverted before driving the QDEC.

- **SWAP: Swap PHA and PHB**

0: No swap between PHA and PHB.

1: Swap PHA and PHB internally, prior to driving the QDEC.

- **IDXPHB: Index Pin is PHB Pin**

0: IDX pin of the rotary sensor must drive TIOA1.

1: IDX pin of the rotary sensor must drive TIOB0.

- **MAXFILT: Maximum Filter**

1–63: Defines the filtering capabilities.

Pulses with a period shorter than MAXFILT+1 peripheral clock cycles are discarded.

### 32.7.15 TC QDEC Interrupt Enable Register

**Name:** TC_QIER

**Address:** 0x400100C8 (0), 0x400140C8 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | QERR | DIRCHG | IDX |

- **IDX: Index**

0: No effect.

1: Enables the interrupt when a rising edge occurs on IDX input.

- **DIRCHG: Direction Change**

0: No effect.

1: Enables the interrupt when a change on rotation direction is detected.

- **QERR: Quadrature Error**

0: No effect.

1: Enables the interrupt when a quadrature error occurs on PHA, PHB.

### 32.7.16 TC QDEC Interrupt Disable Register

**Name:** TC_QIDR

**Address:** 0x400100CC (0), 0x400140CC (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | QERR | DIRCHG | IDX |

- **IDX: Index**

0: No effect.

1: Disables the interrupt when a rising edge occurs on IDX input.

- **DIRCHG: Direction Change**

0: No effect.

1: Disables the interrupt when a change on rotation direction is detected.

- **QERR: Quadrature Error**

0: No effect.

1: Disables the interrupt when a quadrature error occurs on PHA, PHB.

### 32.7.17 TC QDEC Interrupt Mask Register

**Name:** TC_QIMR

**Address:** 0x400100D0 (0), 0x400140D0 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | QERR | DIRCHG | IDX |

- **IDX: Index**

0: The interrupt on IDX input is disabled.

1: The interrupt on IDX input is enabled.

- **DIRCHG: Direction Change**

0: The interrupt on rotation direction change is disabled.

1: The interrupt on rotation direction change is enabled.

- **QERR: Quadrature Error**

0: The interrupt on quadrature error is disabled.

1: The interrupt on quadrature error is enabled.

### 32.7.18 TC QDEC Interrupt Status Register

**Name:** TC_QISR

**Address:** 0x400100D4 (0), 0x400140D4 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | DIR |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | QERR | DIRCHG | IDX |

- **IDX: Index**

0: No Index input change since the last read of TC_QISR.

1: The IDX input has changed since the last read of TC_QISR.

- **DIRCHG: Direction Change**

0: No change on rotation direction since the last read of TC_QISR.

1: The rotation direction changed since the last read of TC_QISR.

- **QERR: Quadrature Error**

0: No quadrature error since the last read of TC_QISR.

1: A quadrature error occurred since the last read of TC_QISR.

- **DIR: Direction**

Returns an image of the actual rotation direction.

### 32.7.19 TC Write Protection Mode Register

**Name:** TC_WPMR

**Address:** 0x400100E4 (0), 0x400140E4 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | WPEN |

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x54494D (“TIM” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x54494D (“TIM” in ASCII).

The Timer Counter clock of the first channel must be enabled to access this register.

See Section 32.6.16 ”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.<br>Always reads as 0. |

# 33. Pulse Width Modulation Controller (PWM)

## 33.1 Description

The PWM macrocell controls several channels independently. Each channel controls one square output waveform. Characteristics of the output waveform such as period, duty-cycle and polarity are configurable through the user interface. Each channel selects and uses one of the clocks provided by the clock generator. The clock generator provides several clocks resulting from the division of the PWM macrocell master clock.

All PWM macrocell accesses are made through APB mapped registers.

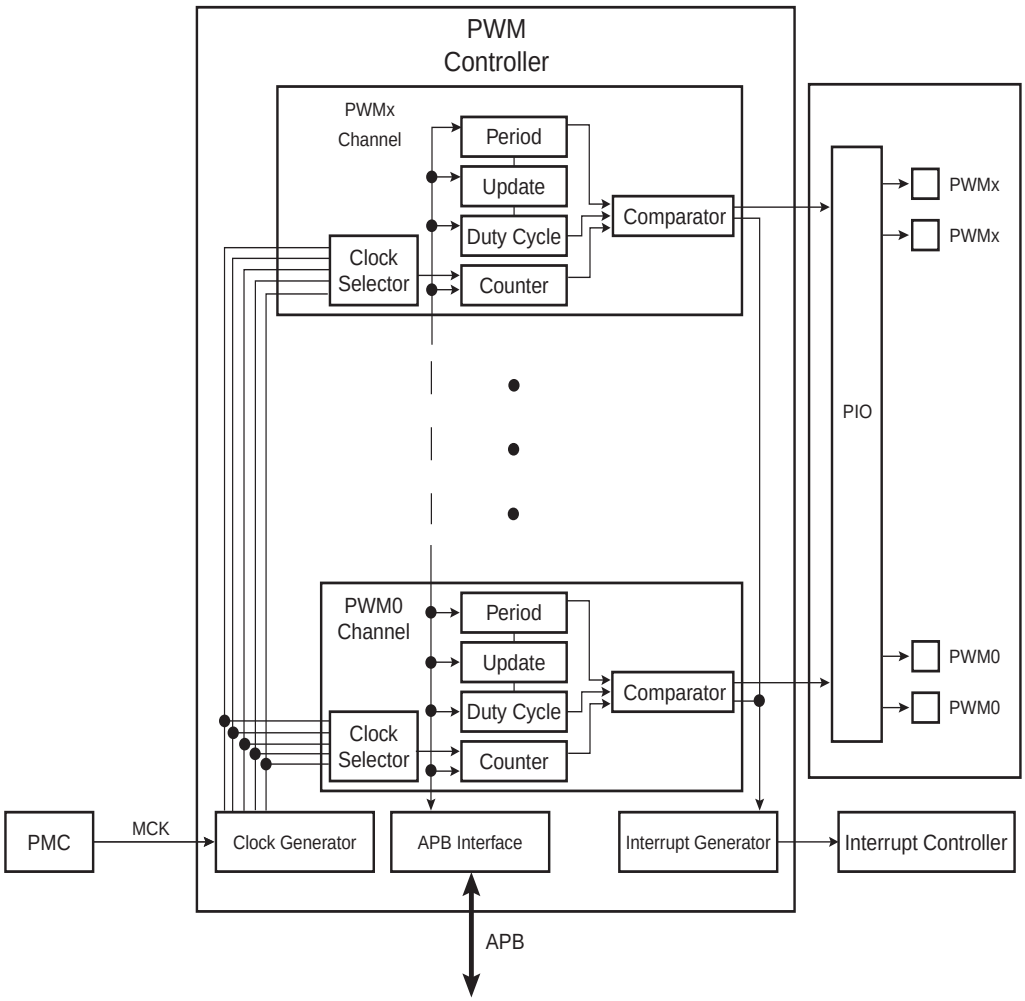
Channels can be synchronized, to generate non overlapped waveforms. All channels integrate a double buffering system in order to prevent an unexpected output waveform while modifying the period or the duty-cycle.

## 33.2 Embedded Characteristics

- 4 Channels
- One 16-bit Counter Per Channel
- Common Clock Generator Providing Thirteen Different Clocks
  - A Modulo n Counter Providing Eleven Clocks
  - Two Independent Linear Dividers Working on Modulo n Counter Outputs
- Independent Channels
  - Independent Enable Disable Command for Each Channel
  - Independent Clock Selection for Each Channel
  - Independent Period and Duty Cycle for Each Channel
  - Double Buffering of Period or Duty Cycle for Each Channel
  - Programmable Selection of The Output Waveform Polarity for Each Channel
  - Programmable Center or Left Aligned Output Waveform for Each Channel

## 33.3 Block Diagram

**Figure 33-1. Pulse Width Modulation Controller Block Diagram**




## 33.4 I/O Lines Description

Each channel outputs one waveform on one external I/O line.

**Table 33-1. I/O Line Description**

| Name | Description | Type |
|---|---|---|
| PWMx | PWM Waveform Output for channel x | Output |

## 33.5 Product Dependencies

### 33.5.1 I/O Lines

The pins used for interfacing the PWM may be multiplexed with PIO lines. The programmer must first program the PIO controller to assign the desired PWM pins to their peripheral function. If I/O lines of the PWM are not used by the application, they can be used for other purposes by the PIO controller.

All of the PWM outputs may or may not be enabled. If an application requires only four channels, then only four PIO lines will be assigned to PWM outputs.

**Table 33-2. I/O Lines**

| Instance | Signal | I/O Line | Peripheral |
|---|---|---|---|
| PWM | PWM0 | PA0 | A |
| PWM | PWM0 | PA11 | B |
| PWM | PWM0 | PA23 | B |
| PWM | PWM0 | PB0 | A |
| PWM | PWM0 | PC8 | B |
| PWM | PWM0 | PC18 | B |
| PWM | PWM0 | PC22 | B |
| PWM | PWM1 | PA1 | A |
| PWM | PWM1 | PA12 | B |
| PWM | PWM1 | PA24 | B |
| PWM | PWM1 | PB1 | A |
| PWM | PWM1 | PC9 | B |
| PWM | PWM1 | PC19 | B |
| PWM | PWM2 | PA2 | A |
| PWM | PWM2 | PA13 | B |
| PWM | PWM2 | PA25 | B |
| PWM | PWM2 | PB4 | B |
| PWM | PWM2 | PC10 | B |
| PWM | PWM2 | PC20 | B |
| PWM | PWM3 | PA7 | B |
| PWM | PWM3 | PA14 | B |
| PWM | PWM3 | PB14 | B |
| PWM | PWM3 | PC11 | B |
| PWM | PWM3 | PC21 | B |

### 33.5.2 Power Management

The PWM is not continuously clocked. The programmer must first enable the PWM clock in the Power Management Controller (PMC) before using the PWM. However, if the application does not require PWM operations, the PWM clock can be stopped when not needed and be restarted later. In this case, the PWM will resume its operations where it left off.

Configuring the PWM does not require the PWM clock to be enabled.

### 33.5.3 Interrupt Sources

The PWM interrupt line is connected on one of the internal sources of the Interrupt Controller. Using the PWM interrupt requires the Interrupt Controller to be programmed first. Note that it is not recommended to use the PWM interrupt line in edge sensitive mode.

**Table 33-3. Peripheral IDs**

| Instance | ID |
|---|---|
| PWM | 31 |

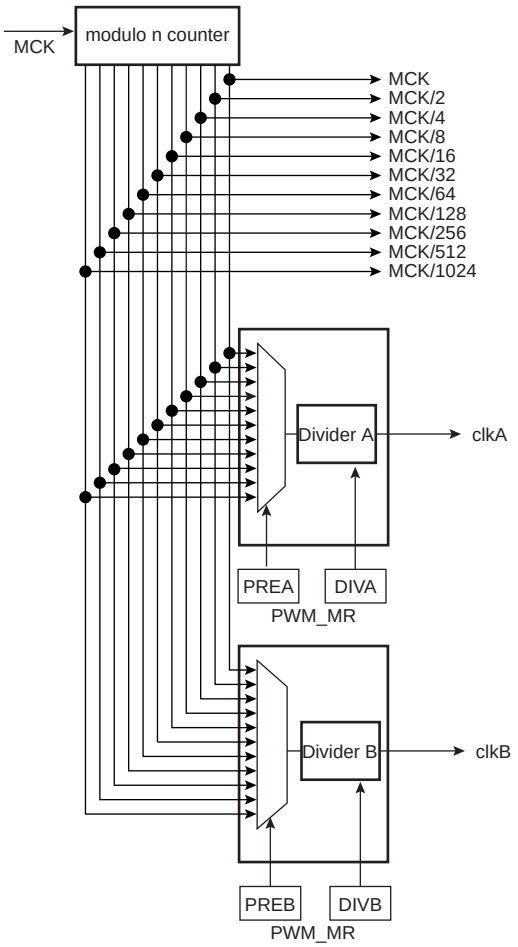
## 33.6 Functional Description

The PWM macrocell is primarily composed of a clock generator module and 4 channels.

- Clocked by the system clock, MCK, the clock generator module provides 13 clocks.
- Each channel can independently choose one of the clock generator outputs.
- Each channel generates an output waveform with attributes that can be defined independently for each channel through the user interface registers.

### 33.6.1 PWM Clock Generator

**Figure 33-2. Functional View of the Clock Generator Block Diagram**




**Caution:** Before using the PWM macrocell, the programmer must first enable the PWM clock in the Power Management Controller (PMC).

The PWM macrocell master clock, MCK, is divided in the clock generator module to provide different clocks available for all channels. Each channel can independently select one of the divided clocks.

The clock generator is divided in three blocks:

- a modulo n counter which provides 11 clocks: $F_{MCK}$, $F_{MCK}/2$, $F_{MCK}/4$, $F_{MCK}/8$, $F_{MCK}/16$, $F_{MCK}/32$, $F_{MCK}/64$, $F_{MCK}/128$, $F_{MCK}/256$, $F_{MCK}/512$, $F_{MCK}/1024$
- two linear dividers (1, 1/2, 1/3, ... 1/255) that provide two separate clocks: clkA and clkB

Each linear divider can independently divide one of the clocks of the modulo n counter. The selection of the clock to be divided is made according to the PREA (PREB) field of the PWM Mode register (PWM_MR). The resulting clock clkA (clkB) is the clock selected divided by DIVA (DIVB) field value in the PWM Mode register (PWM_MR).

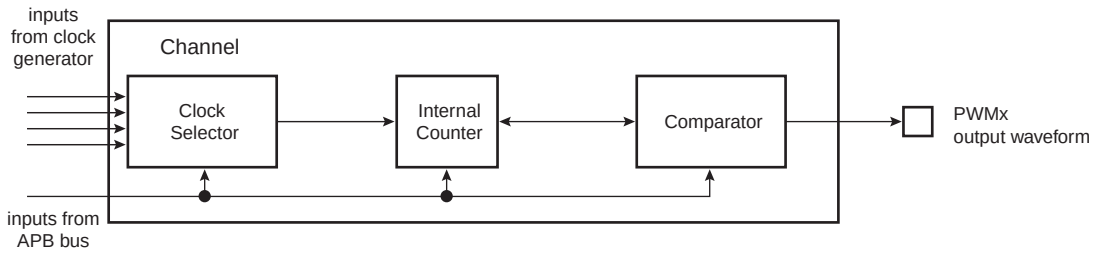
After a reset of the PWM controller, DIVA (DIVB) and PREA (PREB) in the PWM Mode register are set to 0. This implies that after reset clkA (clkB) are turned off.

At reset, all clocks provided by the modulo n counter are turned off except clock “clk”. This situation is also true when the PWM master clock is turned off through the Power Management Controller.

## 33.6.2 PWM Channel

### 33.6.2.1 Block Diagram

**Figure 33-3. Functional View of the Channel Block Diagram**




Each of the 4 channels is composed of three blocks:

- A clock selector which selects one of the clocks provided by the clock generator described in Section 33.6.1 “PWM Clock Generator” on page 641.
- An internal counter clocked by the output of the clock selector. This internal counter is incremented or decremented according to the channel configuration and comparators events. The size of the internal counter is 16 bits.
- A comparator used to generate events according to the internal counter value. It also computes the PWMx output waveform according to the configuration.

#### 33.6.2.2 Waveform Properties

The different properties of output waveforms are:

- the ***internal clock selection***. The internal channel counter is clocked by one of the clocks provided by the clock generator described in the previous section. This channel parameter is defined in the CPRE field of the PWM_CMRx register. This field is reset at 0.
- the ***waveform period***. This channel parameter is defined in the CPRD field of the PWM_CPRDx register.
  - If the waveform is left aligned, then the output waveform period depends on the counter source clock and can be calculated:
  By using the Master Clock (MCK) divided by an X given prescaler value
  (with X being 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024), the resulting period formula will be:

  $$\frac{(X \times CPRD)}{MCK}$$

  By using a Master Clock divided by one of both DIVA or DIVB divider, the formula becomes, respectively:

  $$\frac{(X*CPRD*DIVA)}{MCK} \text{ or } \frac{(X*CPRD*DIVB)}{MCK}$$

  If the waveform is center aligned then the output waveform period depends on the counter source clock and can be calculated:
  By using the Master Clock (MCK) divided by an X given prescaler value
  (with X being 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024). The resulting period formula will be:

  $$\frac{(2 \times X \times CPRD)}{MCK}$$

  By using a Master Clock divided by one of both DIVA or DIVB divider, the formula becomes, respectively:

  $$\frac{(2*X*CPRD*DIVA)}{MCK} \text{ or } \frac{(2*X*CPRD*DIVB)}{MCK}$$

- the ***waveform duty cycle***. This channel parameter is defined in the CDTY field of the PWM_CDTYx register.
  If the waveform is left aligned then:

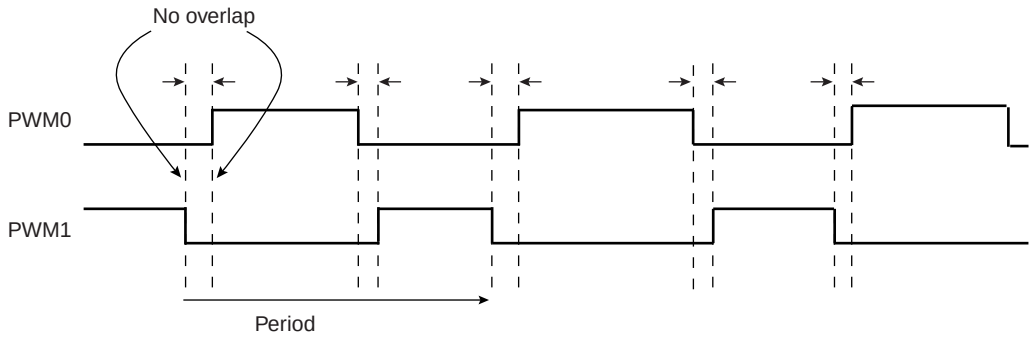
  $$\text{duty cycle} = \frac{(period - 1/\text{fchannel\_x\_clock} \times CDTY)}{period}$$

  If the waveform is center aligned, then:

  $$\text{duty cycle} = \frac{((period/2) - 1/\text{fchannel\_x\_clock} \times CDTY))}{(period/2)}$$

- the ***waveform polarity.*** At the beginning of the period, the signal can be at high or low level. This property is defined in the CPOL field of the PWM_CMRx register. By default the signal starts by a low level.
- the ***waveform alignment***. The output waveform can be left or center aligned. Center aligned waveforms can be used to generate non overlapped waveforms. This property is defined in the CALG field of the PWM_CMRx register. The default mode is left aligned.

**Figure 33-4. Non Overlapped Center Aligned Waveforms**




Note: 1. See Figure 33-5 on page 645 for a detailed description of center aligned waveforms.

When center aligned, the internal channel counter increases up to CPRD and.decreases down to 0. This ends the period.

When left aligned, the internal channel counter increases up to CPRD and is reset. This ends the period.

Thus, for the same CPRD value, the period for a center aligned channel is twice the period for a left aligned channel.

Waveforms are fixed at 0 when:

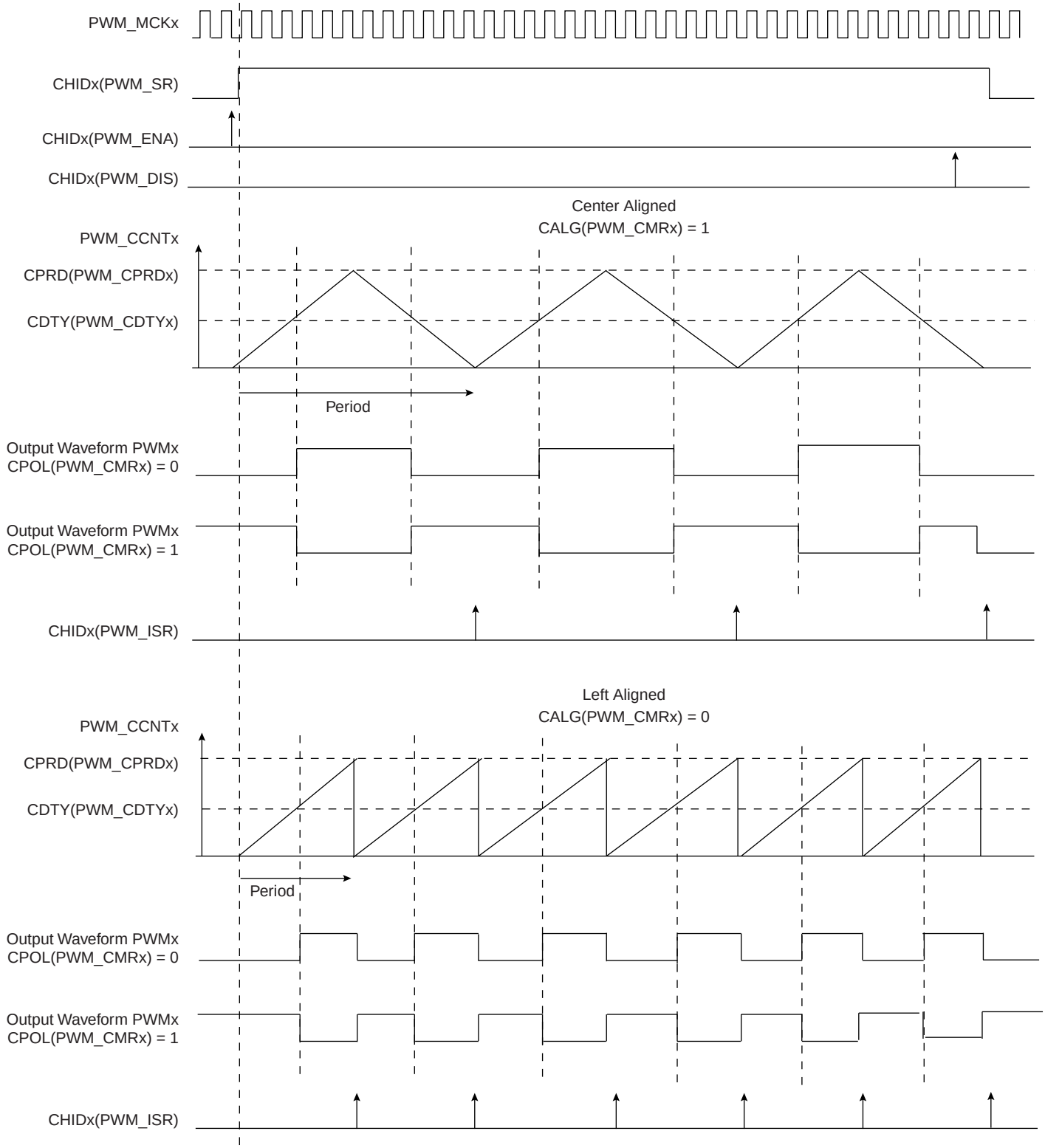
- CDTY = CPRD and CPOL = 0
- CDTY = 0 and CPOL = 1

Waveforms are fixed at 1 (once the channel is enabled) when:

- CDTY = 0 and CPOL = 0
- CDTY = CPRD and CPOL = 1

The waveform polarity must be set before enabling the channel. This immediately affects the channel output level. Changes on channel polarity are not taken into account while the channel is enabled.

**Figure 33-5. Waveform Properties**



### 33.6.3 PWM Controller Operations

#### 33.6.3.1 Initialization

Before enabling the output channel, this channel must have been configured by the software application:

- Configuration of the clock generator if DIVA and DIVB are required
- Selection of the clock for each channel (CPRE field in the PWM_CMRx register)
- Configuration of the waveform alignment for each channel (CALG field in the PWM_CMRx register)
- Configuration of the period for each channel (CPRD in the PWM_CPRDx register). Writing in PWM_CPRDx Register is possible while the channel is disabled. After validation of the channel, the user must use PWM_CUPDx Register to update PWM_CPRDx as explained below.
- Configuration of the duty cycle for each channel (CDTY in the PWM_CDTYx register). Writing in PWM_CDTYx Register is possible while the channel is disabled. After validation of the channel, the user must use PWM_CUPDx Register to update PWM_CDTYx as explained below.
- Configuration of the output waveform polarity for each channel (CPOL in the PWM_CMRx register)
- Enable Interrupts (Writing CHIDx in the PWM_IER register)
- Enable the PWM channel (Writing CHIDx in the PWM_ENA register)

It is possible to synchronize different channels by enabling them at the same time by means of writing simultaneously several CHIDx bits in the PWM_ENA register.

- In such a situation, all channels may have the same clock selector configuration and the same period specified.

#### 33.6.3.2 Source Clock Selection Criteria

The large number of source clocks can make selection difficult. The relationship between the value in the Period Register (PWM_CPRDx) and the Duty Cycle Register (PWM_CDTYx) can help the user in choosing. The event number written in the Period Register gives the PWM accuracy. The Duty Cycle quantum cannot be lower than *1/PWM_CPRDx* value. The higher the value of PWM_CPRDx, the greater the PWM accuracy.

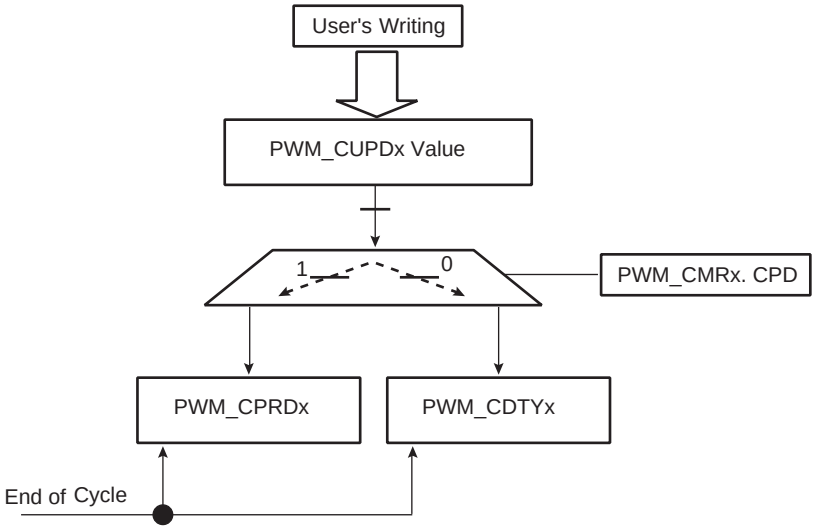
For example, if the user sets 15 (in decimal) in PWM_CPRDx, the user is able to set a value between 1 up to 14 in PWM_CDTYx Register. The resulting duty cycle quantum cannot be lower than 1/15 of the PWM period.

#### 33.6.3.3 Changing the Duty Cycle or the Period

It is possible to modulate the output waveform duty cycle or period.

To prevent unexpected output waveform, the user must use the update register (PWM_CUPDx) to change waveform parameters while the channel is still enabled. The user can write a new period value or duty cycle value in the update register (PWM_CUPDx). This register holds the new value until the end of the current cycle and updates the value for the next cycle. Depending on the CPD field in the PWM_CMRx register, PWM_CUPDx either updates PWM_CPRDx or PWM_CDTYx. Note that even if the update register is used, the period must not be smaller than the duty cycle.

**Figure 33-6. Synchronized Period or Duty Cycle Update**



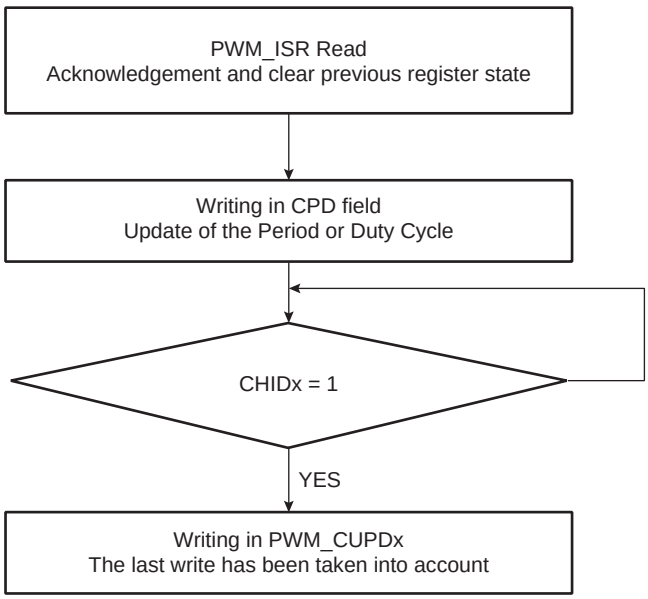

To prevent overwriting the PWM_CUPDx by software, the user can use status events in order to synchronize his software. Two methods are possible. In both, the user must enable the dedicated interrupt in PWM_IER at PWM Controller level.

The first method (polling method) consists of reading the relevant status bit in PWM_ISR Register according to the enabled channel(s). See Figure 33-7.

The second method uses an Interrupt Service Routine associated with the PWM channel.

Note: Reading the PWM_ISR register automatically clears CHIDx flags.

**Figure 33-7. Polling Method**




Note: Polarity and alignment can be modified only when the channel is disabled.

#### 33.6.3.4 Interrupts

Depending on the interrupt mask in the PWM_IMR register, an interrupt is generated at the end of the corresponding channel period. The interrupt remains active until a read operation in the PWM_ISR register occurs.

A channel interrupt is enabled by setting the corresponding bit in the PWM_IER register. A channel interrupt is disabled by setting the corresponding bit in the PWM_IDR register.

## 33.7 Pulse Width Modulation Controller (PWM) User Interface

**Table 33-4. Register Mapping**[2]

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x00 | PWM Mode Register | PWM_MR | Read-write | 0 |
| 0x04 | PWM Enable Register | PWM_ENA | Write-only | - |
| 0x08 | PWM Disable Register | PWM_DIS | Write-only | - |
| 0x0C | PWM Status Register | PWM_SR | Read-only | 0 |
| 0x10 | PWM Interrupt Enable Register | PWM_IER | Write-only | - |
| 0x14 | PWM Interrupt Disable Register | PWM_IDR | Write-only | - |
| 0x18 | PWM Interrupt Mask Register | PWM_IMR | Read-only | 0 |
| 0x1C | PWM Interrupt Status Register | PWM_ISR | Read-only | 0 |
| 0x20 - 0xFC | Reserved | – | – | – |
| 0x100 - 0x1FC | Reserved | | | |
| 0x200 + ch_num * 0x20 + 0x00 | PWM Channel Mode Register | PWM_CMR | Read-write | 0x0 |
| 0x200 + ch_num * 0x20 + 0x04 | PWM Channel Duty Cycle Register | PWM_CDTY | Read-write | 0x0 |
| 0x200 + ch_num * 0x20 + 0x08 | PWM Channel Period Register | PWM_CPRD | Read-write | 0x0 |
| 0x200 + ch_num * 0x20 + 0x0C | PWM Channel Counter Register | PWM_CCNT | Read-only | 0x0 |
| 0x200 + ch_num * 0x20 + 0x10 | PWM Channel Update Register | PWM_CUPD | Write-only | - |

2. Some registers are indexed with “ch_num” index ranging from 0 to 3.

### 33.7.1 PWM Mode Register

**Name:** PWM_MR

**Address:** 0x40020000

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | PREB | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| DIVB | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | PREA | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| DIVA | | | | | | | |

- **DIVA, DIVB: CLKA, CLKB Divide Factor**

| Value | Name | Description |
|---|---|---|
| 0 | CLK_OFF | CLKA, CLKB clock is turned off |
| 1 | CLK_DIV1 | CLKA, CLKB clock is clock selected by PREA, PREB |
| 2-255 | – | CLKA, CLKB clock is clock selected by PREA, PREB divided by DIVA, DIVB factor. |

- **PREA, PREB**

| Value | Name | Description |
|---|---|---|
| 0000 | MCK | Master Clock |
| 0001 | MCKDIV2 | Master Clock divided by 2 |
| 0010 | MCKDIV4 | Master Clock divided by 4 |
| 0011 | MCKDIV8 | Master Clock divided by 8 |
| 0100 | MCKDIV16 | Master Clock divided by 16 |
| 0101 | MCKDIV32 | Master Clock divided by 32 |
| 0110 | MCKDIV64 | Master Clock divided by 64 |
| 0111 | MCKDIV128 | Master Clock divided by 128 |
| 1000 | MCKDIV256 | Master Clock divided by 256 |
| 1001 | MCKDIV512 | Master Clock divided by 512 |
| 1010 | MCKDIV1024 | Master Clock divided by 1024 |

Values which are not listed in the table must be considered as “reserved”.

### 33.7.2 PWM Enable Register

**Name:** PWM_ENA

**Address:** 0x40020004

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | | CHID3 | CHID2 | CHID1 | CHID0 |

- **CHIDx: Channel ID**

0 = No effect.

1 = Enable PWM output for channel x.

### 33.7.3 PWM Disable Register

**Name:** PWM_DIS

**Address:** 0x40020008

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | | CHID3 | CHID2 | CHID1 | CHID0 |

- **CHIDx: Channel ID**

0 = No effect.

1 = Disable PWM output for channel x.

### 33.7.4 PWM Status Register

**Name:** PWM_SR

**Address:** 0x4002000C

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | | CHID3 | CHID2 | CHID1 | CHID0 |

- **CHIDx: Channel ID**

0 = PWM output for channel x is disabled.

1 = PWM output for channel x is enabled.

### 33.7.5 PWM Interrupt Enable Register

**Name:** PWM_IER

**Address:** 0x40020010

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | | CHID3 | CHID2 | CHID1 | CHID0 |

- **CHIDx: Channel ID.**

0 = No effect.

1 = Enable interrupt for PWM channel x.

### 33.7.6 PWM Interrupt Disable Register

**Name:** PWM_IDR

**Address:** 0x40020014

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | | CHID3 | CHID2 | CHID1 | CHID0 |

- **CHIDx: Channel ID.**

0 = No effect.

1 = Disable interrupt for PWM channel x.

### 33.7.7 PWM Interrupt Mask Register

**Name:** PWM_IMR

**Address:** 0x40020018

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | | CHID3 | CHID2 | CHID1 | CHID0 |

- **CHIDx: Channel ID.**

0 = Interrupt for PWM channel x is disabled.

1 = Interrupt for PWM channel x is enabled.

### 33.7.8 PWM Interrupt Status Register

**Name:** PWM_ISR

**Address:** 0x4002001C

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | | CHID3 | CHID2 | CHID1 | CHID0 |

- **CHIDx: Channel ID**

0 = No new channel period has been achieved since the last read of the PWM_ISR register.

1 = At least one new channel period has been achieved since the last read of the PWM_ISR register.

Note: Reading PWM_ISR automatically clears CHIDx flags.

### 33.7.9 PWM Channel Mode Register

**Name:** PWM_CMR[0..3]

**Addresses:** 0x40020200 [0], 0x40020220 [1], 0x40020240 [2], 0x40020260 [3]

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | CPD | CPOL | CALG |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | CPRE | | | |

- **CPRE: Channel Pre-scaler**

| Value | Name | Description |
|---|---|---|
| 0000 | MCK | Master Clock |
| 0001 | MCKDIV2 | Master Clock divided by 2 |
| 0010 | MCKDIV4 | Master Clock divided by 4 |
| 0011 | MCKDIV8 | Master Clock divided by 8 |
| 0100 | MCKDIV16 | Master Clock divided by 16 |
| 0101 | MCKDIV32 | Master Clock divided by 32 |
| 0110 | MCKDIV64 | Master Clock divided by 64 |
| 0111 | MCKDIV128 | Master Clock divided by 128 |
| 1000 | MCKDIV256 | Master Clock divided by 256 |
| 1001 | MCKDIV512 | Master Clock divided by 512 |
| 1010 | MCKDIV1024 | Master Clock divided by 1024 |
| 1011 | CLKA | Clock A |
| 1100 | CLKB | Clock B |

Values which are not listed in the table must be considered as “reserved”.

- **CALG: Channel Alignment**

0 = The period is left aligned.

1 = The period is center aligned.

- **CPOL: Channel Polarity**

0 = The output waveform starts at a low level.

1 = The output waveform starts at a high level.

- **CPD: Channel Update Period**

0 = Writing to the PWM_CUPDx will modify the duty cycle at the next period start event.

1 = Writing to the PWM_CUPDx will modify the period at the next period start event.

### 33.7.10 PWM Channel Duty Cycle Register

**Name:** PWM_CDTY[0..3]

**Addresses:** 0x40020204 [0], 0x40020224 [1], 0x40020244 [2], 0x40020264 [3]

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| CDTY | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| CDTY | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CDTY | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CDTY | | | | | | | |

Only the first 16 bits (internal channel counter size) are significant.

- **CDTY: Channel Duty Cycle**

Defines the waveform duty cycle. This value must be defined between 0 and CPRD (PWM_CPRx).

### 33.7.11 PWM Channel Period Register

**Name:** PWM_CPRD[0..3]

**Addresses:** 0x40020208 [0], 0x40020228 [1], 0x40020248 [2], 0x40020268 [3]

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| CPRD | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| CPRD | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CPRD | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CPRD | | | | | | | |

Only the first 16 bits (internal channel counter size) are significant.

- **CPRD: Channel Period**

If the waveform is left-aligned, then the output waveform period depends on the counter source clock and can be calculated:

– By using the Master Clock (MCK) divided by an X given prescaler value (with X being 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024). The resulting period formula will be:

$$\frac{(X \times CPRD)}{MCK}$$

– By using a Master Clock divided by one of both DIVA or DIVB divider, the formula becomes, respectively:

$$\frac{(CRPD \times DIVA)}{MCK} \text{ or } \frac{(CRPD \times DIVAB)}{MCK}$$

If the waveform is center-aligned, then the output waveform period depends on the counter source clock and can be calculated:

– By using the Master Clock (MCK) divided by an X given prescaler value (with X being 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024). The resulting period formula will be:

$$\frac{(2 \times X \times CPRD)}{MCK}$$

– By using a Master Clock divided by one of both DIVA or DIVB divider, the formula becomes, respectively:

$$\frac{(2 \times CPRD \times DIVA)}{MCK} \text{ or } \frac{(2 \times CPRD \times DIVB)}{MCK}$$

### 33.7.12 PWM Channel Counter Register

**Name:** PWM_CCNT[0..3]

**Addresses:** 0x4002020C [0], 0x4002022C [1], 0x4002024C [2], 0x4002026C [3]

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| CNT | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| CNT | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CNT | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CNT | | | | | | | |

- **CNT: Channel Counter Register**

Internal counter value. This register is reset when:

- the channel is enabled (writing CHIDx in the PWM_ENA register).
- the counter reaches CPRD value defined in the PWM_CPRDx register if the waveform is left aligned.

### 33.7.13 PWM Channel Update Register

**Name:** PWM_CUPD[0..3]

**Addresses:** 0x40020210 [0], 0x40020230 [1], 0x40020250 [2], 0x40020270 [3]

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| CUPD | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| CUPD | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CUPD | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CUPD | | | | | | | |

CUPD: Channel Update Register

This register acts as a double buffer for the period or the duty cycle. This prevents an unexpected waveform when modifying the waveform period or duty-cycle.

Only the first 16 bits (internal channel counter size) are significant.

When CPD field of PWM_CMRx register = 0, the duty-cycle (CDTY of PWM_CDTYx register) is updated with the CUPD value at the beginning of the next period.

When CPD field of PWM_CMRx register = 1, the period (CPRD of PWM_CPRDx register) is updated with the CUPD value at the beginning of the next period.

# 34. Analog-to-digital Converter (ADC)

## 34.1 Description

The ADC is based on a 10-bit Analog-to-Digital Converter (ADC) managed by an ADC Controller. Refer to the Block Diagram: Figure 34-1. It also integrates a 16-to-1 analog multiplexer, making possible the analog-to-digital conversions of 16 analog lines. The conversions extend from 0V to ADVREF. The ADC supports an 8-bit or 10-bit resolution mode, and conversion results are reported in a common register for all channels, as well as in a channel-dedicated register. Software trigger, external trigger on rising edge of the ADTRG pin or internal triggers from Timer Counter output(s) are configurable.

The comparison circuitry allows automatic detection of values below a threshold, higher than a threshold, in a given range or outside the range, thresholds and ranges being fully configurable.

The ADC also integrates a Sleep Mode and a conversion sequencer and connects with a PDC channel. These features reduce both power consumption and processor intervention.

A whole set of reference voltages is generated internally from a single external reference voltage node that may be equal to the analog supply voltage. An external decoupling capacitance is required for noise filtering.

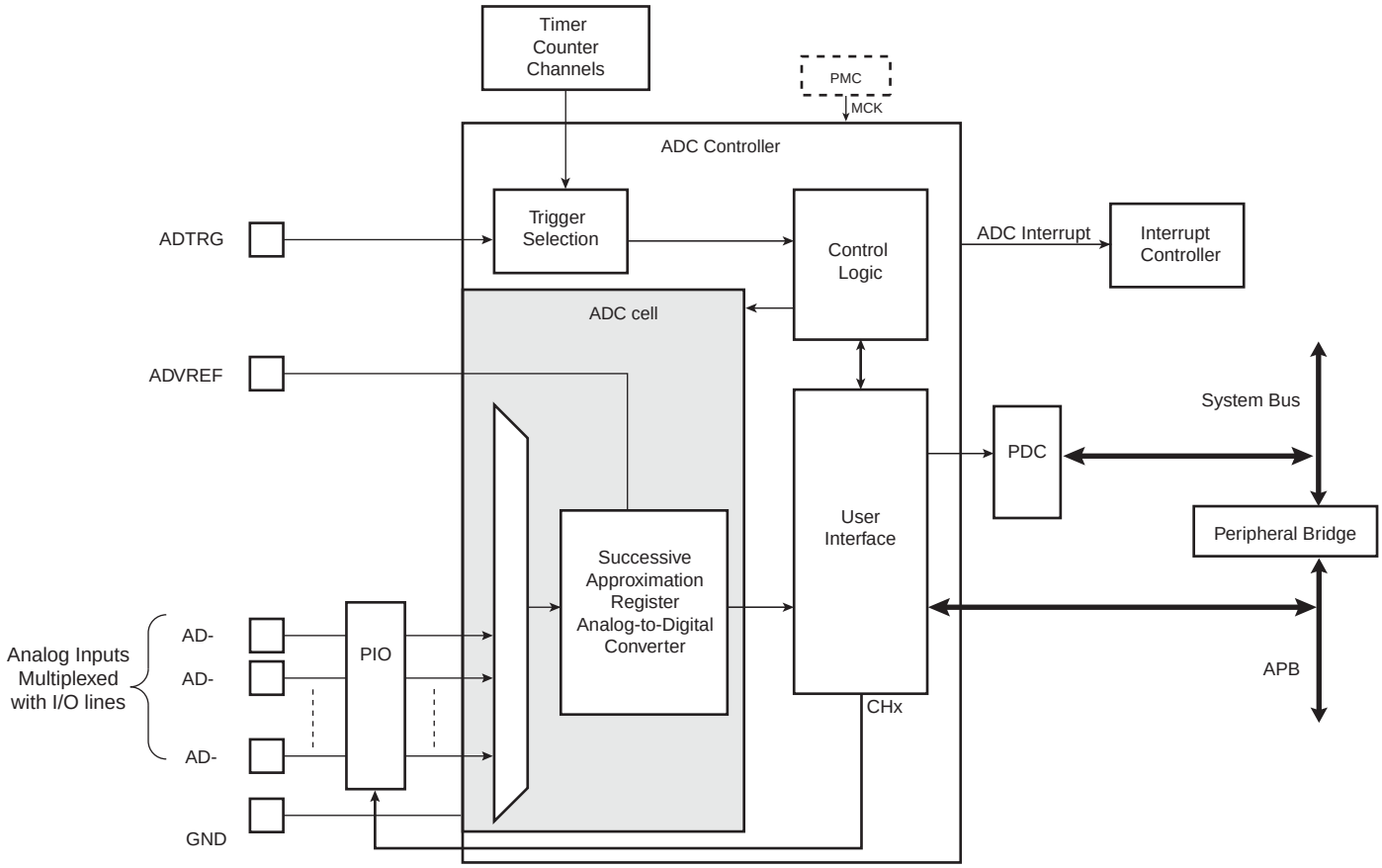
Finally, the user can configure ADC timings, such as Startup Time and Tracking Time.

## 34.2 Embedded Characteristics

- 10-bit Resolution
- 500 kHz Conversion Rate
- Wide Range Power Supply Operation
- Integrated Multiplexer Offering Up to 16 Independent Analog Inputs
- Individual Enable and Disable of Each Channel
- Hardware or Software Trigger
  - External Trigger Pin
  - Timer Counter Outputs (Corresponding TIOA Trigger)
- PDC Support
- Possibility of ADC Timings Configuration
- Two Sleep Modes and Conversion Sequencer
  - Automatic Wakeup on Trigger and Back to Sleep Mode after Conversions of all Enabled Channels
  - Possibility of Customized Channel Sequence
- Standby Mode for Fast Wakeup Time Response
  - Power Down Capability
- Automatic Window Comparison of Converted Values
- Write Protect Registers

## 34.3 Block Diagram

**Figure 34-1. Analog-to-Digital Converter Block Diagram**




## 34.4 Signal Description

**Table 34-1. ADC Pin Description**

| Pin Name | Description |
|---|---|
| ADVREF | Reference voltage |
| AD0 - AD15 | Analog input channels |
| ADTRG | External trigger |

## 34.5 Product Dependencies

### 34.5.1 Power Management

The ADC Controller is not continuously clocked. The programmer must first enable the ADC Controller MCK in the Power Management Controller (PMC) before using the ADC Controller. However, if the application does not require ADC operations, the ADC Controller clock can be stopped when not needed and restarted when necessary. Configuring the ADC Controller does not require the ADC Controller clock to be enabled.

### 34.5.2 Interrupt Sources

The ADC interrupt line is connected on one of the internal sources of the Interrupt Controller. Using the ADC interrupt requires the NVIC to be programmed first.

**Table 34-2. Peripheral IDs**

| Instance | ID |
|---|---|
| ADC | 29 |

### 34.5.3 Analog Inputs

The analog input pins can be multiplexed with PIO lines. In this case, the assignment of the ADC input is automatically done as soon as the corresponding channel is enabled by writing the register ADC_CHER. By default, after reset, the PIO line is configured as input with its pull-up enabled and the ADC input is connected to the GND.

### 34.5.4 I/O Lines

The pin ADTRG may be shared with other peripheral functions through the PIO Controller. In this case, the PIO Controller should be set accordingly to assign the pin ADTRG to the ADC function.

**Table 34-3. I/O Lines**

| Instance | Signal | I/O Line | Peripheral |
|---|---|---|---|
| ADC | ADTRG | PA8 | B |
| ADC | AD0 | PA17 | X1 |
| ADC | AD1 | PA18 | X1 |
| ADC | AD2/WKUP9 | PA19 | X1 |
| ADC | AD3/WKUP10 | PA20 | X1 |
| ADC | AD4 | PB0 | X1 |
| ADC | AD5 | PB1 | X1 |
| ADC | AD6/WKUP12 | PB2 | X1 |
| ADC | AD7 | PB3 | X1 |
| ADC | AD8 | PA21 | X1 |
| ADC | AD9 | PA22 | X1 |
| ADC | AD10 | PC13 | X1 |
| ADC | AD11 | PC15 | X1 |
| ADC | AD12 | PC12 | X1 |
| ADC | AD13 | PC29 | X1 |
| ADC | AD14 | PC30 | X1 |
| ADC | AD15 | PC31 | X1 |

### 34.5.5 Timer Triggers

Timer Counters may or may not be used as hardware triggers depending on user requirements. Thus, some or all of the timer counters may be unconnected.

### 34.5.6 Conversion Performances

For performance and electrical characteristics of the ADC, see the product DC Characteristics section.

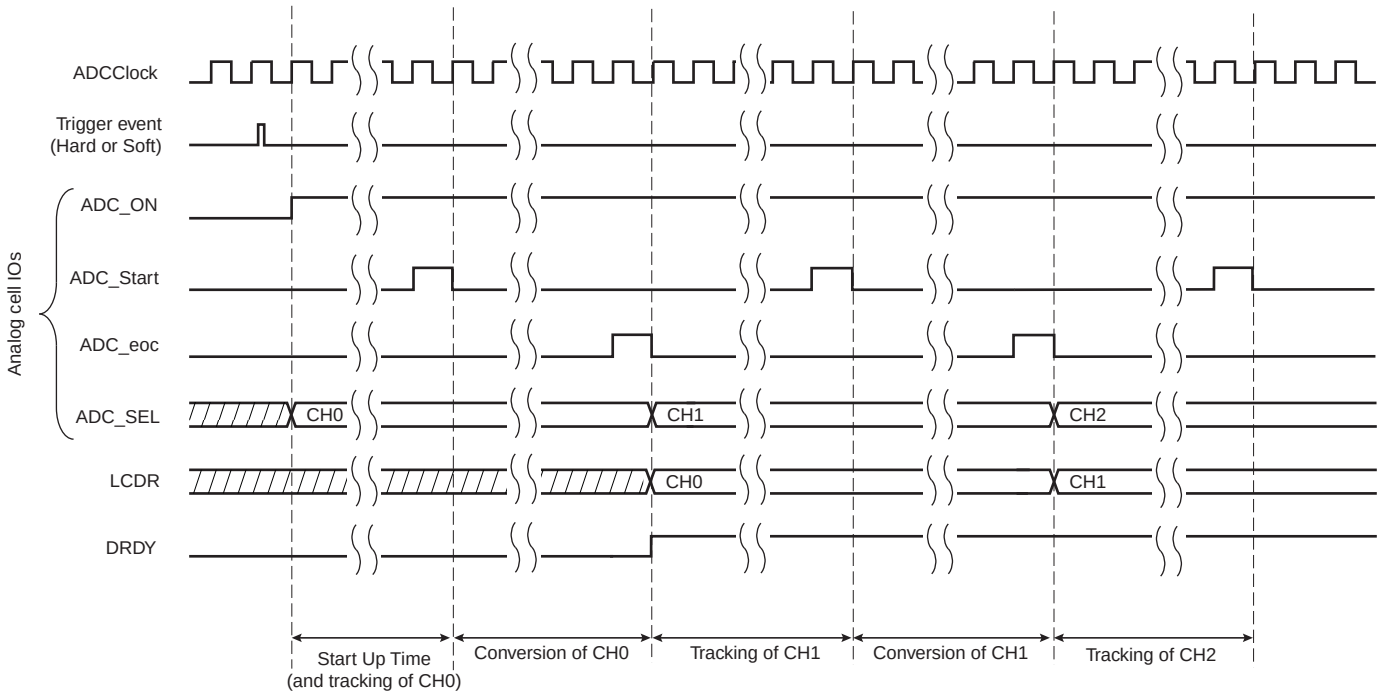
## 34.6 Functional Description

### 34.6.1 Analog-to-digital Conversion

The ADC uses the ADC Clock to perform conversions. Converting a single analog value to a 10-bit digital data requires Tracking Clock cycles as defined in the field TRACKTIM of the “ADC Mode Register” on page 675 and Transfer Clock cycles as defined in the field TRANSFER of the same register. The ADC Clock frequency is selected in the PRESCAL field of the Mode Register (ADC_MR). The tracking phase starts during the conversion of the previous channel. If the tracking time is longer than the conversion time, the tracking phase is extended to the end of the previous conversion.

The ADC clock range is between MCK/2, if PRESCAL is 0, and MCK/512, if PRESCAL is set to 255 (0xFF). PRESCAL must be programmed in order to provide an ADC clock frequency according to the parameters given in the product Electrical Characteristics section.

**Figure 34-2. Sequence of ADC conversions**




### 34.6.2 Conversion Reference

The conversion is performed on a full range between 0V and the reference voltage pin ADVREF. Analog inputs between these voltages convert to values based on a linear conversion.

### 34.6.3 Conversion Resolution

The ADC supports 8-bit or 10-bit resolutions. The 8-bit selection is performed by setting the LOWRES bit in the ADC Mode Register (ADC_MR). By default, after a reset, the resolution is the highest and the DATA field in the data registers is fully used. By setting the LOWRES bit, the ADC switches to the lowest resolution and the conversion results can be read in the lowest significant bits of the data registers. The two highest bits of the DATA field in the corresponding ADC_CDR register and of the LDATA field in the ADC_LCDR register read 0.

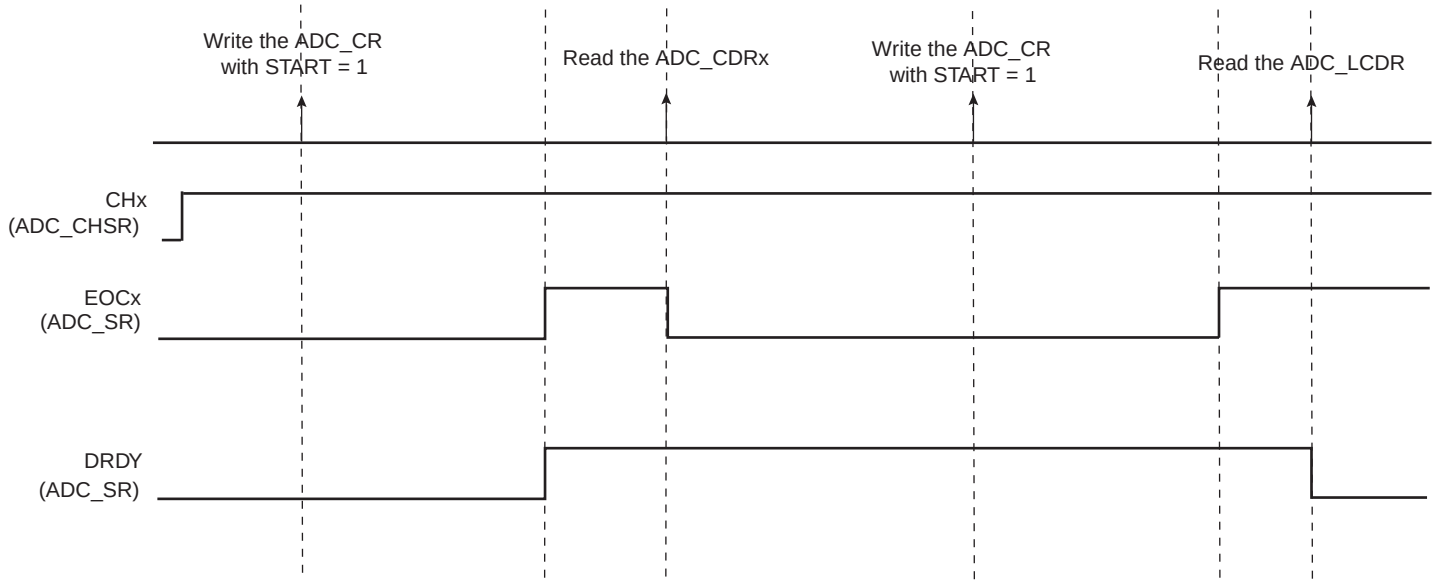
### 34.6.4 Conversion Results

When a conversion is completed, the resulting 10-bit digital value is stored in the Channel Data Register (ADC_CDRx) of the current channel and in the ADC Last Converted Data Register (ADC_LCDR). By setting the TAG option in the ADC_EMR, the ADC_LCDR presents the channel number associated to the last converted data in the CHNB field.

The channel EOC bit in the Status Register (ADC_SR) is set and the DRDY is set. In the case of a connected PDC channel, DRDY rising triggers a data transfer request. In any case, either EOC and DRDY can trigger an interrupt.

Reading one of the ADC_CDR registers clears the corresponding EOC bit. Reading ADC_LCDR clears the DRDY bit and EOC bit corresponding to the last converted channel.

**Figure 34-3. EOCx and DRDY Flag Behavior**

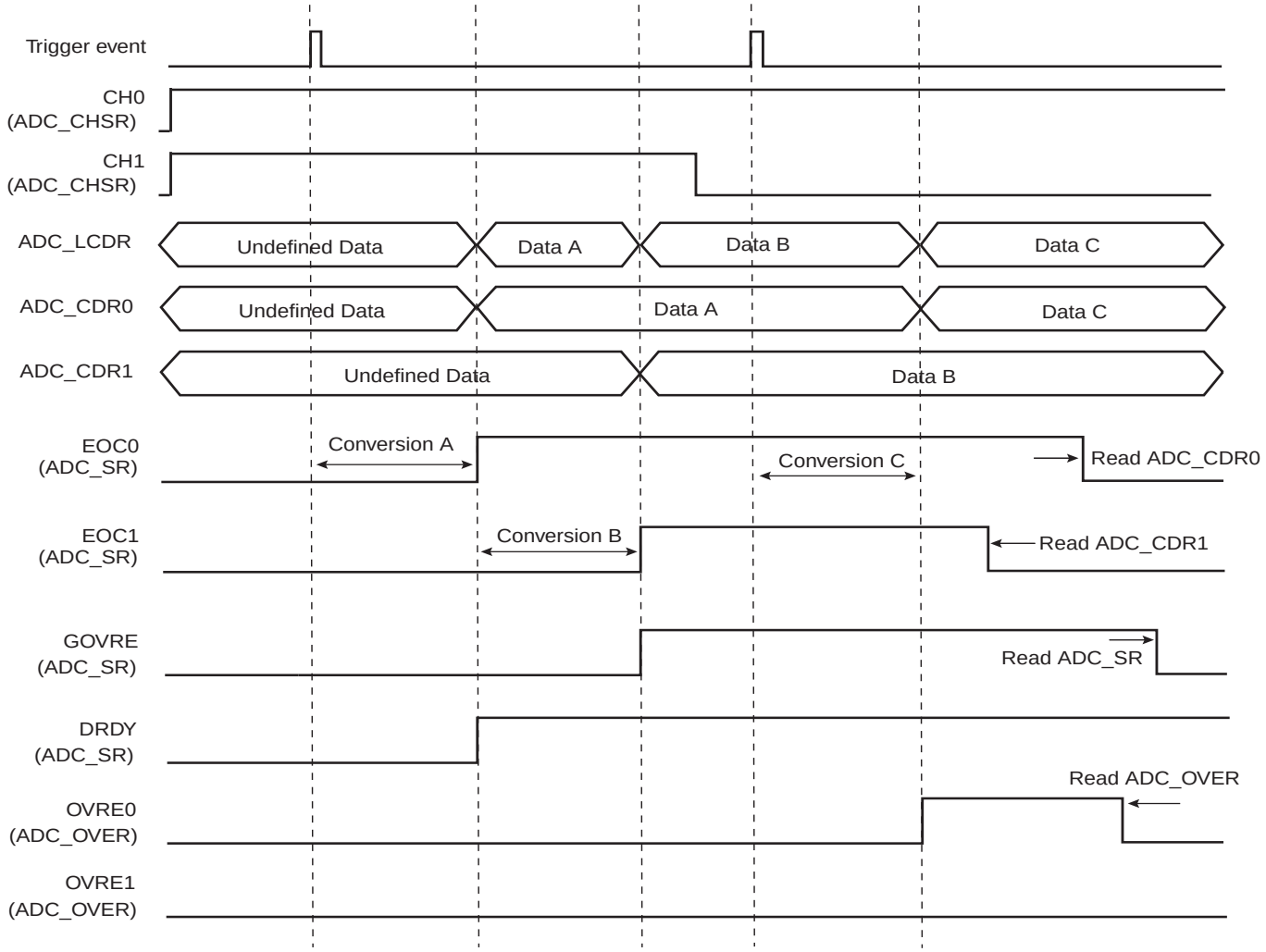



If the ADC_CDR is not read before further incoming data is converted, the corresponding Overrun Error (OVREx) flag is set in the Overrun Status Register (ADC_OVER).

Likewise, new data converted when DRDY is high sets the GOVRE bit (General Overrun Error) in ADC_SR.

The OVREx flag is automatically cleared when ADC_OVER is read, and GOVRE flag is automatically cleared when ADC_SR is read.

**Figure 34-4. GOVRE and OVREx Flag Behavior**




**Warning:** If the corresponding channel is disabled during a conversion or if it is disabled and then reenabled during a conversion, its associated data and its corresponding EOC and OVRE flags in ADC_SR are unpredictable.

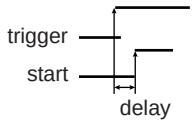
### 34.6.5 Conversion Triggers

Conversions of the active analog channels are started with a software or hardware trigger. The software trigger is provided by writing the Control Register (ADC_CR) with the START bit at 1.

The hardware trigger can be one of the TIOA outputs of the Timer Counter channels or the external trigger input of the ADC (ADTRG). The hardware trigger is selected with the TRGSEL field in the Mode Register (ADC_MR). The selected hardware trigger is enabled with the TRGEN bit in the Mode Register (ADC_MR).

The minimum time between 2 consecutive trigger events must be strictly greater than the duration time of the longest conversion sequence according to configuration of registers ADC_MR, ADC_CHSR, ADC_SEQR1, ADC_SEQR2.

If a hardware trigger is selected, the start of a conversion is triggered after a delay starting at each rising edge of the selected signal. Due to asynchronous handling, the delay may vary in a range of 2 MCK clock periods to 1 ADC clock period.




If one of the TIOA outputs is selected, the corresponding Timer Counter channel must be programmed in Waveform Mode.

Only one start command is necessary to initiate a conversion sequence on all the channels. The ADC hardware logic automatically performs the conversions on the active channels, then waits for a new request. The Channel Enable (ADC_CHER) and Channel Disable (ADC_CHDR) Registers permit the analog channels to be enabled or disabled independently.

If the ADC is used with a PDC, only the transfers of converted data from enabled channels are performed and the resulting data buffers should be interpreted accordingly.

### 34.6.6 Sleep Mode and Conversion Sequencer

The ADC Sleep Mode maximizes power saving by automatically deactivating the ADC when it is not being used for conversions. Sleep Mode is selected by setting the SLEEP bit in the Mode Register ADC_MR.

The Sleep mode is automatically managed by a conversion sequencer, which can automatically process the conversions of all channels at lowest power consumption.

This mode can be used when the minimum period of time between 2 successive trigger events is greater than the startup period of Analog-Digital converter (See the product ADC Characteristics section).

When a start conversion request occurs, the ADC is automatically activated. As the analog cell requires a start-up time, the logic waits during this time and starts the conversion on the enabled channels. When all conversions are complete, the ADC is deactivated until the next trigger. Triggers occurring during the sequence are not taken into account.

A fast wake-up mode is available in the ADC Mode Register (ADC_MR) as a compromise between power saving strategy and responsiveness. Setting the FWUP bit to ‘1’ enables the fast wake-up mode. In fast wake-up mode the ADC cell is not fully deactivated while no conversion is requested, thereby providing less power saving but faster wakeup.

The conversion sequencer allows automatic processing with minimum processor intervention and optimized power consumption. Conversion sequences can be performed periodically using a Timer/Counter output. The periodic acquisition of several samples can be processed automatically without any intervention of the processor thanks to the PDC.

The sequence can be customized by programming the Sequence Channel Registers, ADC_SEQR1 and ADC_SEQR2 and setting to 1 the USEQ bit of the Mode Register (ADC_MR). The user can choose a specific order of channels and can program up to 16 conversions by sequence. The user is totally free to create a personal

sequence, by writing channel numbers in ADC_SEQR1 and ADC_SEQR2. Not only can channel numbers be written in any sequence, channel numbers can be repeated several times. Only enabled sequence bitfields are converted, consequently to program a 15-conversion sequence, the user can simply put a disable in ADC_CHSR[15], thus disabling the 16THCH field of ADC_SEQR2.

If all ADC channels (i.e. 16) are used on an application board, there is no restriction of usage of the user sequence. But as soon as some ADC channels are not enabled for conversion but rather used as pure digital inputs, the respective indexes of these channels cannot be used in the user sequence fields (ADC_SEQR1, ADC_SEQR2 bitfields). For example, if channel 4 is disabled (ADC_CSR[4] = 0), ADC_SEQR1, ADC_SEQR2 register bitfields USCH1 up to USCH16 must not contain the value 4. Thus the length of the user sequence may be limited by this behavior.

As an example, if only 4 channels over 16 (CH0 up to CH3) are selected for ADC conversions, the user sequence length cannot exceed 4 channels. Each trigger event may launch up to 4 successive conversions of any combination of channels 0 up to 3 but no more (i.e. in this case the sequence CH0, CH0, CH1, CH1, CH1 is impossible).

A sequence that repeats several times the same channel requires more enabled channels than channels actually used for conversion. For example, a sequence like CH0, CH0, CH1, CH1 requires 4 enabled channels (4 free channels on application boards) whereas only CH0, CH1 are really converted.

Note: The reference voltage pins always remain connected in normal mode as in sleep mode.

### 34.6.7 Comparison Window

The ADC Controller features automatic comparison functions. It compares converted values to a low threshold or a high threshold or both, according to the CMPMODE function chosen in the Extended Mode Register (ADC_EMR). The comparison can be done on all channels or only on the channel specified in CMPSEL field of ADC_EMR. To compare all channels the CMP_ALL parameter of ADC_EMR should be set.

The flag can be read on the COMPE bit of the Interrupt Status Register (ADC_ISR) and can trigger an interrupt.

The High Threshold and the Low Threshold can be read/write in the Comparison Window Register (ADC_CWR).

### 34.6.8 ADC Timings

Each ADC has its own minimal Startup Time that is programmed through the field STARTUP in the Mode Register, ADC_MR.

A minimal Tracking Time is necessary for the ADC to guarantee the best converted final value between two channel selections. This time has to be programmed through the TRACKTIM bit field in the Mode Register, ADC_MR.

**Warning:** No input buffer amplifier to isolate the source is included in the ADC. This must be taken into consideration to program a precise value in the TRACKTIM field. See the product ADC Characteristics section.

### 34.6.9 Buffer Structure

The PDC read channel is triggered each time new data is stored in ADC_LCDR register. The same structure of data is repeatedly stored in ADC_LCDR register each time a trigger event occurs. Depending on user mode of operation (ADC_MR, ADC_CHSR, ADC_SEQR1, ADC_SEQR2) the structure differs. Each data transferred to PDC buffer, carried on a half-word (16-bit), consists of last converted data right aligned and when TAG is set in ADC_EMR register, the 4 most significant bits are carrying the channel number thus allowing an easier post-processing in the PDC buffer or better checking the PDC buffer integrity.

### 34.6.10 Write Protection Registers

To prevent any single software error that may corrupt ADC behavior, certain address spaces can be write-protected by setting the WPEN bit in the "ADC Write Protect Mode Register" (ADC_WPMR).

If a write access to the protected registers is detected, then the WPVS flag in the ADC Write Protect Status Register (ADC_WPSR) is set and the field WPVSRC indicates in which register the write access has been attempted.

The WPVS flag is reset by writing the ADC Write Protect Mode Register (ADC_WPMR) with the appropriate access key, WPKEY.

The protected registers are:

"ADC Mode Register" on page 675

"ADC Channel Sequence 1 Register" on page 677

"ADC Channel Sequence 2 Register" on page 678

"ADC Channel Enable Register" on page 679

"ADC Channel Disable Register" on page 680

"ADC Extended Mode Register" on page 688

"ADC Compare Window Register" on page 689

## 34.7 Analog-to-Digital Converter (ADC) User Interface

Any offset not listed in Table 34-4 must be considered as “reserved”.

**Table 34-4. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x00 | Control Register | ADC_CR | Write-only | – |
| 0x04 | Mode Register | ADC_MR | Read-write | 0x00000000 |
| 0x08 | Channel Sequence Register 1 | ADC_SEQR1 | Read-write | 0x00000000 |
| 0x0C | Channel Sequence Register 2 | ADC_SEQR2 | Read-write | 0x00000000 |
| 0x10 | Channel Enable Register | ADC_CHER | Write-only | – |
| 0x14 | Channel Disable Register | ADC_CHDR | Write-only | – |
| 0x18 | Channel Status Register | ADC_CHSR | Read-only | 0x00000000 |
| 0x1C | Reserved | – | – | – |
| 0x20 | Last Converted Data Register | ADC_LCDR | Read-only | 0x00000000 |
| 0x24 | Interrupt Enable Register | ADC_IER | Write-only | – |
| 0x28 | Interrupt Disable Register | ADC_IDR | Write-only | – |
| 0x2C | Interrupt Mask Register | ADC_IMR | Read-only | 0x00000000 |
| 0x30 | Interrupt Status Register | ADC_ISR | Read-only | 0x00000000 |
| 0x34 | Reserved | – | – | – |
| 0x38 | Reserved | – | – | – |
| 0x3C | Overrun Status Register | ADC_OVER | Read-only | 0x00000000 |
| 0x40 | Extended Mode Register | ADC_EMR | Read-write | 0x00000000 |
| 0x44 | Compare Window Register | ADC_CWR | Read-write | 0x00000000 |
| 0x50 | Channel Data Register 0 | ADC_CDR0 | Read-only | 0x00000000 |
| 0x54 | Channel Data Register 1 | ADC_CDR1 | Read-only | 0x00000000 |
| ... | ... | ... | ... | ... |
| 0x8C | Channel Data Register 15 | ADC_CDR15 | Read-only | 0x00000000 |
| - 0x90 | Reserved | – | – | – |
| 0x98 - 0xAC | Reserved | – | – | – |
| 0xC4 - 0xE0 | Reserved | – | – | – |
| 0xE4 | Write Protect Mode Register | ADC_WPMR | Read-write | 0x00000000 |
| 0xE8 | Write Protect Status Register | ADC_WPSR | Read-only | 0x00000000 |
| 0xEC - 0xF8 | Reserved | – | – | – |
| 0xFC | Reserved | – | – | – |

Note: If an offset is not listed in the table it must be considered as “reserved”.

### 34.7.1 ADC Control Register

**Name:** ADC_CR

**Address:** 0x40038000

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | START | SWRST |

- **SWRST: Software Reset**

0 = No effect.

1 = Resets the ADC simulating a hardware reset.

- **START: Start Conversion**

0 = No effect.

1 = Begins analog-to-digital conversion.

### 34.7.2 ADC Mode Register

**Name:** ADC_MR

**Address:** 0x40038004

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| USEQ | – | – | – | TRACKTIM | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | STARTUP | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| PRESCAL | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| FREERUN | FWUP | SLEEP | LOWRES | TRGSEL | | | TRGEN |

This register can only be written if the WPEN bit is cleared in “ADC Write Protect Mode Register” on page 691.

- **TRGEN: Trigger Enable**

| Value | Name | Description |
|---|---|---|
| 0 | DIS | Hardware triggers are disabled. Starting a conversion is only possible by software. |
| 1 | EN | Hardware trigger selected by TRGSEL field is enabled. |

- **TRGSEL: Trigger Selection**

| Value | Name | Description |
|---|---|---|
| 0 | ADC_TRIG0 | External trigger |
| 1 | ADC_TRIG1 | TIO Output of the Timer Counter Channel 0 |
| 2 | ADC_TRIG2 | TIO Output of the Timer Counter Channel 1 |
| 3 | ADC_TRIG3 | TIO Output of the Timer Counter Channel 2 |
| 4 | ADC_TRIG4 | Reserved |
| 5 | ADC_TRIG5 | Reserved |
| 6 | ADC_TRIG6 | Reserved |
| 7 | – | Reserved |

- **LOWRES: Resolution**

| Value | Name | Description |
|---|---|---|
| 0 | BITS_10 | 10-bit resolution |
| 1 | BITS_8 | 8-bit resolution |

- **SLEEP: Sleep Mode**

| Value | Name | Description |
|---|---|---|
| 0 | NORMAL | Normal Mode: The ADC Core and reference voltage circuitry are kept ON between conversions |
| 1 | SLEEP | Sleep Mode: The ADC Core and reference voltage circuitry are OFF between conversions |

- **FWUP: Fast Wake Up**

| Value | Name | Description |
|---|---|---|
| 0 | OFF | Normal Sleep Mode: The sleep mode is defined by the SLEEP bit |
| 1 | ON | Fast Wake Up Sleep Mode: The Voltage reference is ON between conversions and ADC Core is OFF |

- **FREERUN: Free Run Mode**

| Value | Name | Description |
|---|---|---|
| 0 | OFF | Normal Mode |
| 1 | ON | Free Run Mode: Never wait for any trigger. |

- **PRESCAL: Prescaler Rate Selection**

  ADCClock = MCK / ( (PRESCAL+1) * 2 )

- **STARTUP: Start Up Time**

| Value | Name | Description |
|---|---|---|
| 0 | SUT0 | 0 periods of ADCClock |
| 1 | SUT8 | 8 periods of ADCClock |
| 2 | SUT16 | 16 periods of ADCClock |
| 3 | SUT24 | 24 periods of ADCClock |
| 4 | SUT64 | 64 periods of ADCClock |
| 5 | SUT80 | 80 periods of ADCClock |
| 6 | SUT96 | 96 periods of ADCClock |
| 7 | SUT112 | 112 periods of ADCClock |
| 8 | SUT512 | 512 periods of ADCClock |
| 9 | SUT576 | 576 periods of ADCClock |
| 10 | SUT640 | 640 periods of ADCClock |
| 11 | SUT704 | 704 periods of ADCClock |
| 12 | SUT768 | 768 periods of ADCClock |
| 13 | SUT832 | 832 periods of ADCClock |
| 14 | SUT896 | 896 periods of ADCClock |
| 15 | SUT960 | 960 periods of ADCClock |

- **TRACKTIM: Tracking Time**

  Tracking Time = (TRACKTIM + 1) * ADCClock periods.

- **USEQ: Use Sequence Enable**

| Value | Name | Description |
|---|---|---|
| 0 | NUM_ORDER | Normal Mode: The controller converts channels in a simple numeric order. |
| 1 | REG_ORDER | User Sequence Mode: The sequence respects what is defined in ADC_SEQR1 and ADC_SEQR2 registers. |

### 34.7.3 ADC Channel Sequence 1 Register

**Name:** ADC_SEQR1

**Address:** 0x40038008

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| USCH8 | | | | USCH7 | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| USCH6 | | | | USCH5 | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| USCH4 | | | | USCH3 | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| USCH2 | | | | USCH1 | | | |

This register can only be written if the WPEN bit is cleared in “ADC Write Protect Mode Register” on page 691.

- **USCHx: User Sequence Number x**

The sequence number x (USCHx) can be programmed by the Channel number CHy where y is the value written in this field. The allowed range is 0 up to 15. So it is only possible to use the sequencer from CH0 to CH15.

This register activates only if ADC_MR(USEQ) field is set to ‘1’.

Any USCHx field is taken into account only if ADC_CHSR(CHx) register field reads logical ‘1’ else any value written in USCHx does not add the corresponding channel in the conversion sequence.

Configuring the same value in different fields leads to multiple samples of the same channel during the conversion sequence. This can be done consecutively, or not, according to user needs.

### 34.7.4 ADC Channel Sequence 2 Register

**Name:** ADC_SEQR2

**Address:** 0x4003800C

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| USCH16 | | | | USCH15 | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| USCH14 | | | | USCH13 | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| USCH12 | | | | USCH11 | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| USCH10 | | | | USCH9 | | | |

This register can only be written if the WPEN bit is cleared in “ADC Write Protect Mode Register” on page 691.

- **USCHx: User Sequence Number x**

The sequence number x (USCHx) can be programmed by the Channel number CHy where y is the value written in this field. The allowed range is 0 up to 15. So it is only possible to use the sequencer from CH0 to CH15.

This register activates only if ADC_MR(USEQ) field is set to ‘1’.

Any USCHx field is taken into account only if ADC_CHSR(CHx) register field reads logical ‘1’ else any value written in USCHx does not add the corresponding channel in the conversion sequence.

Configuring the same value in different fields leads to multiple samples of the same channel during the conversion sequence. This can be done consecutively, or not, according to user needs.

### 34.7.5 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 |
| CH15 | CH14 | CH13 | CH12 | CH11 | CH10 | CH9 | CH8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CH7 | CH6 | CH5 | CH4 | CH3 | CH2 | CH1 | CH0 |

This register can only be written if the WPEN bit is cleared in “ADC Write Protect Mode Register” on page 691.

- **CHx: Channel x Enable**

0 = No effect.

1 = Enables the corresponding channel.

Note: if USEQ = 1 in ADC_MR register, CHx corresponds to the xth channel of the sequence described in ADC_SEQR1 and ADC_SEQR2.

### 34.7.6 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 |
| CH15 | CH14 | CH13 | CH12 | CH11 | CH10 | CH9 | CH8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CH7 | CH6 | CH5 | CH4 | CH3 | CH2 | CH1 | CH0 |

This register can only be written if the WPEN bit is cleared in “ADC Write Protect Mode Register” on page 691.

- **CHx: Channel x Disable**

0 = No effect.

1 = Disables the corresponding channel.

**Warning:** If the corresponding channel is disabled during a conversion or if it is disabled then reenabled during a conversion, its associated data and its corresponding EOC and OVRE flags in ADC_SR are unpredictable.

### 34.7.7 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 |
| CH15 | CH14 | CH13 | CH12 | CH11 | CH10 | CH9 | CH8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CH7 | CH6 | CH5 | CH4 | CH3 | CH2 | CH1 | CH0 |

- **CHx: Channel x Status**

0 = Corresponding channel is disabled.

1 = Corresponding channel is enabled.

### 34.7.8 ADC Last Converted Data Register

**Name:** ADC_LCDR

**Address:** 0x40038020

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| CHNB | | | | LDATA | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| LDATA | | | | | | | |

- **LDATA: Last Data Converted**

The analog-to-digital conversion data is placed into this register at the end of a conversion and remains until a new conversion is completed.

- **CHNB: Channel Number**

Indicates the last converted channel when the TAG option is set to 1 in ADC_EMR register. If TAG option is not set, CHNB = 0.

### 34.7.9 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 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| EOC15 | EOC14 | EOC13 | EOC12 | EOC11 | EOC10 | EOC9 | EOC8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| EOC7 | EOC6 | EOC5 | EOC4 | EOC3 | EOC2 | EOC1 | EOC0 |

- **EOCx: End of Conversion Interrupt Enable x**
- **DRDY: Data Ready Interrupt Enable**
- **GOVRE: General Overrun Error Interrupt Enable**
- **COMPE: Comparison Event Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable**
- **RXBUFF: Receive Buffer Full Interrupt Enable**

0 = No effect.

1 = Enables the corresponding interrupt.

### 34.7.10 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 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| EOC15 | EOC14 | EOC13 | EOC12 | EOC11 | EOC10 | EOC9 | EOC8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| EOC7 | EOC6 | EOC5 | EOC4 | EOC3 | EOC2 | EOC1 | EOC0 |

- **EOCx: End of Conversion Interrupt Disable x**
- **DRDY: Data Ready Interrupt Disable**
- **GOVRE: General Overrun Error Interrupt Disable**
- **COMPE: Comparison Event Interrupt Disable**
- **ENDRX: End of Receive Buffer Interrupt Disable**
- **RXBUFF: Receive Buffer Full Interrupt Disable**

0 = No effect.

1 = Disables the corresponding interrupt.

### 34.7.11 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 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| EOC15 | EOC14 | EOC13 | EOC12 | EOC11 | EOC10 | EOC9 | EOC8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| EOC7 | EOC6 | EOC5 | EOC4 | EOC3 | EOC2 | EOC1 | EOC0 |

- **EOCx: End of Conversion Interrupt Mask x**
- **DRDY: Data Ready Interrupt Mask**
- **GOVRE: General Overrun Error Interrupt Mask**
- **COMPE: Comparison Event Interrupt Mask**
- **ENDRX: End of Receive Buffer Interrupt Mask**
- **RXBUFF: Receive Buffer Full Interrupt Mask**

0 = The corresponding interrupt is disabled.

1 = The corresponding interrupt is enabled.

### 34.7.12 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 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| EOC15 | EOC14 | EOC13 | EOC12 | EOC11 | EOC10 | EOC9 | EOC8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| EOC7 | EOC6 | EOC5 | EOC4 | EOC3 | EOC2 | EOC1 | EOC0 |

- **EOCx: End of Conversion x**

0 = Corresponding analog channel is disabled, or the conversion is not finished. This flag is cleared when reading the corresponding ADC_CDRx registers.

1 = Corresponding analog channel is enabled and conversion is complete.

- **DRDY: Data Ready**

0 = No data has been converted since the last read of ADC_LCDR.

1 = At least one data has been converted and is available in ADC_LCDR.

- **GOVRE: General Overrun Error**

0 = No General Overrun Error occurred since the last read of ADC_ISR.

1 = At least one General Overrun Error has occurred since the last read of ADC_ISR.

- **COMPE: Comparison Error**

0 = No Comparison Error since the last read of ADC_ISR.

1 = At least one Comparison Error has occurred since the last read of ADC_ISR.

- **ENDRX: End of RX Buffer**

0 = The Receive Counter Register has not reached 0 since the last write in ADC_RCR or ADC_RNCR.

1 = The Receive Counter Register has reached 0 since the last write in ADC_RCR or ADC_RNCR.

- **RXBUFF: RX Buffer Full**

0 = ADC_RCR or ADC_RNCR have a value other than 0.

1 = Both ADC_RCR and ADC_RNCR have a value of 0.

### 34.7.13 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 |
| OVRE15 | OVRE14 | OVRE13 | OVRE12 | OVRE11 | OVRE10 | OVRE9 | OVRE8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| OVRE7 | OVRE6 | OVRE5 | OVRE4 | OVRE3 | OVRE2 | OVRE1 | OVRE0 |

- **OVREx: Overrun Error x**

0 = No overrun error on the corresponding channel since the last read of ADC_OVER.

1 = There has been an overrun error on the corresponding channel since the last read of ADC_OVER.

### 34.7.14 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 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | CMPALL | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CMPSEL | | | | – | – | CMPMODE | |

This register can only be written if the WPEN bit is cleared in “ADC Write Protect Mode Register” on page 691.

- **CMPMODE: Comparison Mode**

| Value | Name | Description |
|---|---|---|
| 0 | LOW | Generates an event when the converted data is lower than the low threshold of the window. |
| 1 | HIGH | Generates an event when the converted data is higher than the high threshold of the window. |
| 2 | IN | Generates an event when the converted data is in the comparison window. |
| 3 | OUT | Generates an event when the converted data is out of the comparison window. |

- **CMPSEL: Comparison Selected Channel**

If CMPALL = 0: CMPSEL indicates which channel has to be compared.

If CMPALL = 1: No effect.

- **CMPALL: Compare All Channels**

0 = Only channel indicated in CMPSEL field is compared.

1 = All channels are compared.

- **TAG: TAG of ADC_LDCR register**

0 = set CHNB to zero in ADC_LDCR.

1 = append the channel number to the conversion result in ADC_LDCR register.

### 34.7.15 ADC Compare Window Register

**Name:** ADC_CWR

**Address:** 0x40038044

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | HIGHTHRES | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| HIGHTHRES | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | LOWTHRES | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| LOWTHRES | | | | | | | |

This register can only be written if the WPEN bit is cleared in “ADC Write Protect Mode Register” on page 691.

- **LOWTHRES: Low Threshold**

Low threshold associated to compare settings of ADC_EMR register.

- **HIGHTHRES: High Threshold**

High threshold associated to compare settings of ADC_EMR register.

### 34.7.16 ADC Channel Data Register

**Name:** ADC_CDRx [x=0..15]

**Address:** 0x40038050

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | DATA | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| DATA | | | | | | | |

- **DATA: Converted Data**

The analog-to-digital conversion data is placed into this register at the end of a conversion and remains until a new conversion is completed. The Convert Data Register (CDR) is only loaded if the corresponding analog channel is enabled.

### 34.7.17 ADC Write Protect Mode Register

**Name:** ADC_WPMR

**Address:** 0x400380E4

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | WPEN |

- **WPEN: Write Protect Enable**

0 = Disables the Write Protect if WPKEY corresponds to 0x414443 (“ADC” in ASCII).

1 = Enables the Write Protect if WPKEY corresponds to 0x414443 (“ADC” in ASCII).

Protects the registers:

“ADC Mode Register” on page 675

“ADC Channel Sequence 1 Register” on page 677

“ADC Channel Sequence 2 Register” on page 678

“ADC Channel Enable Register” on page 679

“ADC Channel Disable Register” on page 680

“ADC Extended Mode Register” on page 688

“ADC Compare Window Register” on page 689

- **WPKEY: Write Protect KEY**

Should be written at value 0x414443 (“ADC” in ASCII). Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0.

### 34.7.18 ADC Write Protect 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 Protect Violation Status**

0 = No Write Protect Violation has occurred since the last read of the ADC_WPSR register.

1 = A Write Protect Violation has occurred since the last read of the ADC_WPSR register. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protect Violation Source**

When WPVS is active, this field indicates the write-protected register (through address offset or code) in which a write access has been attempted.

Reading ADC_WPSR automatically clears all fields.

# 35. Digital to Analog Converter Controller (DACC)

## 35.1 Description

The Digital-to-Analog Converter Controller (DACC) has one analog output, making it possible for the digital-to-analog conversion to drive one analog line.

The DACC supports 10-bit resolution and data to be converted are sent in a common register. External triggers, through the ext_trig pins, or internal triggers are configurable.

The DACC Controller connects with a PDC channel. This feature reduces processor intervention.

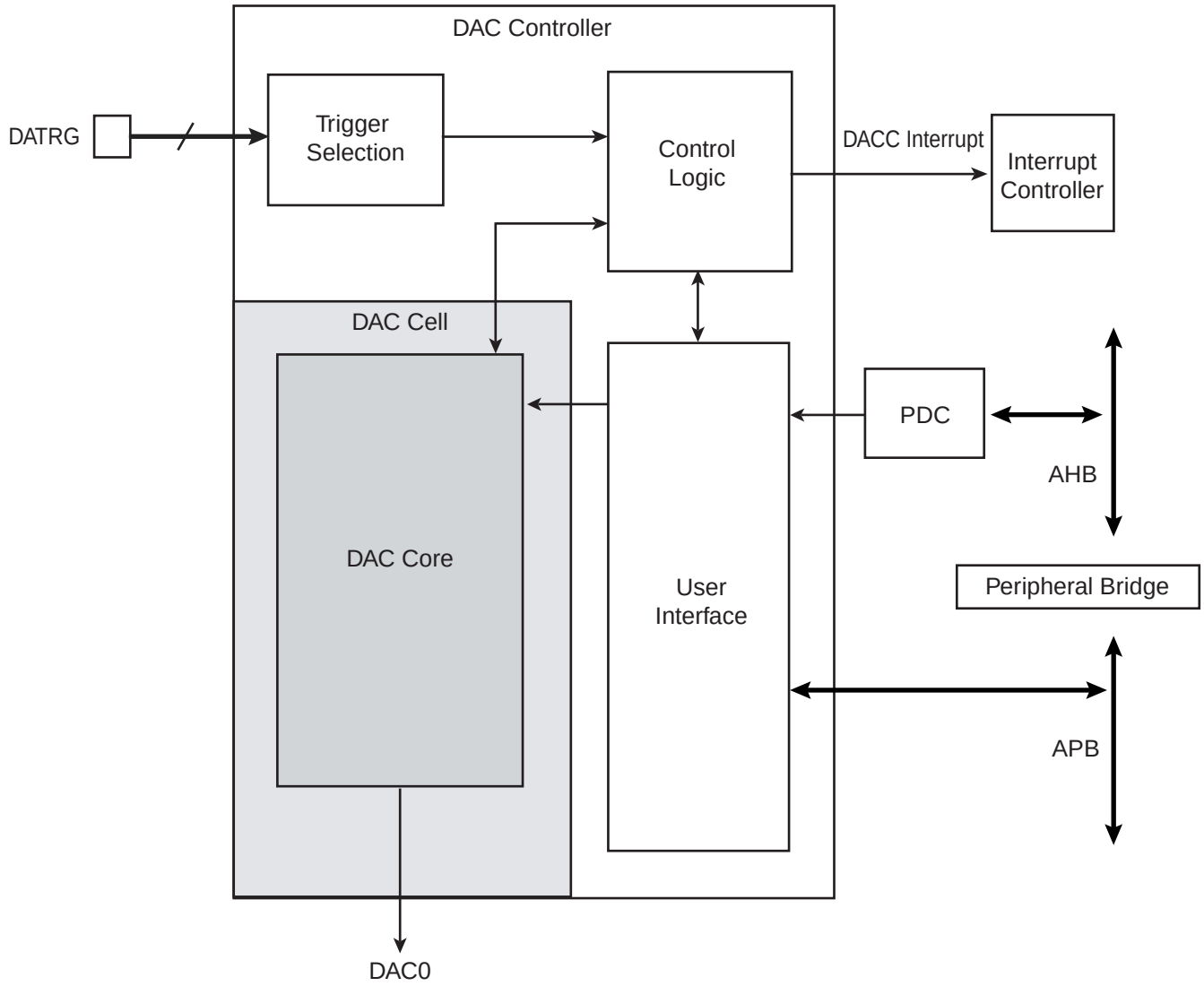
Finally, the user can configure DACC timings such as Startup Time and the Internal Trigger Period.

## 35.2 Embedded Characteristics

- 1 channel 10-bit DAC
- Up to 500 ksamples/s conversion rate
- Flexible conversion range
- Multiple trigger sources
- One PDC channel

## 35.3 Block Diagram

**Figure 35-1. Digital-to-Analog Converter Controller Block Diagram**




## 35.4 Signal Description

**Table 35-1. DAC Pin Description**

| Pin Name | Description |
|---|---|
| DAC0 | Analog output channel |
| DATRG | External triggers |

## 35.5 Product Dependencies

### 35.5.1 Power Management

The DAC can be enabled and disabled through the DACEN bit of the DACC Mode Register.

### 35.5.2 Interrupt Sources

The DACC interrupt line is connected on one of the internal sources of the Interrupt Controller. Using the DACC interrupt requires the Interrupt Controller to be programmed first.

**Table 35-2. Peripheral IDs**

| Instance | ID |
|---|---|
| DACC | 30 |

### 35.5.3 Conversion Performances

For performance and electrical characteristics of the DAC, see the product DC Characteristics section.

## 35.6 Functional Description

### 35.6.1 Digital-to-analog Conversion

The DAC uses the master clock (MCK) to perform conversions.

Once a conversion has started, the DAC will take a setup time to provide the analog result on the analog output.

Refer to the product electrical characteristics for more information.

### 35.6.2 Conversion Results

When a conversion is completed, the resulting analog value is available at the DAC channel output.

### 35.6.3 Conversion Triggers

In internal trigger mode, conversion starts as soon as the DACC is enabled, data is written in the DACC Conversion Data Register and an internal trigger event occurs (see Figure 35-2). The internal trigger frequency is configurable through the CLKDIV field of the DACC Mode Register and must not be above the maximum frequency allowed by the DAC.

In external trigger mode, the conversion waits for a rising edge event on the selected trigger to begin (see Figure 35-3).

**Warning:** Disabling the external trigger mode will automatically set the DACC in internal trigger mode.

**Figure 35-2. Internal trigger**

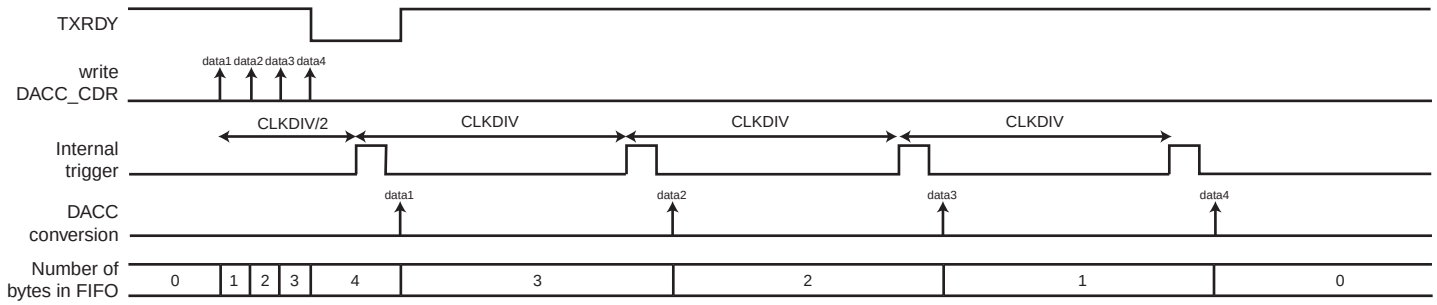
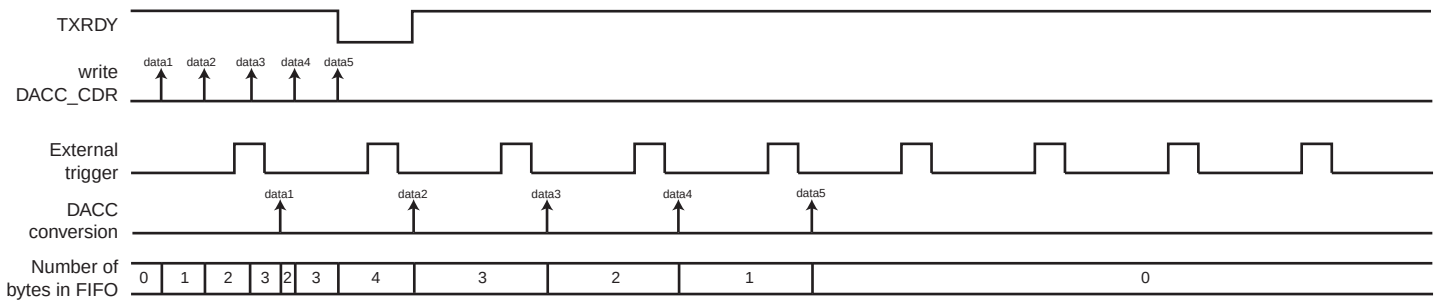



**Figure 35-3. External trigger**




### 35.6.4 Conversion FIFO

To provide flexibility and high efficiency, a 4 half-word FIFO is used to handle the data to be converted.

As long as the TXRDY flag in the DACC Interrupt Status Register is active the DAC Controller is ready to accept conversion requests by writing data in the DACC Conversion Data Register (DACC_CDR). Data which cannot be converted immediately are stored in the DACC FIFO.

When the FIFO is full or the DACC is not ready to accept conversion requests, the TXRDY flag is inactive.

**Warning:** Writing in the DACC_CDR register while TXRDY flag is inactive will corrupt FIFO data.

### 35.6.5 Conversion Width

The WORD field of the DACC Mode Register allows the user to switch between half-word and word transfer.

In half-word transfer mode only one 10-bit data item is sampled (DACC_MR[9:0]) per DACC_CDR register write.

In word transfer mode each time the DACC_CDR register is written 2 data items are sampled. First data item sampled for conversion will be DACC_CDR[9:0] and the second DACC_CDR[25:16].

### 35.6.6 DAC Timings

The DAC startup time must be defined by the user in the STARTUP field of the DACC Mode Register.

The DAC maximum clock frequency is 13 MHz, therefore the internal trigger period can be configured through the CLKDIV field of the DACC Mode Register.

### 35.6.7 Write Protection Registers

In order to bring security to the DACC, a write protection system has been implemented.

The write protection mode prevents the write of the DACC Mode Register. When this mode is enabled and the protected register is written an error is generated in the DACC Write Protect Status Register and the register write request is canceled. When a write protection error occurs, the WPROTERR flag is set and the address of the corresponding canceled register write is available in the WPROTADRR field of the DACC Write Protect Status Register.

Due to the nature of the write protection feature, enabling and disabling the write protection mode requires the use of a security code. Thus when enabling or disabling the write protection mode, the WPKEY field of the DACC Write Protect Mode Register must be filled with the “DAC” ASCII code (corresponding to 0x444143) otherwise the register write will be canceled.

## 35.7 Digital-to-Analog Converter Controller (DACC) User Interface

**Table 35-3. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x00 | Control Register | DACC_CR | Write-only | – |
| 0x04 | Mode Register | DACC_MR | Read-write | 0x00000000 |
| 0x08 | Conversion Data Register | DACC_CDR | Write-only | 0x00000000 |
| 0x0C | Interrupt Enable Register | DACC_IER | Write-only | – |
| 0x10 | Interrupt Disable Register | DACC_IDR | Write-only | – |
| 0x14 | Interrupt Mask Register | DACC_IMR | Read-only | 0x00000000 |
| 0x18 | Interrupt Status Register | DACC_ISR | Read-only | 0x00000000 |
| 0xE4 | Write Protect Mode Register | DACC_WPMR | Read-write | 0x00000000 |
| 0xE8 | Write Protect Status Register | DACC_WPSR | Read-only | 0x00000000 |
| ... | ... | ... | ... | ... |
| 0xEC - 0xFC | Reserved | – | – | – |

### 35.7.1 DACC Control Register

**Name:** DACC_CR

**Address:** 0x4003C000

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | SWRST |

- **SWRST: Software Reset**

0 = No effect.

1 = Resets the DACC simulating a hardware reset.

### 35.7.2 DACC Mode Register

**Name:** DACC_MR

**Address:** 0x4003C004

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| CLKDIV | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| CLKDIV | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| STARTUP | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | WORD | DACEN | TRGSEL | | | TRGEN |

- **TRGEN: Trigger Enable**

| TRGEN | Selected Mode |
|---|---|
| 0 | External trigger mode disabled. DACC in free running mode. |
| 1 | External trigger mode enabled. |

- **TRGSEL: Trigger Selection**

| Value | Name | Description |
|---|---|---|
| 0 | TRGSEL0 | External trigger |
| 1 | TRGSEL1 | TIO Output of the Timer Counter Channel 0 |
| 2 | TRGSEL2 | TIO Output of the Timer Counter Channel 1 |
| 3 | TRGSEL3 | TIO Output of the Timer Counter Channel 2 |
| 4 | TRGSEL4 | Reserved |
| 5 | TRGSEL5 | Reserved |
| 6 | TRGSEL6 | Reserved |
| 7 | | Reserved |

- **DACEN: DAC enable**

0 = DAC disabled.

1 = DAC enabled.

- **WORD: Word Transfer**

| WORD | Selected Resolution |
|---|---|
| 0 | Half-Word transfer |
| 1 | Word Transfer |

- **STARTUP: Startup Time Selection**

Startup Time = (STARTUP+1) * Clock period

- **CLKDIV: DAC Clock Divider for Internal Trigger**

Trigger Period = CLKDIV * Clock period

### 35.7.3 DACC Conversion Data Register

**Name:** DACC_CDR

**Address:** 0x4003C008

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| DATA | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| DATA | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| DATA | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| DATA | | | | | | | |

- **DATA: Data to Convert**

Data to convert. Can be one half-word or two half-word s depending on WORD bit in DACC_MR register.

### 35.7.4 DACC Interrupt Enable Register

**Name:** DACC_IER

**Address:** 0x4003C00C

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | TXBUFE | ENDTX | TXRDY |

- **TXRDY: Transmission Ready Interrupt Enable**

Enables ready for transmission interrupt.

- **ENDTX: End of PDC Interrupt Enable**

- **TXBUFE: Buffer Empty Interrupt Enable**

Enables end of conversion IT.

### 35.7.5 DACC Interrupt Disable Register

**Name:** DACC_IDR

**Address:** 0x4003C010

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | TXBUFE | ENDTX | TXRDY |

- **TXRDY: Transmission Ready Interrupt Disable**

Disables ready for transmission interrupt.

- **ENDTX: End of PDC Interrupt Disable**
- **TXBUFE: Buffer Empty Interrupt Disable**

### 35.7.6 DACC Interrupt Mask Register

**Name:** DACC_IMR

**Address:** 0x4003C014

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | TXBUFE | ENDTX | TXRDY |

- **TXRDY: Transmission Ready Interrupt Mask**
- **ENDTX: End of PDC Interrupt Mask**
- **TXBUFE: Buffer Empty Interrupt Mask**

### 35.7.7 DACC Interrupt Status Register

**Name:** DACC_ISR

**Address:** 0x4003C018

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | TXBUFE | ENDTX | TXRDY |

- **TXRDY: Transmission Ready Interrupt Flag**
- **ENDTX: End of PDC Interrupt Flag**
- **TXBUFE: Buffer Empty Interrupt Flag**

### 35.7.8 DACC Write Protect Mode Register

**Name:** DACC_WPMR

**Address:** 0x4003C0E4

**Access:** Read-write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| WPKEY | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| WPKEY | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | WPEN |

- **WPEN: Write Protect Enable**

0 = Disables the Write Protect if WPKEY corresponds to 0x444143 (“DAC” in ASCII).

1 = Enables the Write Protect if WPKEY corresponds to 0x444143 (“DAC” in ASCII).

Protects the DACC Mode Register.

- **WPKEY: Write Protect KEY**

This security code is needed to set/reset the WPROT bit value (see Section 35.6.7 ”Write Protection Registers” for details).

Must be filled with “DAC” ASCII code.

### 35.7.9 DACC Write Protect Status Register

**Name:** DACC_WPSR

**Address:** 0x4003C0E8

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| WPROTADDR | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | WPROTERR |

- **WPROTERR: Write protection error**

Indicates a write protection error.

- **WPROTADDR: Write protection error address**

Indicates the address of the register write request which generated the error.

# 36. Electrical Characteristics

## 36.1 Absolute Maximum Ratings

**Table 36-1. Absolute Maximum Ratings***

| |
|---|
| Operating Temperature (Industrial).....-40°C to + 85°C |
| Storage Temperature.....-60°C to + 150°C |
| Voltage on Input Pins<br>with Respect to Ground.....-0.3V to + 4.0V |
| Maximum Operating Voltage<br>(VDDCORE).....2.0V |
| Maximum Operating Voltage<br>(VDDIO).....4.0V |
| Total DC Output Current on all I/O lines<br>100-lead LQFP.....150 mA<br>100-ball TFBGA.....150 mA<br>64-lead LQFP.....100 mA<br>48-lead LQFP.....100 mA<br>64-pad QFN.....100 mA<br>48-pad QFN.....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.**

## 36.2 DC Characteristics

The following characteristics are applicable to the operating temperature range: $T_A$ = -40°C to 85°C, unless otherwise specified.

**Table 36-2. DC Characteristics**

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| $V_{DDCORE}$ | DC Supply Core | | 1.62 | 1.8 | 1.95 | V |
| $V_{DDIO}$ | DC Supply I/Os | (2) | 1.62 | 3.3 | 3.6 | V |
| $V_{DDPLL}$ | PLL and Main Oscillator Supply | | 1.62 | | 1.95 | V |
| $V_{IL}$ | Input Low-level Voltage | PA0–PA31, PB0–PB14, PC0–PC31 | -0.3 | | $0.3 \times V_{DDIO}$ | V |
| $V_{IH}$ | Input High-level Voltage | PA0–PA31, PB0–PB14, PC0–PC31 | $0.7 \times V_{DDIO}$ | | $V_{DDIO}$ + 0.3V | V |
| $V_{OH}$ | Output High-level Voltage | PA0–PA31, PB0–PB14, PC0–PC31<br>$I_{OH}$ ~ 0<br>$I_{OH}$ > 0 (See $I_{OH}$ details below) | | | <br>0.2<br>0.4 | V |
| $V_{OL}$ | Output Low-level Voltage | PA0–PA31, PB0–PB14, PC0–PC31<br>$I_{OH}$ ~ 0<br>$I_{OH}$ > 0 (See $I_{OL}$ details below) | <br>$V_{DDIO}$ - 0.2V<br>$V_{DDIO}$ - 0.4V | | | V |
| $V_{hys}$ | Hysteresis Voltage | PA0–PA31, PB0–PB14, PC0–PC31 (Hysteresis mode enabled) | 150 | | 500 | mV |
| | | ERASE, TST, JTAGSEL | 230 | | 700 | mV |
| $I_{OH}$ | Source Current | 1.62V < VDDIO < 1.95V;<br>$V_{OH} = V_{DDIO}$ - 0.4<br>- PA14 (SPCK), pins<br>- PA0–PA3<br>- Other pins(1) | | | <br><br>-6<br>-6<br>-3 | mA |
| | | 3.0V < VDDIO < 3.6V; $V_{OH} = V_{DDIO}$ - 0.4<br>- PA14 (SPCK), pins<br>- PA0–PA3<br>- Other pins(1) | | | <br>-6<br>-6<br>-3 | |
| | | 1.62V < VDDIO < 3.6V; $V_{OH} = V_{DDIO}$ - 0.4<br>- NRST | | | -2 | |
| | | Relaxed Mode:<br>3.0V < VDDIO < 3.6V; $V_{OH}$ = 2.2V<br>- PA14 (SPCK), pins<br>- PA0–PA3<br>- Other pins(1) | | | <br>-14<br>-16<br>-8 | |

**Table 36-2. DC Characteristics (Continued)**

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| $I_{OL}$ | Sink Current | 1.62V < VDDIO < 1.95V; $V_{OL}$ = 0.4V<br>- PA14 (SPCK), pins<br>- PA0–PA3<br>- Other pins(1) | | | <br>8<br>8<br>4 | mA |
| | | 3.0V < VDDIO < 3.6V; $V_{OL}$ = 0.4V<br>- PA14 (SPCK), pins<br>- PA0–PA3<br>- Other pins(1) | | | <br>9<br>12<br>6 | |
| | | 1.62V < VDDIO < 3.6V; $V_{OL}$ = 0.4V<br>- NRST | | | 2 | |
| | | Relaxed Mode:<br>3.0V < VDDIO < 3.6V; $V_{OL}$ = 0.6V<br>- PA14 (SPCK), pins<br>- PA0–PA3<br>- Other pins(1) | | | <br>14<br>18<br>9 | |
| $I_{IL\_lkg}$ | Input Low Leakage Current | No pull-up or pull-down; $V_{IN}$ = GND; $V_{DDIO}$ Max.<br>(Typ: $T_A$ = 25°C, Max: $T_A$ = 85°C) | | 5 | 30 | nA |
| $I_{IH\_lkg}$ | Input High Leakage Current | No pull-up or pull-down; $V_{IN}$ = VDD; $V_{DDIO}$ Max.<br>(Typ: $T_A$ = 25°C, Max: $T_A$ = 85°C) | | 2 | 18 | nA |
| $R_{PULLUP}$ | Pull-up Resistor | PA0–PA31, PB0–PB14, PC0–PC31 | 50 | 100 | 175 | kΩ |
| | | NRST | 50 | 100 | 175 | |
| $R_{PULLDOWN}$ | Pull-down Resistor | PA0–PA31, PB0–PB14, PC0–PC31 | 50 | 100 | 175 | kΩ |
| | | TST, JTAGSEL | 10 | 100 | 20 | |
| $R_{ODT}$ | On-die Series Termination Resistor | PA4–PA31, PB0–PB14, PC0–PC31<br>PA0–PA3 | | 36<br>18 | | Ω |

Note: 1. PA[4–13], PA[15–28], PB[0–14], PC[0–31]

2. Refer to Section 5.2.2 “VDDIO Versus VDDIN”

**Table 36-3. 1.8V Voltage Regulator Characteristics**

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| $V_{DDIN}$ | DC Input Voltage Range | (3) | 1.8 | 3.3 | 3.6 | V |
| $V_{DDOUT}$ | DC Output Voltage | Normal Mode | | 1.8 | | V |
| | | Standby Mode | | 0 | | |
| $V_{O(accuracy)}$ | Output Voltage Accuracy | $I_{LOAD}$ = 0.5– 60 mA | -3 | | 3 | % |
| $I_{LOAD}$ | Maximum DC Output Current | $V_{DDIN}$ > 2V | | | 60 | mA |
| | | $V_{DDIN} \leq$ 2V | | | 40 | |
| $V_{DROPOUT}$ | Dropout Voltage | $V_{DDIN}$ = 1.8V<br>$I_{LOAD}$ = 40 mA | | | 150 | mV |
| $V_{LINE}$ | Line Regulation | $V_{DDIN}$ 2.7–3.6 V<br>$I_{LOAD}$ MAX | | 20 | 50 | mV |
| $V_{LINE-TR}$ | Transient Line Regulation | $V_{DDIN}$ 2.7–3.6 V<br>$I_{LOAD}$ Max<br>$t_r = t_f$ = 5 µs<br>$CD_{OUT}$ = 1 µF | | 50 | 100 | mV |
| $V_{LOAD}$ | Load Regulation | $V_{DDIN} \geq$ 2.2V<br>$I_{LOAD}$ = 10% to 90% MAX | | 20 | 50 | mV |
| $V_{LOAD-TR}$ | Transient Load Regulation | $V_{DDIN} \geq$ 2.2V<br>$I_{LOAD}$ = 10% to 90% MAX<br>$t_r = t_f$ = 5 µs<br>$CD_{OUT}$ = 1 µF | | 50 | 100 | mV |
| $I_Q$ | Quiescent Current | Normal Mode @ $I_{LOAD}$ = 0 mA | | 7 | 10 | µA |
| | | Normal Mode @ $I_{LOAD}$ = 60 mA | | 700 | 1200 | |
| | | Standby Mode | | | 1 | |
| $CD_{IN}$ | Input Decoupling Capacitor | (1) | | 10 | | µF |
| $CD_{OUT}$ | Output Decoupling Capacitor | (2) | 0.75 | 1 | | µF |
| | | ESR | 0.1 | | 10 | Ω |
| $t_{on}$ | Turn on Time | $CD_{OUT}$ = 1 µF, $V_{DDOUT}$ reaches $V_{T+}$ (core power brownout detector supply rising threshold) | | 100 | 200 | µs |
| $t_{off}$ | Turn off Time | $CD_{OUT}$ = 1 µF | | | 40 | ms |

Notes:
1. A 10 µF or higher ceramic capacitor must be connected between VDDIN and the closest GND pin of the device. This large decoupling capacitor is mandatory to reduce startup current, improving transient response and noise rejection.
2. To ensure stability, an external 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 0.1 to 10 ohms. Solid tantalum, and multilayer ceramic capacitors are all suitable as output capacitor.
   A 100nF bypass capacitor between VDDOUT and the closest GND pin of the device helps decreasing output noise and improves the load transient response.
3. Refer to Section 5.2.2 “VDDIO Versus VDDIN”

**Table 36-4. Core Power Supply Brownout Detector Characteristics**

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| $V_{T-}$ | Supply Falling Threshold(1) | | 1.52 | 1.55 | 1.58 | V |
| $V_{hys-}$ | Hysteresis $V_{T-}$ | | | 25 | 38 | mV |
| $V_{T+}$ | Supply Rising Threshold | | 1.35 | 1.50 | 1.62 | V |
| $t_{RST}$ | Reset Period | | 100 | – | 350 | µs |
| $V_{hys+}$ | Hysteresis $V_{T+}$ | | 100 | 170 | 250 | mV |
| $I_{DDON}$ | Current Consumption on VDDCORE | Brownout Detector enabled | | 18 | | µA |
| $I_{DDOFF}$ | | Brownout Detector disabled | | | 200 | nA |
| $t_{d-}$ | $V_{T-}$ detection propagation time | VDDCORE = $V_{T+}$ to ($V_{T-}$ - 100mV) | | | 200 | ns |
| $t_{START}$ | Startup Time | From disabled state to enabled state | | 100 | 200 | µs |

Note: 1. The product is guaranteed to be functional at $V_{T-}$

**Figure 36-1. Core Brownout Output Waveform**

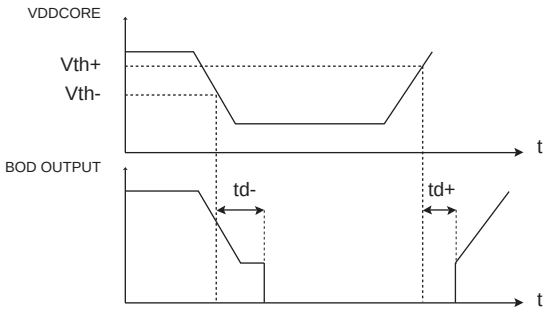


**Table 36-5. VDDIO Supply Monitor**

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| $V_T$ | Supply Monitor Threshold | 16 selectable steps of 100mV | 1.9 | | 3.4 | V |
| $V_{T(accuracy)}$ | Threshold Level Accuracy | | -1.5 | | +1.5 | % |
| $V_{hys}$ | Hysteresis | | | 20 | 30 | mV |
| $I_{DDON}$ | Current Consumption on VDDCORE | Supply Monitor enabled | | 18 | 28 | µA |
| $I_{DDOFF}$ | | Supply Monitor disabled | | | 1 | |
| $t_{START}$ | Startup Time | From disabled state to enabled state | | | 140 | µs |

**Figure 36-2. VDDIO Supply Monitor**

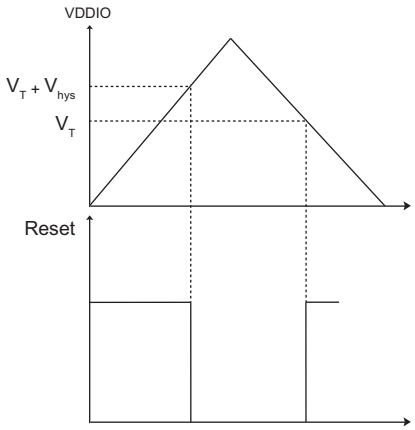



**Table 36-6. Zero-Power-on Reset Characteristics**

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| $V_{T+}$ | Threshold Voltage Rising | At startup | 1.46 | 1.55 | 1.60 | V |
| $V_{T-}$ | Threshold Voltage Falling | | 1.36 | 1.45 | 1.54 | V |
| $t_{RST}$ | Reset Period | | 40 | 90 | 150 | µs |

**Figure 36-3. Zero-Power-on Reset Characteristics**

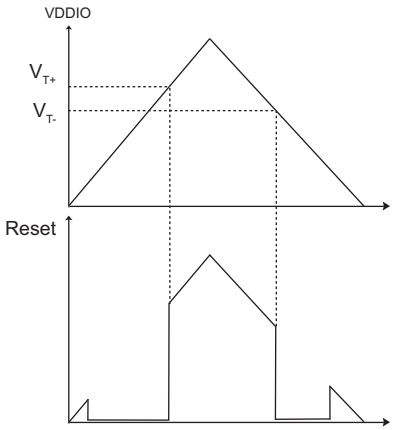


**Table 36-7. DC Flash Characteristics**

| Symbol | Parameter | Conditions | Typ | Max | Unit |
|---|---|---|---|---|---|
| $I_{DD(standby)}$ | Standby Current | @ 25°C onto VDDCORE = 1.8V | 3.2 | 4 | µA |
| | | @ 85°C onto VDDCORE = 1.8V | 6 | 8 | |
| | | @ 25°C onto VDDCORE = 1.95V | 4 | 4.8 | |
| | | @ 85°C onto VDDCORE = 1.95V | 6.5 | 9 | |
| $I_{CC}$ | Active Current | 128-bit Mode Read Access: | | | mA |
| | | Maximum Read Frequency onto VDDCORE = 1.8V @ 25 °C | 19 | 22.5 | |
| | | Maximum Read Frequency onto VDDCORE = 1.95V @ 25 °C | 25 | 30 | |
| | | 64-bit Mode Read Access: | | | |
| | | Maximum Read Frequency onto VDDCORE = 1.8V @ 25 °C | 8 | 11 | |
| | | Maximum Read Frequency onto VDDCORE = 1.95V @ 25 °C | 12.5 | 15 | |
| | | Write onto VDDCORE = 1.8V @ 25 °C | 7.5 | 9.5 | |
| | | Write onto VDDCORE = 1.95V @ 25 °C | 5.5 | 6.0 | |

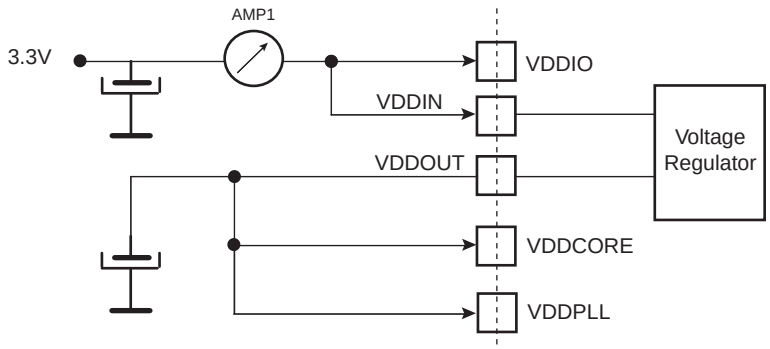
## 36.3 Power Consumption

- Power consumption of the device according to the different Low Power Mode Capabilities (Backup, Wait, Sleep) and Active Mode
- Power consumption on power supply in different modes: Backup, Wait, Sleep, and Active
- Power consumption by peripheral: calculated as the difference in current measurement after having enabled then disabled the corresponding clock

### 36.3.1 Backup Mode Current Consumption

The Backup mode configuration and measurements are defined as follows.

**Figure 36-4. Measurement Setup**



#### 36.3.1.1 Configuration A: Embedded Slow Clock RC Oscillator Enabled

- Supply Monitor on VDDIO is disabled
- RTC is running
- RTT is enabled in 1 Hz mode
- One WKUPx enabled
- Current measurement on AMP1 (See Figure 36-4)

#### 36.3.1.2 Configuration B: 32.768 kHz Crystal Oscillator Enabled

- Supply Monitor on VDDIO is disabled
- RTC is running
- RTT is enabled in 1 Hz mode
- One WKUPx enabled
- Current measurement on AMP1 (See Figure 36-4)

**Table 36-8. Power Consumption for Backup Mode (SAM3N4/2/1 MRL A)**

| Conditions | Total Consumption (AMP1) Configuration A | Total Consumption (AMP1) Configuration B | Unit |
|---|---|---|---|
| VDDIO = 3.3V @ 25°C | 2.85 | 3.25 | µA |
| VDDIO = 3.0V @ 25°C | 2.55 | 2.96 | |
| VDDIO = 2.5V @ 25°C | 2.1 | 2.50 | |
| VDDIO = 1.8V @ 25°C | 1.56 | 1.89 | |
| VDDIO = 3.3V @ 85°C | 10.5 | 10.9 | µA |
| VDDIO = 3.0V @ 85°C | 9.56 | 9.98 | |
| VDDIO = 2.5V @ 85°C | 7.88 | 8.3 | |
| VDDIO = 1.8V @ 85°C | 5.85 | 6.25 | |

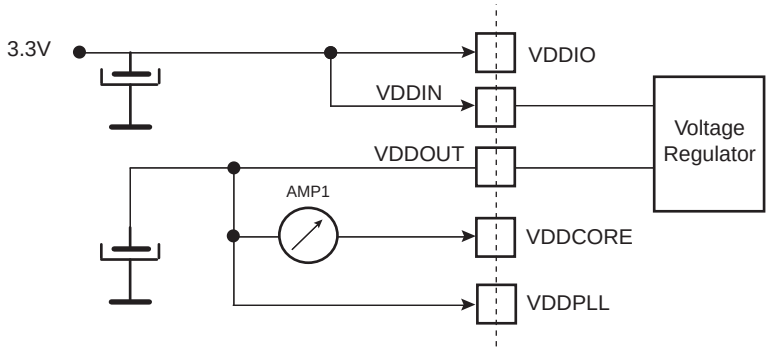
**Table 36-9. Power Consumption for Backup Mode (SAM3N1 MRL B and SAM3N0/00 MRL A)**

| Conditions | Total Consumption (AMP1) Configuration A | Total Consumption (AMP1) Configuration B | Unit |
|---|---|---|---|
| VDDIO = 3.3V @ 25°C | 1.55 | 1.60 | µA |
| VDDIO = 3.0V @ 25°C | 1.40 | 1.45 | |
| VDDIO = 2.5V @ 25°C | 1.20 | 1.25 | |
| VDDIO = 1.8V @ 25°C | 1.20 | 1.25 | |
| VDDIO = 3.3V @ 85°C | 5.50 | 5.80 | µA |
| VDDIO = 3.0V @ 85°C | 5.25 | 5.50 | |
| VDDIO = 2.5V @ 85°C | 4.75 | 4.90 | |
| VDDIO = 1.8V @ 85°C | 4.45 | 4.60 | |

### 36.3.2 Sleep and Wait Mode Current Consumption

The Wait mode and Sleep mode configuration and measurements are defined below.

**Figure 36-5. Measurement Setup for Sleep Mode**




#### 36.3.2.1 Sleep Mode

- Core Clock OFF
- Master Clock (MCK) running at various frequencies with PLL or the fast RC oscillator
- Fast startup through pins WKUP0–15
- Current measurement as shown in Figure 36-6
- All peripheral clocks deactivated

Table 36-10 gives current consumption in typical conditions.

**Table 36-10. Typical Current Consumption for Sleep Mode**

| Conditions | VDDCORE Consumption (AMP1) | Total Consumption (AMP2) | Unit |
|---|---|---|---|
| See Figure 36-5<br>@ 25°C<br>MCK = 48 MHz<br>There is no activity on the I/Os of the device. | 6.4 | 8.4 | mA |

**Figure 36-6. Current Consumption in Sleep Mode (AMP1) Versus Master Clock Ranges (refer to Table 36-10)**

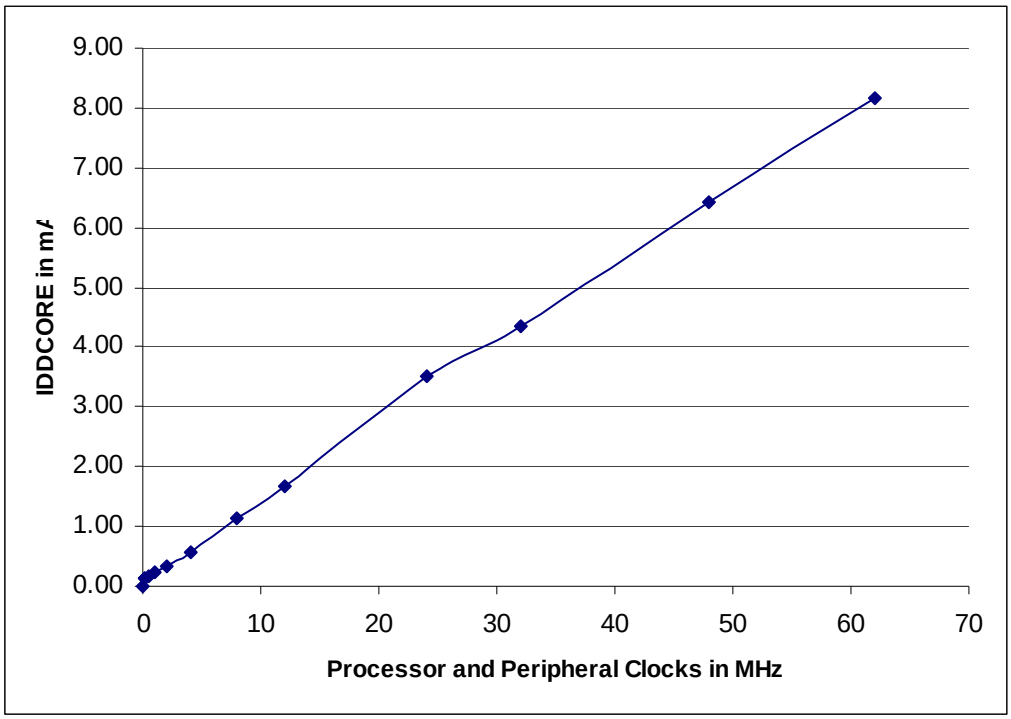
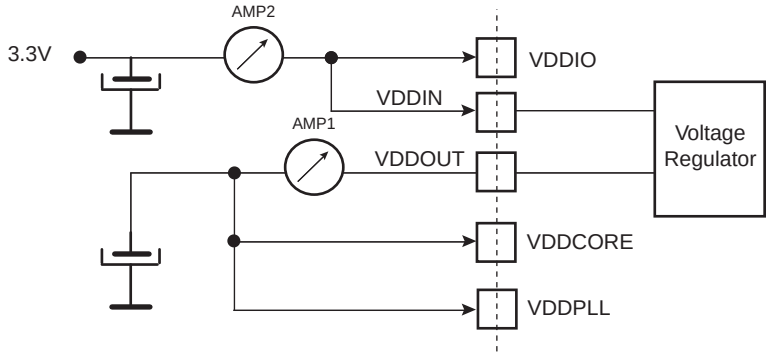



**Table 36-11. Sleep Mode Current Consumption Versus Master Clock (MCK) Variation**

| Core Clock/MCK (MHz) | VDDCORE Consumption (AMP1) | Total Consumption (AMP2) | Unit |
|---|---|---|---|
| 62 | 8.16 | 10.7 | mA |
| 48 | 6.4 | 8.4 | |
| 32 | 4.3 | 5.65 | |
| 24 | 3.5 | 5.5 | |
| 12 | 1.68 | 1.71 | |
| 8 | 1.13 | 1.16 | |
| 4 | 0.56 | 0.57 | |
| 2 | 0.33 | 0.35 | |
| 1 | 0.22 | 0.23 | |
| 0.5 | 0.16 | 0.17 | |
| 0.25 | 0.14 | 0.16 | |
| 0.125 | 0.12 | 0.13 | |
| 0.032 | 0.01 | 0.02 | |

### 36.3.2.2 Wait Mode

**Figure 36-7. Measurement Setup for Wait Mode**




- Core Clock and Master Clock stopped
- Current measurement as shown in Figure 36-7
- All peripheral clocks deactivated

Table 36-12 gives current consumption in typical conditions.

**Table 36-12. Typical Current Consumption in Wait Mode**

| Conditions | VDDOUT Consumption (AMP1) | Total Consumption (AMP2) | Unit |
|---|---|---|---|
| See Figure 36-7<br>@ 25°C<br>There is no activity on the I/Os of the device. | 5.7 | 14.9 | µA |

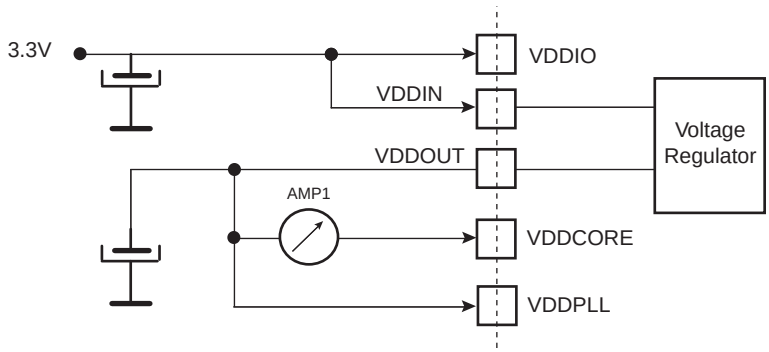
### 36.3.3 Active Mode Power Consumption

The Active Mode configuration and measurements are defined as follows:

- VDDIO = VDDIN = 3.3V
- VDDCORE = 1.8V (Internal Voltage regulator used) and 1.62V (external supply)
- $T_A$ = 25°C
- Recursive Fibonacci Algorithm or division operation running from Flash memory
- All peripheral clocks are deactivated.
- Master Clock (MCK) running at various frequencies with PLL or the fast RC oscillator
- Current measurement on AMP1 (VDDCORE)

Note: 1. Recursive Fibonacci is a high computation test whereas division operation is a low computation test.

**Figure 36-8. Active Mode Measurement Setup**



#### 36.3.3.1 Active Power Consumption with VDDCORE @ 1.8V

**Table 36-13. Master Clock (MCK) and Core Clock Variation (SAM3N4/2/1 MRL A)**

| Core Clock/MCK (MHz) | AMP1 (VDDOUT) Consumption | | | | Unit |
|---|---|---|---|---|---|
| | Division | | Fibonacci | | |
| | 128-bit Flash access | 64-bit Flash access | 128-bit Flash access | 64-bit Flash access | |
| 62 | 30 | 25.3 | 31.4 | 28.55 | mA |
| 48 | 24.45 | 20.6 | 26.2 | 23.15 | |
| 32 | 15.6 | 14.3 | 20 | 17.7 | |
| 24 | 11.4 | 10.5 | 15.6 | 15 | |
| 12 | 6.45 | 5.7 | 9.2 | 8.5 | |
| 8 | 4.9 | 4.2 | 7.1 | 6.4 | |
| 4 | 4.3 | 2.9 | 4.5 | 2.9 | |
| 2 | 2.2 | 1.5 | 2.4 | 1.7 | |
| 1 | 1.1 | 0.84 | 1.2 | 0.9 | |

**Table 36-14. Master Clock (MCK) and Core Clock Variation (SAM3N1 MRL B and SAM3N0/00 MRL B)**

| Core Clock/MCK (MHz) | AMP1 (VDDOUT) Consumption | | | | Unit |
|---|---|---|---|---|---|
| | Division | | Fibonacci | | |
| | 128-bit Flash access | 64-bit Flash access | 128-bit Flash access | 64-bit Flash access | |
| 62 | 18.5 | 17.7 | 21.28 | 23.4 | mA |
| 48 | 14.43 | 13.76 | 16.68 | 18.1 | |
| 32 | 9.87 | 9.32 | 11.26 | 12.26 | |
| 24 | 8.91 | 8.44 | 9.84 | 10.64 | |
| 12 | 3.34 | 3.07 | 3.74 | 4.17 | |
| 8 | 2.25 | 2.07 | 2.52 | 2.81 | |
| 4 | 1.07 | 0.98 | 1.19 | 1.32 | |
| 2 | 0.59 | 0.55 | 0.65 | 0.72 | |
| 1 | 0.35 | 0.33 | 0.38 | 0.41 | |
| 0.5 | 0.23 | 0.22 | 0.24 | 0.26 | |
| 0.25 | 0.17 | 0.16 | 0.18 | 0.18 | |
| 0.125 | 0.14 | 0.13 | 0.14 | 0.15 | |
| 0.032 | 0.013 | 0.012 | 0.014 | 0.015 | |

### 36.3.3.2 Active Power Consumption with VDDCORE @ 1.62V

**Table 36-15. Master Clock (MCK) and Core Clock Variation (SAM3N4/2/1 MRL A)**

| Core Clock/MCK (MHz) | AMP1 (VDDOUT) Consumption | | | | Unit |
|---|---|---|---|---|---|
| | Division | | Fibonacci | | |
| | 128-bit Flash access | 64-bit Flash access | 128-bit Flash access | 64-bit Flash access | |
| 62 | 25.7 | 22.6 | 27.05 | 25.2 | mA |
| 48 | 20.8 | 18 | 23.2 | 20.4 | |
| 32 | 14.1 | 12.5 | 17.2 | 15.75 | |
| 24 | 11.1 | 9.25 | 13.65 | 13.2 | |
| 12 | 5.6 | 5 | 7.9 | 7.36 | |
| 8 | 4.2 | 3.6 | 5.9 | 5.41 | |
| 4 | 3.55 | 2.4 | 3.6 | 5.5 | |
| 2 | 1.84 | 1.3 | 1.88 | 1.3 | |
| 1 | 1 | 0.72 | 1.2 | 0.72 | |

**Table 36-16. Master Clock (MCK) and Core Clock Variation (SAM3N1 MRL B and SAM3N0/00 MRL B)**

| Core Clock/MCK (MHz) | AMP1 (VDDOUT) Consumption | | | | Unit |
|---|---|---|---|---|---|
| | Division | | Fibonacci | | |
| | 128-bit Flash access | 64-bit Flash access | 128-bit Flash access | 64-bit Flash access | |
| 62 | 16.72 | 16.17 | 19.31 | 20.99 | mA |
| 48 | 12.97 | 12.38 | 14.95 | 16.14 | |
| 32 | 8.81 | 8.38 | 10.12 | 10.91 | |
| 24 | 8.02 | 7.69 | 8.96 | 9.59 | |
| 12 | 2.92 | 2.71 | 3.30 | 3.65 | |
| 8 | 1.96 | 1.82 | 2.22 | 2.45 | |
| 4 | 0.93 | 0.87 | 1.05 | 1.16 | |
| 2 | 0.52 | 0.48 | 0.58 | 0.63 | |
| 1 | 0.31 | 0.29 | 0.34 | 0.37 | |
| 0.5 | 0.21 | 0.20 | 0.22 | 0.23 | |
| 0.25 | 0.15 | 0.15 | 0.16 | 0.17 | |
| 0.125 | 0.13 | 0.12 | 0.13 | 0.13 | |
| 0.032 | 0.011 | 0.010 | 0.012 | 0.013 | |

### 36.3.4 Peripheral Power Consumption in Active Mode

**Table 36-17. Power Consumption on VDDCORE [1] (SAM3N4/2/1 MRL A)**

| Peripheral | Consumption (Typ) | Unit |
|---|---|---|
| PIO Controller A (PIOA) | 10 | µA/MHz |
| PIO Controller B (PIOB) | 5.15 | |
| PIO Controller C (PIOC) | 9.8 | |
| UART0 (PDC) | 14 | |
| UART1 (no PDC) | 3.8 | |
| USART0 (PDC) | 21.2 | |
| USART1 (no PDC) | 8.2 | |
| PWM | 10.55 | |
| TWI0 (PDC) | 15.25 | |
| TWI1 (no PDC) | 4.6 | |
| SPI | 12.5 | |
| TC0, TC3 | 9 | |
| TC1, TC2, TC4, TC5 | 5 | |
| ADC | 17.6 | |
| DACC | 7.75 | |

Note: 1. VDDIO = 3.3V, VDDCORE = 1.80V, $T_A$ = 25°C

**Table 36-18. Power Consumption on VDDCORE [1] (SAM3N1 MRL B, SAM3N0/00 MRLA)**

| Peripheral | Consumption (Typ) | Unit |
|---|---|---|
| PIO Controller A (PIOA) | 11 | µA/MHz |
| PIO Controller B (PIOB) | 6.78 | |
| PIO Controller C (PIOC) | 12.72 | |
| UART0 (PDC) | 8.9 | |
| UART1 (no PDC) | 3.02 | |
| USART0 (PDC) | 15.58 | |
| USART1 (no PDC) | 10.04 | |
| PWM | 8.10 | |
| TWI0 (PDC) | 9.54 | |
| TWI1 (no PDC) | 3.61 | |
| SPI | 8.17 | |
| TC0, TC3 | 7.1 | |
| TC1, TC2, TC4, TC5 | 4 | |
| ADC | 10.4 | |
| DACC | 4.54 | |

Note: 1. VDDIO = 3.3V, VDDCORE = 1.80V, $T_A$ = 25°C

## 36.4 Crystal Oscillators Characteristics

### 36.4.1 32 kHz RC Oscillator Characteristics

**Table 36-19. 32 kHz RC Oscillator Characteristics**

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| $f_{OSC}$ | RC Oscillator Frequency | | 20 | 32 | 44 | kHz |
| | Frequency Supply Dependency | | -3 | | 3 | %/V |
| | Frequency Temperature Dependency | Over temperature range -40 to 85°C versus $T_A$ 25°C | -11 | | 11 | % |
| Duty | Duty Cycle | | 45 | 50 | 55 | % |
| $t_{START}$ | Startup Time | | | | 100 | µs |
| $I_{DDON}$ | Current Consumption | After startup time<br>Temp. range = -40 to 85°C<br>Typical consumption at 2.2V supply and $T_A$ = 25°C | | 540 | 870 | nA |

### 36.4.2 4/8/12 MHz RC Oscillators Characteristics

**Table 36-20. 4/8/12 MHz RC Oscillators Characteristics**

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| $f_{OSC}$ | RC Oscillator Frequency Range | (1) | 4 | | 12 | MHz |
| $ACC_4$ | 4 MHz Total Accuracy | -40°C < Temp < +85°C<br>4 MHz output selected (1)(2) | | | ±35 | % |
| $ACC_8$ | 8 MHz Total Accuracy | -40°C < Temp < +85°C<br>8 MHz output selected (1)(3) | | | ±3.5 | % |
| | | -20°C < Temp < +85°C<br>8 MHz output selected (1)(3) | | | ±2.5 | % |
| | | 0°C < Temp < +70°C<br>8 MHz output selected (1)(3) | | | ±2 | % |
| $ACC_{12}$ | 12 MHz Total Accuracy | -40°C < Temp < +85°C<br>12 MHz output selected (1)(3) | | | ±3.5 | % |
| | | -20°C < Temp < +85°C<br>12 MHz output selected (1)(3) | | | ±2.7 | % |
| | | 0°C < Temp < +70°C<br>12 MHz output selected (1)(3) | | | ±2 | % |
| | Frequency deviation versus trimming code | 8 MHz<br>12 MHz | | 49.2<br>37.5 | | kHz/trimming code |
| Duty | Duty Cycle | | 45 | 50 | 55 | % |
| $t_{START}$ | Startup Time | | | | 10 | µs |
| $I_{DDON}$ | Active Current Consumption | 4 MHz<br>8 MHz<br>12 MHz | | 80<br>105<br>145 | 120<br>160<br>210 | µA |

Notes:
1. Frequency range can be configured in the Supply Controller registers.
2. Not trimmed from factory
3. After Trimming from factory

The 4/8/12 MHz Fast RC oscillator is calibrated in production. This calibration can be read through the Get CALIB Bit command (see Section 19. “Enhanced Embedded Flash Controller (EEFC)”) and the frequency can be trimmed by software through the PMC. Figure 36-9 and Figure 36-10 show the frequency versus trimming for 8 and 12 MHz.

**Figure 36-9. RC 8 MHz Trimming**

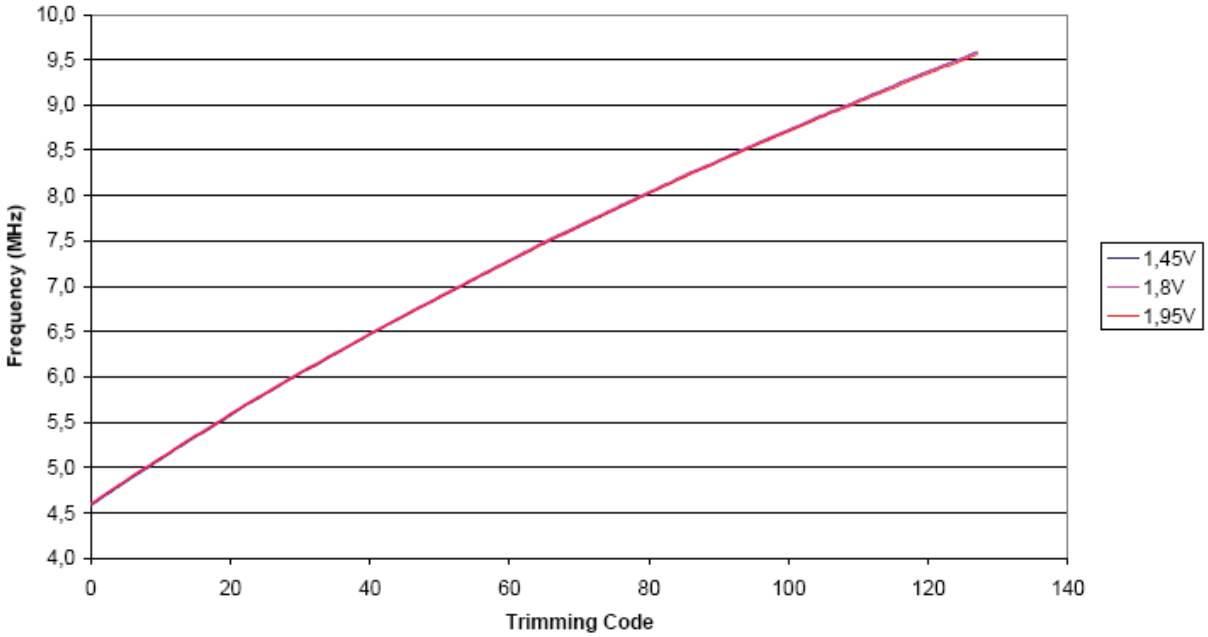
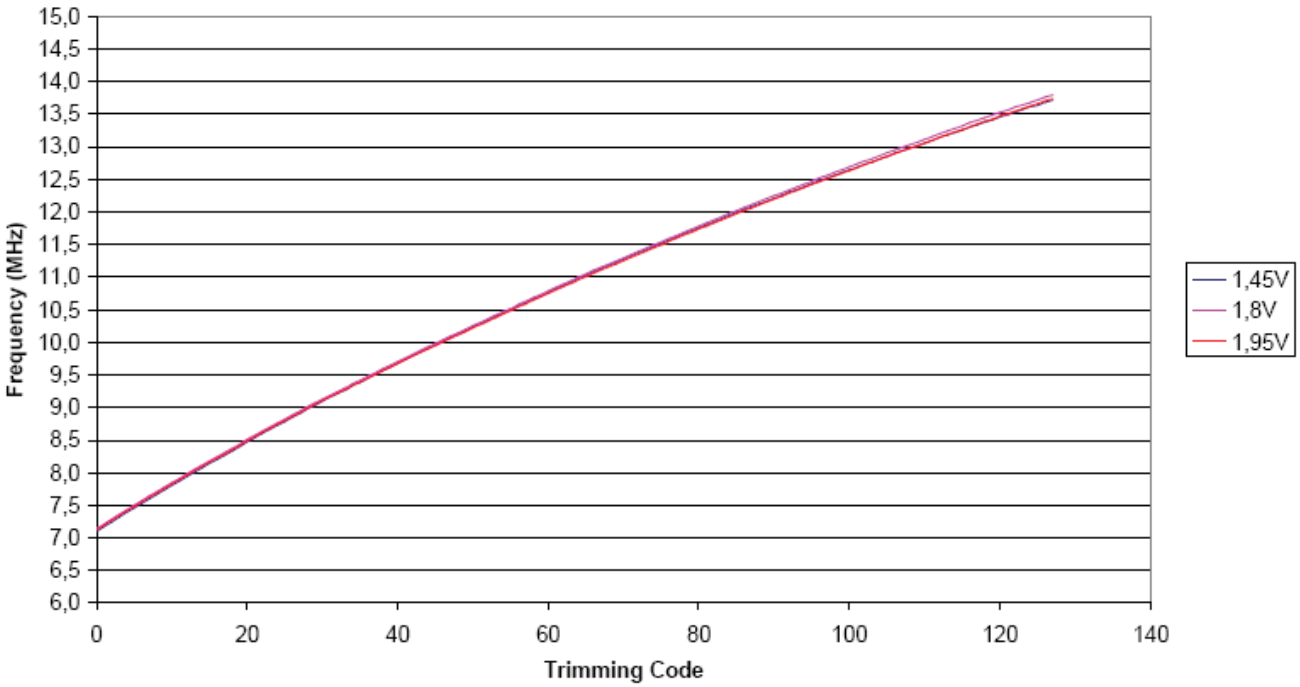



**Figure 36-10. RC 12 MHz Trimming**



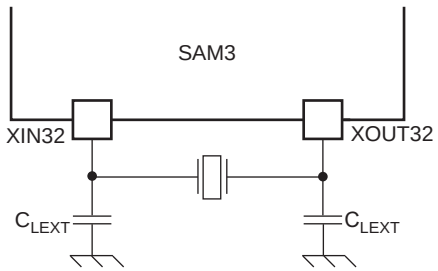
### 36.4.3 32.768 kHz Crystal Oscillator Characteristics

**Table 36-21. 32.768 kHz Crystal Oscillator Characteristics**

| Symbol | Parameter | Conditions | | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|---|
| $f_{OSC}$ | Operating Frequency | Normal mode with crystal | | | | 32.768 | kHz |
| $V_{rip(VDDIO)}$ | Supply Ripple Voltage (on VDDIO) | RMS value, 10 kHz to 10 MHz | | | | 30 | mV |
| | Duty Cycle | | | 40 | 50 | 60 | % |
| $t_{START}$ | Startup Time | $R_S$ < 50KΩ(1) | $C_{crystal}$ = 12.5 pF | | | 900 | ms |
| | | | $C_{crystal}$ = 6 pF | | | 300 | |
| | | $R_S$ < 100KΩ(1) | $C_{crystal}$ = 12.5 pF | | | 1200 | |
| | | | $C_{crystal}$ = 6 pF | | | 500 | |
| $I_{DDON}$ | Current Consumption | $R_S$ < 50KΩ(1) | $C_{crystal}$ = 12.5 pF | | 650 | 1400 | nA |
| | | | $C_{crystal}$ = 6 pF | | 450 | 1200 | |
| | | $R_S$ < 100KΩ(1) | $C_{crystal}$ = 12.5 pF | | 900 | 1600 | |
| | | | $C_{crystal}$ = 6 pF | | 650 | 1400 | |
| $P_{ON}$ | Drive Level | | | | | 0.1 | µW |
| $R_f$ | Internal Resistor | Between XIN32 and XOUT32 | | | 10 | | MΩ |
| $C_{crystal}$ | Allowed Crystal Capacitance Load | From crystal specification | | 6 | | 12.5 | pF |
| $C_{para}$ | Internal Parasitic Capacitance | | | 0.8 | 1 | 1.2 | pF |

Note: 1. $R_S$ is the series resistor.

**Figure 36-11. 32.768 kHz Crystal Oscillator Schematics**




$C_{LEXT} = 2 \times (C_{crystal} - C_{para} - C_{PCB})$

where:

$C_{PCB}$ is the capacitance of the printed circuit board (PCB) track layout from the crystal to the SAM3 pin.

### 36.4.4 32.768 kHz Crystal Characteristics

**Table 36-22. Crystal Characteristics**

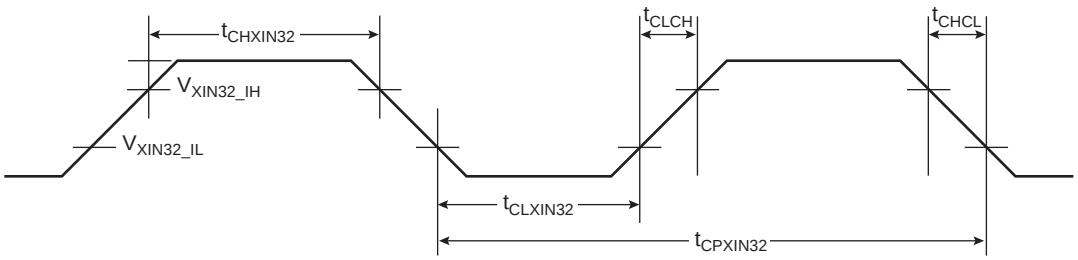
| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| ESR | Equivalent Series Resistor ($R_S$) | Crystal @ 32.768 kHz | | 50 | 100 | kΩ |
| $C_m$ | Motional Capacitance | | 0.6 | | 3 | fF |
| $C_{SHUNT}$ | Shunt Capacitance | | 0.6 | | 2 | pF |

### 36.4.5 32.768 kHz XIN32 Clock Input Characteristics in Bypass Mode

**Table 36-23. XIN32 Clock Electrical Characteristics (In Bypass Mode)**

| Symbol | Parameter | Conditions | Min | Max | Unit |
|---|---|---|---|---|---|
| $1/(t_{CPXIN32})$ | XIN32 Clock Frequency | 32.768 kHz crystal oscillator is in Bypass mode:<br>SUPC_MR.OSCBYPASS = 1<br>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 |
| $C_{IN}$ | XIN32 Input Capacitance | – | | 6 | pF |
| $R_{IN}$ | XIN32 Pull-down Resistor | – | 3 | 5 | MΩ |
| $V_{XIN32\_IL}$ | $V_{XIN32}$ Input Low-level Voltage | – | -0.3 | $0.3 \times V_{DDIO}$ | V |
| $V_{XIN32\_IH}$ | $V_{XIN32}$ Input High-level Voltage | – | $0.7 \times V_{DDIO}$ | $V_{DDIO} + 0.3$ | V |

**Figure 36-12. XIN32 Clock Timing**

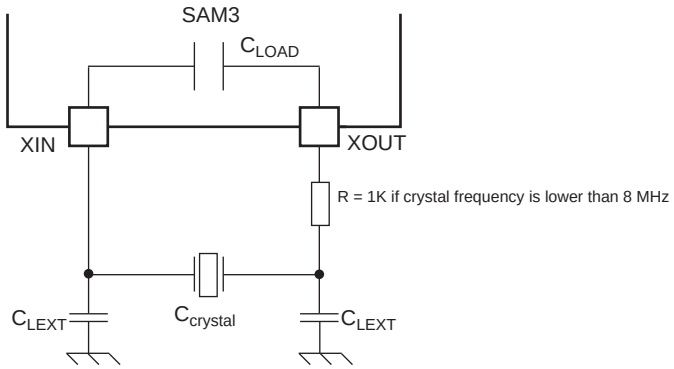


### 36.4.6 3 to 20 MHz Crystal Oscillator Characteristics

**Table 36-24. 3 to 20 MHz Crystal Oscillator Characteristics**

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| $f_{OSC}$ | Operating Frequency | Normal mode with crystal | 3 | 16 | 20 | MHz |
| $f_{OSC(bypass)}$ | Operating Frequency In Bypass Mode | External Clock on XIN | | | 50 | MHz |
| $V_{rip(VDDPLL)}$ | Supply Ripple Voltage (on VDDPLL) | RMS value, 10 kHz to 10 MHz | | | 30 | mV |
| | Duty Cycle | | 40 | 50 | 60 | % |
| $t_{START}$ | Startup Time | 3 MHz, $C_{SHUNT}$ = 3 pF<br>8 MHz, $C_{SHUNT}$ = 7 pF<br>12 to 16 MHz, $C_{SHUNT}$ = 7 pF<br>20 MHz, $C_{SHUNT}$ = 7 pF | | | 14.5<br>4<br>1.4<br>1 | ms |
| $I_{DDON}$ | Current Consumption | 3 MHz<br>8 MHz<br>12 to 16 MHz<br>20 MHz | | 150<br>200<br>250<br>350 | 230<br>300<br>350<br>450 | µA |
| $P_{ON}$ | Drive Level | 3 MHz | | | 15 | µW |
| | | 8 MHz | | | 30 | |
| | | 12 MHz, 16 MHz, 20 MHz | | | 50 | |
| $R_f$ | Internal Resistor | Between XIN and XOUT | | 1 | | MΩ |
| $C_{crystal}$ | Allowed Crystal Capacitance Load | From crystal specification | 12.5 | | 17.5 | pF |
| $C_{LOAD}$ | Internal Equivalent Load Capacitance | Integrated Load Capacitance (XIN and XOUT in series) | 7.5 | 9.5 | 11.5 | pF |

**Figure 36-13. 3 to 20 MHz Crystal Oscillator Schematic**




$C_{LEXT} = 2 \times (C_{crystal} - C_{LOAD} - C_{PCB})$

where:

$C_{PCB}$ is the capacitance of the printed circuit board (PCB) track layout from the crystal to the SAM3 pin.

### 36.4.7 3 to 20 MHz Crystal Characteristics

**Table 36-25. Crystal Characteristics**

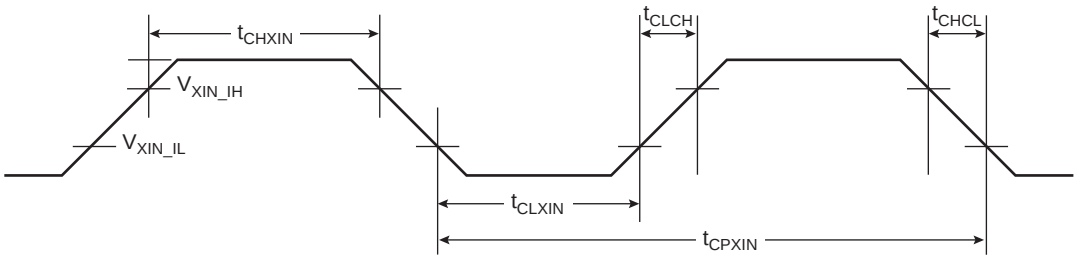
| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| ESR | Equivalent Series Resistor ($R_s$) | Fundamental @ 3 MHz<br>Fundamental @ 8 MHz<br>Fundamental @ 12 MHz<br>Fundamental @ 16 MHz<br>Fundamental @ 20 MHz | | | 200<br>100<br>80<br>80<br>50 | Ω |
| $C_m$ | Motional capacitance | | | | 8 | fF |
| $C_{SHUNT}$ | Shunt capacitance | | | | 7 | pF |

### 36.4.8 3 to 20 MHz XIN Clock Input Characteristics in Bypass Mode

**Table 36-26. 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 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 |
| $V_{XIN\_IL}$ | $V_{XIN}$ Input Low-level Voltage | | -0.3 | | $0.3 \times V_{DDIO}$ | V |
| $V_{XIN\_IH}$ | $V_{XIN}$ Input High-level Voltage | | $0.7 \times V_{DDIO}$ | | $V_{DDIO}$ + 0.3 | V |

**Figure 36-14. XIN Clock Timing**



### 36.4.9 Crystal Oscillators Design Consideration Information

SAM3N oscillators are low-power oscillators requiring particular attention when designing PCB systems.

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 SAM3N specifications are as follows:

- Load Capacitance

  $C_{crystal}$ is the equivalent capacitor value the oscillator must “show” to the crystal in order to oscillate at the target frequency. The crystal must be chosen according to the internal load capacitance ($C_{LOAD}$) of the on-chip oscillator. Having a mismatch for the load capacitance will result in a frequency drift.

- Drive Level

  Crystal Drive Level ≥ Oscillator Drive Level. Having a crystal drive level number lower than the oscillator specification may damage the crystal.

- Equivalent Series Resistor (ESR)

  Crystal ESR ≤ Oscillator ESR Max. Having a crystal with ESR value higher than the oscillator may cause the oscillator to not start.

- Shunt Capacitance

  Max. Crystal Shunt Capacitance ≤ Oscillator Shunt Capacitance ($C_{SHUNT}$). Having a crystal with ESR value higher than the oscillator may cause the oscillator to not start.

## 36.5 PLL Characteristics

**Table 36-27. PLL Characteristics**

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| $V_{DDPLL}$ | Supply Voltage Range | | 1.6 | 1.8 | 1.95 | V |
| $V_{rip(VDDPLL)}$ | Allowable Voltage Ripple | RMS value 10 kHz to 10 MHz | | | 30 | mV |
| | | RMS value > 10 MHz | | | 10 | |
| $f_{IN}$ | Input Frequency | | 3.5 | | 20 | MHz |
| $f_{OUT}$ | Output Frequency | | 60 | | 130 | MHz |
| $I_{PLL}$ | Current Consumption | Active mode @ 60 MHz @ 1.8V | | 1.2 | 1.7 | mA |
| | | Active mode @ 96 MHz @ 1.8V | | 2 | 2.5 | |
| | | Active mode @ 130 MHz @ 1.8V | | 2.5 | 3 | |
| $t_s$ | Settling Time | | | | 150 | µs |

## 36.6 10-bit ADC Characteristics

**Table 36-28. Analog Power Supply Characteristics**

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| $V_{DDIN}$ | ADC Analog Supply | | 3.0 | | 3.6 | V |
| $I_{VDDIN}$ | Active Current Consumption | On VDDIN | | 0.55 | 1 | mA |

**Table 36-29. Channel Conversion Time and ADC Clock**

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| $f_{ADC}$ | ADC Clock Frequency | 10-bit resolution mode | | | 5 | MHz |
| | | 8-bit resolution mode | | | 8 | |
| $t_{START}$ | Startup Time | Return from Idle Mode | | | 20 | µs |
| $t_{TRACK}$ | Track and Hold Acquisition Time | See Section 36.6.1 “Track and Hold Time versus Source Output Impedance” for more details | 600 | | | ns |
| $t_{CONV}$ | Conversion Time | ADC Clock = 5 MHz | | | 2 | µs |
| | | ADC Clock = 8 MHz | | | 1.25 | |
| | Throughput Rate | ADC Clock = 5 MHz | | | 384(1) | ksps |
| | | ADC Clock = 8 MHz | | | 533(2) | |

Notes:
1. Corresponds to 13 clock cycles at 5 MHz: 3 clock cycles for track and hold acquisition time and 10 clock cycles for conversion.
2. Corresponds to 15 clock cycles at 8 MHz: 5 clock cycles for track and hold acquisition time and 10 clock cycles for conversion.

**Table 36-30. External Voltage Reference Input**

| Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|
| ADVREF Input Voltage Range | 10-bit resolution mode | 2.6 | | $V_{DDIN}$ | V |
| | 8-bit resolution mode | 2.5 | | $V_{DDIN}$ | |
| ADVREF Average Current | On 13 samples with ADC Clock = 5 MHz | | 200 | 250 | µA |

**Table 36-31. Transfer Characteristics**

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| | Resolution | | | 10 | | bit |
| INL | Integral Non-linearity | | | | ±2 | LSB |
| DNL | Differential Non-linearity | No missing code | | | ±1 | |
| $E_O$ | Offset Error | | | | ±2 | |
| $E_G$ | Gain Error | | | | ±2 | |
| | Absolute Accuracy | | | | ±4 | |

**Table 36-32. Analog Inputs**

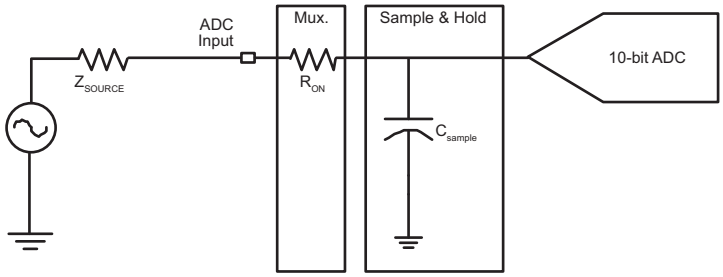
| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| $V_{IR}$ | Input Voltage Range | | 0 | | $V_{ADVREF}$ | V |
| $I_{lkg}$ | Input Leakage Current | | | | ±1 | µA |
| $C_i$ | Input Capacitance | | | 12 | 14 | pF |

Note: 1. Input Voltage range can be up to VDDIN without destruction or over-consumption. If VDDIO < $V_{ADVREF}$, max input voltage is VDDIO.

### 36.6.1 Track and Hold Time versus Source Output Impedance

The following figure gives a simplified acquisition path.

**Figure 36-15. Simplified Acquisition Path**




The user can drive ADC input with impedance of $Z_{SOURCE}$ up to:

- In 8-bit mode: $t_{TRACK} = 0.1 \times Z_{SOURCE} + 470$
- In 10-bit mode: $t_{TRACK} = 0.13 \times Z_{SOURCE} + 589$

with $t_{TRACK}$ (Track and Hold Time register) expressed in ns and $Z_{SOURCE}$ expressed in ohms.

Note: $C_{sample}$ and $R_{ON}$ are taken into account in the formulas

### 36.6.2 ADC Application Information

For more information on data converter terminology, please refer to the application note *Data Converter Terminology*, Atmel literature No. 6022, available on www.atmel.com.

## 36.7 10-bit DAC Characteristics

**Table 36-33. Analog Power Supply Characteristics**

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| $V_{DDIN}$ | Analog Supply | | 2.4 | | 3.6 | V |
| $I_{VDDIN}$ | Active Current Consumption | On VDDIN | | 485 | 660 | µA |
| | | On ADVREF | | 250 | 300 | |

**Table 36-34. Channel Conversion Time and DAC Clock**

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| $f_{DAC}$ | Clock Frequency | | | | 500 | kHz |
| $f_{START}$ | Startup Time | | | | 5 | µs |
| $t_{CONV}$ | Conversion Time | | | 1 | | $t_{CP\_DAC}$ |

External voltage reference for DAC is ADVREF. See the ADC voltage reference characteristics in Table 36-30 on page 729.

**Table 36-35. Static Performance Characteristics**

| Symbol | Parameter | Conditions | | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|---|
| | Resolution | | | | 10 | | bit |
| INL | Integral Non-linearity | Voltage output range between 0V and ($V_{ADVREF}$ - 100 mV) | 3.0V < $V_{DDIN}$ < 3.6V | | ±1 | ±2 | LSB |
| | | | 2.4V < $V_{DDIN}$ < 3.6V | | ±1 | ±3 | |
| DNL | Differential Non-linearity | | 3.0V < $V_{DDIN}$ < 3.6V | | ±0.5 | -0.9/+1 | LSB |
| | | | 2.4V < $V_{DDIN}$ < 3.6V | | ±0.5 | -3/+2 | |
| $E_O$ | Offset Error | | | | 1 | 5 | mV |
| $E_G$ | Gain Error | | | | 5 | 10 | mV |

**Table 36-36. Analog Outputs**

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| $V_{OR}$ | Voltage Output Range | | 0 | | $V_{ADVREF}$ | V |
| $t_s$ | Settling/Setup Time | $R_{LOAD}$ = 5kΩ/0pF < $C_{LOAD}$< 50pF | | | 2 | µs |
| $R_{LOAD}$ | Output Load Resistor | Output Load Resistor | 5 | | | kΩ |
| $C_{LOAD}$ | Output Load Capacitor | Output Load Capacitor | | | 50 | pF |

## 36.8 AC Characteristics

### 36.8.1 Master Clock Characteristics

**Table 36-37. Master Clock Waveform Parameters**

| Symbol | Parameter | Conditions | Min | Max | Unit |
|---|---|---|---|---|---|
| $1/(t_{CPMCK})$ | Master Clock Frequency | VDDCORE @ 1.62V | | 48 | MHz |
| | | VDDCORE @ 1.80V | | 62 | |

### 36.8.2 I/O Characteristics

Criteria used to define the maximum frequency of the I/Os:

- Output duty cycle (40%–60%)
- Minimum output swing: 100 mV to VDDIO - 100 mV
- Minimum output swing: 100 mV to VDDIO - 100 mV
- Addition of rising and falling time inferior to 75% of the period

**Table 36-38. I/O Characteristics**

| Symbol | Parameter | Conditions | | Min | Max | Unit |
|---|---|---|---|---|---|---|
| FreqMax1 | Pin Group 1 (1) Maximum Output Frequency | 30 pF | $V_{DDIO}$ = 1.62V | | 45 | MHz |
| | | | $V_{DDIO}$ = 3.0V | | 62 | |
| | | 45 pF | $V_{DDIO}$ = 1.62V | | 34 | |
| | | | $V_{DDIO}$ = 3.0V | | 45 | |
| $PulseminH_1$ | Pin Group 1 (1) High Level Pulse Width | 30 pF | $V_{DDIO}$ = 1.62V | 11 | | ns |
| | | | $V_{DDIO}$ = 3.0V | 7.7 | | |
| | | 45 pF | $V_{DDIO}$ = 1.8V | 14.7 | | |
| | | | $V_{DDIO}$ = 3.0V | 11 | | |
| $PulseminL_1$ | Pin Group 1 (1) Low Level Pulse Width | 30 pF | $V_{DDIO}$ = 1.62V | 11 | | ns |
| | | | $V_{DDIO}$ = 3.0V | 7.7 | | |
| | | 45 pF | $V_{DDIO}$ = 1.62V | 14.7 | | |
| | | | $V_{DDIO}$ = 3.0V | 11 | | |
| FreqMax2 | Pin Group 2 (2) Maximum Output Frequency | 25 pF | $1.62V < V_{DDIO} < 3.6V$ | | 35 | MHz |
| $PulseminH_2$ | Pin Group 2 (2) High Level Pulse Width | 25 pF | $1.62V < V_{DDIO} < 3.6V$ | 14.5 | | ns |
| $PulseminL_2$ | Pin Group 2 (2) Low Level Pulse Width | 25 pF | $1.62V < V_{DDIO} < 3.6V$ | 14.5 | | ns |

Notes: 1. Pin Group 1 = PA14

2. Pin Group 2 = PA[0–13], PA[15–31], PB[0–14], PC[0–31]

### 36.8.3 SPI Characteristics

**Figure 36-16. SPI Master Mode with (CPOL = NCPHA = 0) or (CPOL = NCPHA = 1)**

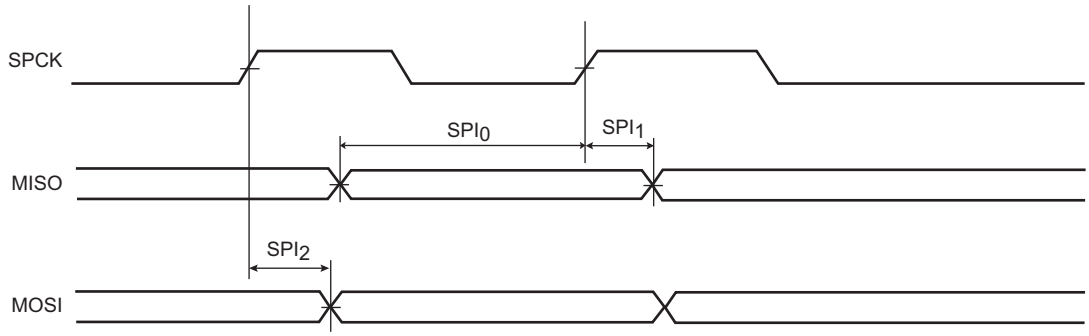



**Figure 36-17. SPI Master Mode with (CPOL = 0 and NCPHA = 1) or (CPOL = 1 and NCPHA = 0)**

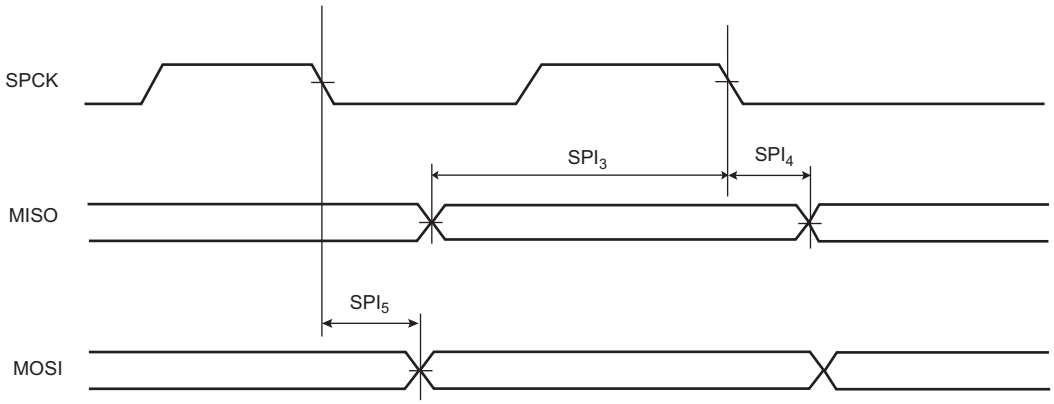



**Figure 36-18. SPI Slave Mode with (CPOL = 0 and NCPHA = 1) or (CPOL = 1 and NCPHA = 0)**

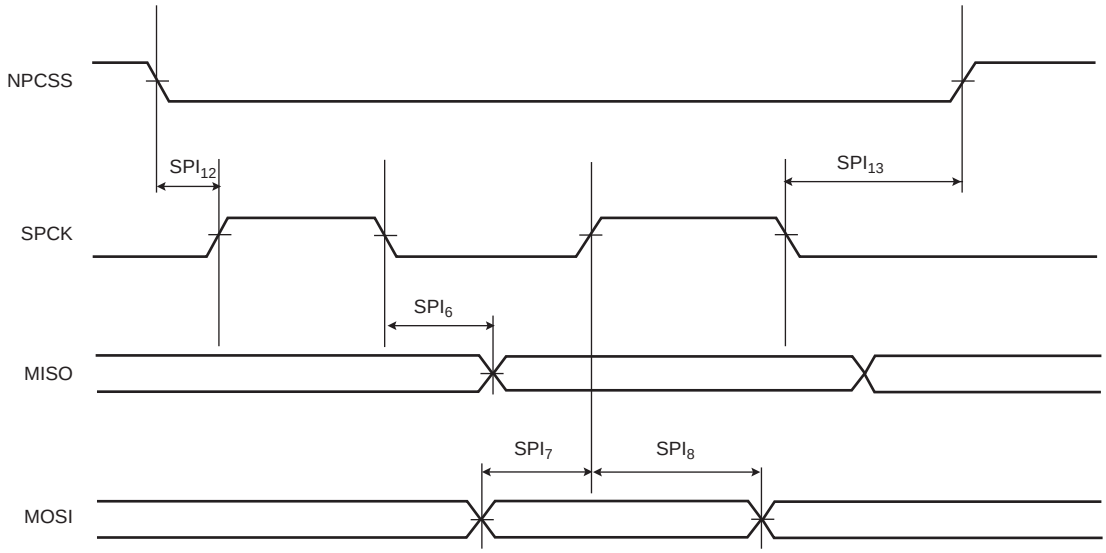
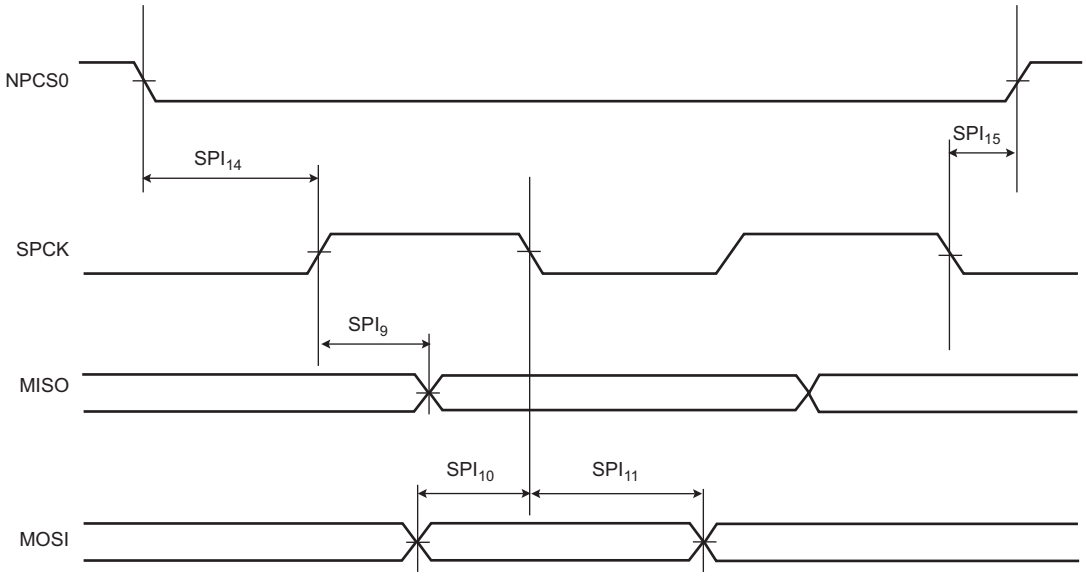


**Figure 36-19. SPI Slave Mode with (CPOL = NCPHA = 0) or (CPOL = NCPHA = 1)**




### 36.8.3.1 Maximum SPI Frequency

The following formulas give maximum SPI frequency in Master read and write modes and in Slave read and write modes.

#### Master Write Mode

The SPI is only sending data to a slave device such as an LCD, for example. The limit is given by $SPI_2$ (or $SPI_5$) timing. Since it gives a maximum frequency above the maximum pad speed (see Section 36.8.2 “I/O Characteristics”), the max SPI frequency is defined by the pin FreqMax value.

#### Master Read Mode

$$f_{SPCK}Max = \frac{1}{SPI_0(orSPI_3) + t_{valid}}$$

$t_{valid}$ is the slave time response to output data after deleting an SPCK edge. For a non-volatile memory with $t_{valid}$ (or $t_v$) = 12 ns, $f_{SPCK}$max = 38 MHz at $V_{DDIO}$ = 3.3V.

#### Slave Read Mode

In slave mode, SPCK is the input clock for the SPI. The max SPCK frequency is given by setup and hold timings $SPI_7$/$SPI_8$(or $SPI_{10}$/$SPI_{11}$). Since this gives a frequency well above the pad limit, the limit in slave read mode is given by SPCK pad.

#### Slave Write Mode

$$f_{SPCK}Max = \frac{1}{2x(SPI_6(orSPI_9) + t_{setup})}$$

For 3.3V I/O domain and SPI6, $f_{SPCK}$Max = 32 MHz. $t_{setup}$ is the setup time from the master before sampling data.

### 36.8.3.2 SPI Timings

SPI timings are given for the following domains:

- 3.3V domain: $V_{DDIO}$ from 3.0V to 3.6V, maximum external capacitor = 30 pF
- 1.8V domain: $V_{DDIO}$ from 1.65V to 1.95V, maximum external capacitor = 30 pF

**Table 36-39. SPI Timings**

| Symbol | Parameter | Conditions | Min | Max | Unit |
|---|---|---|---|---|---|
| $SPI_0$ | MISO Setup time before SPCK rises (master) | 3.3V Domain | 14.2 | | ns |
| | | 1.8V Domain | 17 | | |
| $SPI_1$ | MISO Hold time after SPCK rises (master) | 3.3V Domain | 0 | | |
| | | 1.8V Domain | 0 | | |
| $SPI_2$ | SPCK rising to MOSI Delay (master) | 3.3V Domain | -2.7 | 2.6 | |
| | | 1.8V Domain | -3.6 | 3.4 | |
| $SPI_3$ | MISO Setup time before SPCK falls (master) | 3.3V Domain | 14.4 | | |
| | | 1.8V Domain | 17 | | |
| $SPI_4$ | MISO Hold time after SPCK falls (master) | 3.3V Domain | 0 | | |
| | | 1.8V Domain | 0 | | |
| $SPI_5$ | SPCK falling to MOSI Delay (master) | 3.3V Domain | -2.4 | 2.8 | |
| | | 1.8V Domain | -3.4 | 3.6 | |
| $SPI_6$ | SPCK falling to MISO Delay (slave) | 3.3V Domain | 4.4 | 15.4 | |
| | | 1.8V Domain | 4.6 | 18.5 | |
| $SPI_7$ | MOSI Setup time before SPCK rises (slave) | 3.3V Domain | 0 | | |
| | | 1.8V Domain | 0 | | |
| $SPI_8$ | MOSI Hold time after SPCK rises (slave) | 3.3V Domain | 1.8 | | |
| | | 1.8V Domain | 1.6 | | |
| $SPI_9$ | SPCK rising to MISO Delay (slave) | 3.3V Domain | 4.9 | 15.4 | |
| | | 1.8V Domain | 5.2 | 18.3 | |
| $SPI_{10}$ | MOSI Setup time before SPCK falls (slave) | 3.3V Domain | 0 | | |
| | | 1.8V Domain | 0 | | |
| $SPI_{11}$ | MOSI Hold time after SPCK falls (slave) | 3.3V Domain | 1.9 | | |
| | | 1.8V Domain | 2 | | |
| $SPI_{12}$ | NPCS setup to SPCK rising (slave) | 3.3V Domain | 6.3 | | |
| | | 1.8V Domain | 6.4 | | |
| $SPI_{13}$ | NPCS hold after SPCK falling (slave) | 3.3V Domain | 0 | | |
| | | 1.8V Domain | 0 | | |
| $SPI_{14}$ | NPCS setup to SPCK falling (slave) | 3.3V Domain | 6.4 | | |
| | | 1.8V Domain | 6.4 | | |
| $SPI_{15}$ | NPCS hold after SPCK falling (slave) | 3.3V Domain | 0 | | |
| | | 1.8V Domain | 0 | | |

Note that in SPI master mode the SAM3N does not sample the data (MISO) on the opposite edge where data clocks out (MOSI) but the same edge is used as shown in Figure 36-16 and Figure 36-17.

### 36.8.4 USART in SPI Mode Timings

USART in SPI mode timings are given for the following domains:

- 1.8V domain: $V_{DDIO}$ from 1.65V to 1.95V, maximum external capacitor = 25pF
- 3.3V domain: $V_{DDIO}$ from 3.0V to 3.6V, maximum external capacitor = 25pF

Timings are given with the following conditions:

VDDIO = 1.62V and 3V

SCK/MISO/MOSI Load = 30 pF

**Figure 36-20. USART SPI Master Mode**

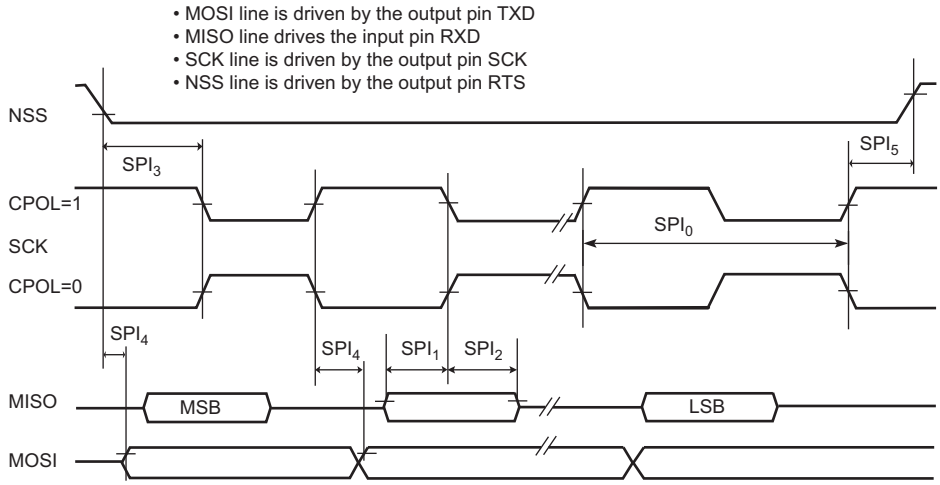



**Figure 36-21. USART SPI Slave mode: (Mode 1 or 2)**

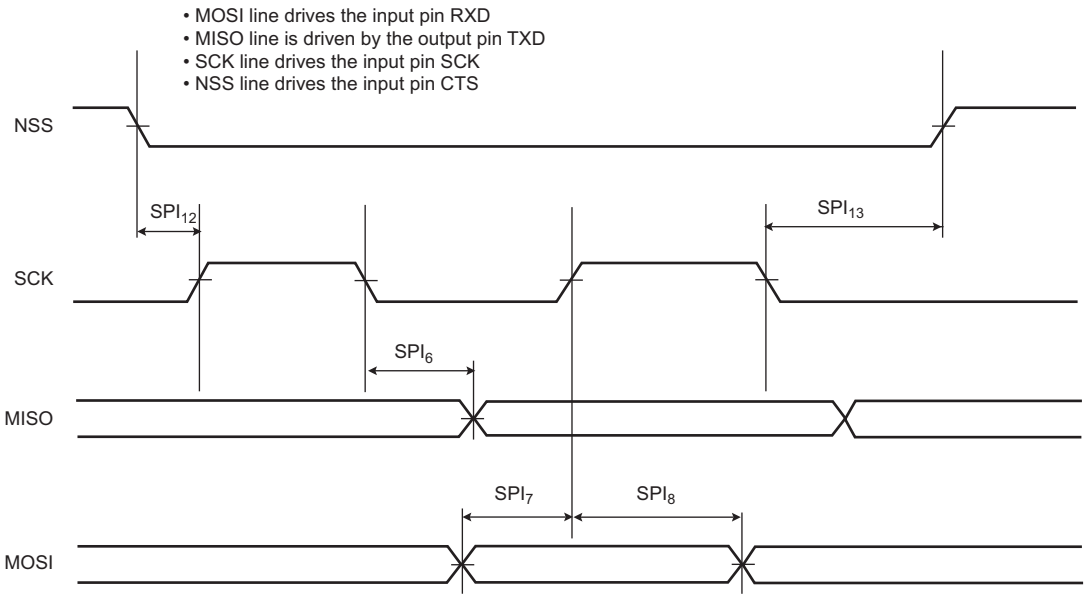


**Figure 36-22. USART SPI Slave mode: (Mode 0 or 3)**

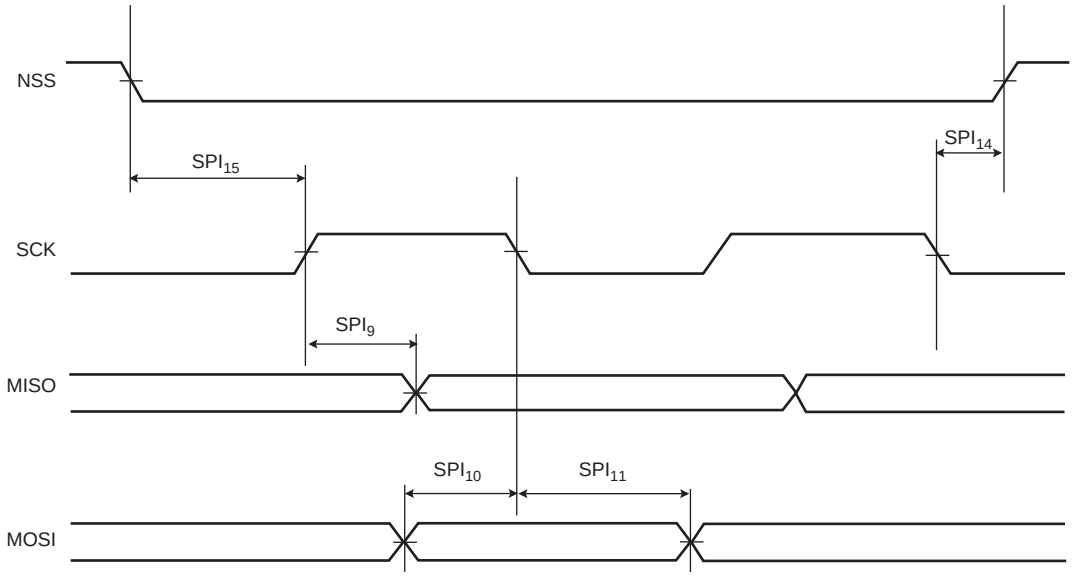


**Table 36-40. USART SPI Timings**

| Symbol | Parameter | Conditions | Min | Max | Unit |
|---|---|---|---|---|---|
| | | **Master Mode** | | | |
| $SPI_0$ | $t_{CPSCK}$ Period | 1.8V Domain<br>3.3V Domain | $t_{CPMCK}$ / 6 | | ns |
| $SPI_1$ | Input Data Setup Time | 1.8V Domain<br>3.3V Domain | $0.5 \times t_{CPMCK}$ + 2.6<br>$0.5 \times t_{CPMCK}$ + 2.4 | | ns |
| $SPI_2$ | Input Data Hold Time | 1.8V Domain<br>3.3V Domain | $1.5 \times t_{CPMCK}$ - 0.3<br>$1.5 \times t_{CPMCK}$ - 0.3 | | ns |
| $SPI_3$ | Chip Select Active to Serial Clock | 1.8V Domain<br>3.3V Domain | $1.5 \times t_{CPSCK}$ - 0.9<br>$1.5 \times t_{CPSCK}$ - 0.6 | | ns |
| $SPI_4$ | Output Data Setup Time | 1.8V Domain<br>3.3V Domain | -6<br>-4.7 | 3.8<br>3.6 | ns |
| $SPI_5$ | Serial Clock to Chip Select Inactive | 1.8V Domain<br>3.3V Domain | $1 \times t_{CPSCK}$ - 6<br>$1 \times t_{CPSCK}$ - 4.6 | | ns |
| | | **Slave Mode** | | | |
| $SPI_6$ | $t_{CPSCK}$ falling to MISO | 1.8V Domain<br>3.3V Domain | 5.7<br>5.3 | 22.6<br>19.8 | ns |
| $SPI_7$ | MOSI Setup time before $t_{CPSCK}$ rises | 1.8V Domain<br>3.3V Domain | $2 \times t_{CPMCK}$ + 1.9<br>$2 \times t_{CPMCK}$ + 1.7 | | ns |
| $SPI_8$ | MOSI Hold time after $t_{CPSCK}$ rises | 1.8V Domain<br>3.3V Domain | 0<br>0 | | ns |
| $SPI_9$ | $t_{CPSCK}$ rising to MISO | 1.8V Domain<br>3.3V Domain | 5.9<br>5.6 | 22<br>19.4 | ns |
| $SPI_{10}$ | MOSI Setup time before $t_{CPSCK}$ falls | 1.8V Domain<br>3.3V Domain | $2 \times t_{CPMCK}$ + 1.8<br>$2 \times t_{CPMCK}$ + 1.7 | | ns |
| $SPI_{11}$ | MOSI Hold time after $t_{CPSCK}$ falls | 1.8V Domain<br>3.3V Domain | 0.5<br>0.4 | | ns |
| $SPI_{12}$ | NPCS0 setup to $t_{CPSCK}$ rising | 1.8V Domain<br>3.3V Domain | $2.5 \times t_{CPMCK}$ -0.26<br>$2.5 \times t_{CPMCK}$ -0.4 | | ns |
| $SPI_{13}$ | NPCS0 hold after $t_{CPSCK}$ falling | 1.8V Domain<br>3.3V Domain | $1.5 \times t_{CPMCK}$ + 2.2<br>$1.5 \times t_{CPMCK}$ + 2 | | ns |
| $SPI_{14}$ | NPCS0 setup to $t_{CPSCK}$ falling | 1.8V Domain<br>3.3V Domain | $2.5 \times t_{CPMCK}$ -0.4<br>$2.5 \times t_{CPMCK}$ -0.4 | | ns |
| $SPI_{15}$ | NPCS0 hold after $t_{CPSCK}$ rising | 1.8V Domain<br>3.3V Domain | $1.5 \times t_{CPMCK}$ + 1.8<br>$1.5 \times t_{CPMCK}$ + 1.7 | | ns |

### 36.8.5 Two-wire Serial Interface Characteristics

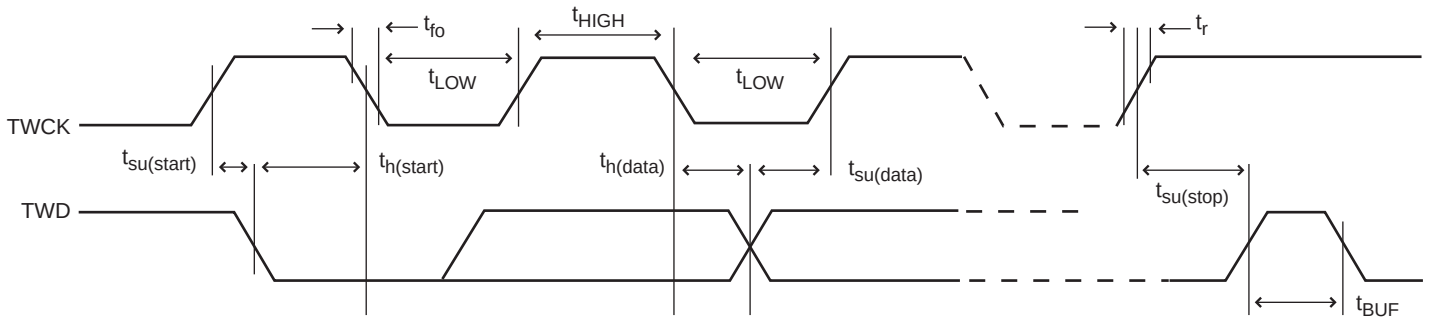
Table 36-41 describes the requirements for devices connected to the Two-wire Serial Bus. For timing symbols refer to Figure 36-23.

**Table 36-41. Two-wire Serial Bus Requirements**

| Symbol | Parameter | Conditions | Min | Max | Unit |
|---|---|---|---|---|---|
| $V_{IL}$ | Input Low-voltage | | -0.3 | $0.3 \times V_{DDIO}$ | V |
| $V_{IH}$ | Input High-voltage | | $0.7 \times V_{DDIO}$ | $V_{CC}$ + 0.3 | V |
| $V_{hys}$ | Hysteresis of Schmitt Trigger Inputs | | 0.150 | – | V |
| $V_{OL}$ | Output Low-voltage | 3 mA sink current | – | 0.4 | V |
| $t_r$ | Rise Time for both TWD and TWCK | | $20 + 0.1C_b$ (1)(2) | 300 | ns |
| $t_{fo}$ | Output Fall Time from $V_{IHmin}$ to $V_{ILmax}$ | 10 pF < $C_b$ < 400 pF Figure 36-23 | $20 + 0.1C_b$ (1)(2) | 250 | ns |
| $C_i$ (1) | Capacitance for each I/O Pin | | – | 10 | pF |
| $f_{TWCK}$ | TWCK Clock Frequency | | 0 | 400 | kHz |
| Rp | Value of Pull-up resistor | $f_{TWCK} \leq 100$ kHz | $(V_{DDIO}$ - 0.4V) ÷ 3mA | 1000ns ÷ $C_b$ | Ω |
| | | $f_{TWCK} > 100$ kHz | | 300ns ÷ $C_b$ | |
| $t_{LOW}$ | Low Period of the TWCK clock | $f_{TWCK} \leq 100$ kHz | (3) | – | µs |
| | | $f_{TWCK} > 100$ kHz | (3) | – | |
| $t_{HIGH}$ | High period of the TWCK clock | $f_{TWCK} \leq 100$ kHz | (4) | – | µs |
| | | $f_{TWCK} > 100$ kHz | (4) | – | |
| $t_{h(start)}$ | Hold Time (repeated) START condition | $f_{TWCK} \leq 100$ kHz | $t_{HIGH}$ | – | µs |
| | | $f_{TWCK} > 100$ kHz | $t_{HIGH}$ | – | |
| $t_{su(start)}$ | Set-up time for a repeated START condition | $f_{TWCK} \leq 100$ kHz | $t_{HIGH}$ | – | µs |
| | | $f_{TWCK} > 100$ kHz | $t_{HIGH}$ | – | |
| $t_{h(data)}$ | Data hold time | $f_{TWCK} \leq 100$ kHz | 0 | $3 \times t_{CPMCK}$ (5) | µs |
| | | $f_{TWCK} > 100$ kHz | 0 | $3 \times t_{CPMCK}$ (5) | |
| $t_{su(data)}$ | Data setup time | $f_{TWCK} \leq 100$ kHz | $t_{LOW} - 3 \times t_{CPMCK}$ (5) | – | ns |
| | | $f_{TWCK} > 100$ kHz | $t_{LOW} - 3 \times t_{CPMCK}$ (5) | – | |
| $t_{su(stop)}$ | Setup time for STOP condition | $f_{TWCK} \leq 100$ kHz | $t_{HIGH}$ | – | µs |
| | | $f_{TWCK} > 100$ kHz | $t_{HIGH}$ | – | |
| $t_{BUF}$ | Bus free time between a STOP and START condition | $f_{TWCK} \leq 100$ kHz | $t_{LOW}$ | – | µs |
| | | $f_{TWCK} > 100$ kHz | $t_{LOW}$ | – | |

Note:
1. Required only for $f_{TWCK} > 100$ kHz.
2. $C_b$ = capacitance of one bus line in pF. Per I2C Standard, $C_b$ Max = 400pF
3. The TWCK low Period is defined as follows: $t_{LOW} = ((CLDIV \times 2^{CKDIV}) + 4) \times t_{MCK}$
4. The TWCK high period is defined as follows: $t_{HIGH} = ((CHDIV \times 2^{CKDIV}) + 4) \times t_{MCK}$
5. $t_{CPMCK}$ = MCK bus period

**Figure 36-23. Two-wire Serial Bus Timing**




### 36.8.6 Embedded Flash Characteristics

The maximum operating frequency given in Table 36-42 is limited by the Embedded Flash access time when the processor is fetching code out of it. The table provides the device maximum operating frequency defined by the value of field FWS in the EEFC_FMR. This field defines the number of wait states required to access the Embedded Flash Memory.

Note: The embedded Flash is fully tested during production test. The Flash contents are not set to a known state prior to shipment. Therefore, the Flash contents should be erased prior to programming an application.

**Table 36-42. Embedded Flash Wait State - VDDCORE 1.65V/1.80V**

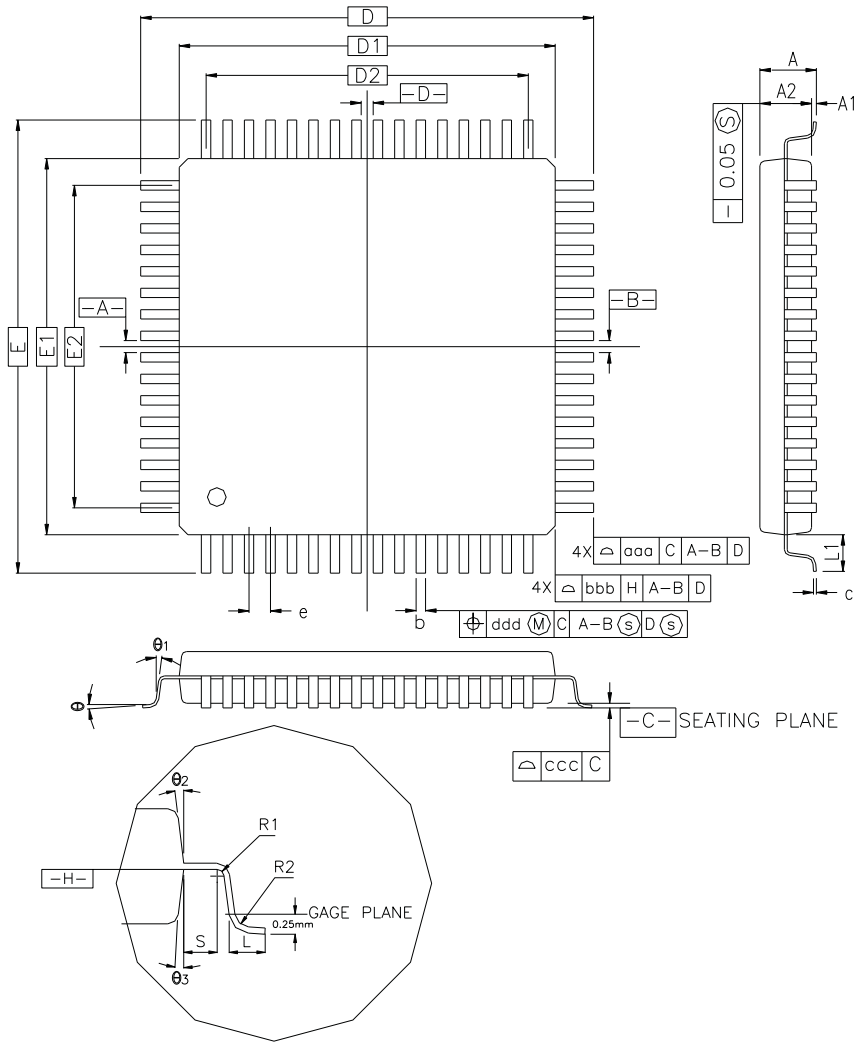
| EEFC_FMR.FWS | Read Operations | Maximum Operating Frequency (MHz) | |
|---|---|---|---|
| | | VDDCORE 1.62V | VDDCORE 1.80V |
| 0 | 1 cycle | 21 | 24 |
| 1 | 2 cycles | 32 | 42 |
| 2 | 3 cycles | 48 | 62 |

**Table 36-43. AC Flash Characteristics**

| Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|
| Program Cycle Time | Per page including auto-erase | | | 4.6 | ms |
| | Per page without auto-erase | | | 2.3 | |
| Full Chip Erase | | | 10 | 11.5 | s |
| Data Retention | Not Powered or Powered | 10 | | | years |
| Endurance | Write/Erase cycles @ 25°C | | 30K | | cycles |
| | Write/Erase cycles @ 85°C | 10K | | | |

# 37. Mechanical Characteristics

**Figure 37-1. 100-lead LQFP Package Mechanical Drawing**




CONTROL DIMENSIONS ARE IN MILLIMETERS.

| SYMBOL | MILLIMETER | | | INCH | | |
|---|---|---|---|---|---|---|
| | MIN. | NOM. | MAX. | MIN. | NOM. | MAX. |
| A | — | — | 1.60 | — | — | 0.063 |
| $A_1$ | 0.05 | — | 0.15 | 0.002 | — | 0.006 |
| $A_2$ | 1.35 | 1.40 | 1.45 | 0.053 | 0.055 | 0.057 |
| D | 16.00 BSC. | | | 0.630 BSC. | | |
| $D_1$ | 14.00 BSC. | | | 0.551 BSC. | | |
| E | 16.00 BSC. | | | 0.630 BSC. | | |
| $E_1$ | 14.00 BSC. | | | 0.551 BSC. | | |
| $R_2$ | 0.08 | — | 0.20 | 0.003 | — | 0.008 |
| $R_1$ | 0.08 | — | — | 0.003 | — | — |
| θ | 0° | 3.5° | 7° | 0° | 3.5° | 7° |
| $θ_1$ | 0° | — | — | 0° | — | — |
| $θ_2$ | 11° | 12° | 13° | 11° | 12° | 13° |
| $θ_3$ | 11° | 12° | 13° | 11° | 12° | 13° |
| c | 0.09 | — | 0.20 | 0.004 | — | 0.008 |
| L | 0.45 | 0.60 | 0.75 | 0.018 | 0.024 | 0.030 |
| $L_1$ | 1.00 REF | | | 0.039 REF | | |
| S | 0.20 | — | — | 0.008 | — | — |
| b | 0.17 | 0.20 | 0.27 | 0.007 | 0.008 | 0.011 |
| e | 0.50 BSC. | | | 0.020 BSC. | | |
| D2 | 12.00 | | | 0.472 | | |
| E2 | 12.00 | | | 0.472 | | |
| TOLERANCES OF FORM AND POSITION | | | | | | |
| aaa | 0.20 | | | 0.008 | | |
| bbb | 0.20 | | | 0.008 | | |
| ccc | 0.08 | | | 0.003 | | |
| ddd | 0.08 | | | 0.003 | | |

Note: 1. This drawing is for general information only. Refer to JEDEC Drawing MS-026 for additional information.

**Table 37-1. Device and 100-lead LQFP Package Maximum Weight**

| | | |
|---|---|---|
| SAM3N4/2/1 | 800 | mg |

**Table 37-2. 100-lead LQFP Package Reference**

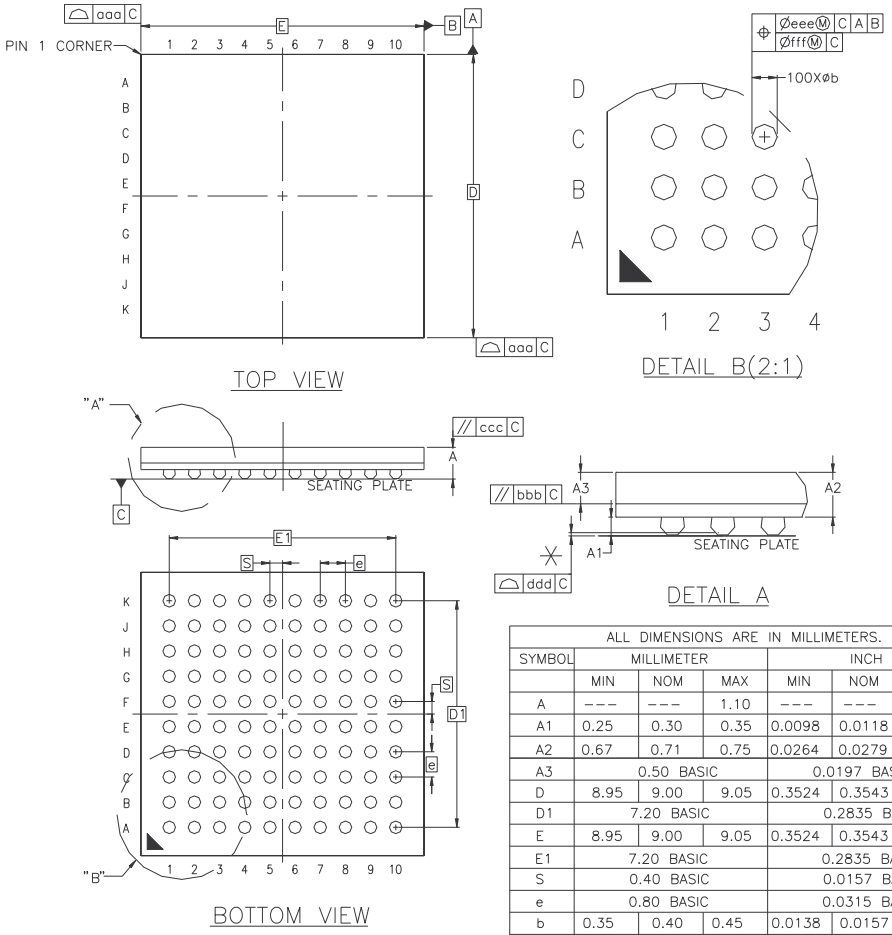
| | |
|---|---|
| JEDEC Drawing Reference | MS-026 |
| JESD97 Classification | e3 |

**Table 37-3. 100-lead LQFP Package Characteristics**

| | |
|---|---|
| Moisture Sensitivity Level | 3 |

This package respects the recommendations of the NEMI User Group.

**Figure 37-2. 100-ball TFBGA Package Drawing**




| ALL DIMENSIONS ARE IN MILLIMETERS. | | | | | | |
|---|---|---|---|---|---|---|
| SYMBOL | MILLIMETER | | | INCH | | |
| | MIN | NOM | MAX | MIN | NOM | MAX |
| A | --- | --- | 1.10 | --- | --- | 0.0575 |
| A1 | 0.25 | 0.30 | 0.35 | 0.0098 | 0.0118 | 0.0138 |
| A2 | 0.67 | 0.71 | 0.75 | 0.0264 | 0.0279 | 0.0295 |
| A3 | 0.50 BASIC | | | 0.0197 BASIC | | |
| D | 8.95 | 9.00 | 9.05 | 0.3524 | 0.3543 | 0.3563 |
| D1 | 7.20 BASIC | | | 0.2835 BASIC | | |
| E | 8.95 | 9.00 | 9.05 | 0.3524 | 0.3543 | 0.3563 |
| E1 | 7.20 BASIC | | | 0.2835 BASIC | | |
| S | 0.40 BASIC | | | 0.0157 BASIC | | |
| e | 0.80 BASIC | | | 0.0315 BASIC | | |
| b | 0.35 | 0.40 | 0.45 | 0.0138 | 0.0157 | 0.0177 |
| aaa | 0.15 | | | 0.0059 | | |
| bbb | 0.20 | | | 0.0079 | | |
| ccc | 0.20 | | | 0.0079 | | |
| ddd | 0.12 | | | 0.0047 | | |
| eee | 0.15 | | | 0.0059 | | |
| fff | 0.08 | | | 0.0031 | | |

**Table 37-4. 100-ball TFBGA Soldering Information (Substrate Level)**

| | |
|---|---|
| Ball Land | 0.35 mm |
| Soldering Mask Opening | 0.35 mm |

**Table 37-5. 100-ball TFBGA Device Maximum Weight**

| | |
|---|---|
| 150 | mg |

**Table 37-6. 100-ball TFBGA Package Characteristics**

| | |
|---|---|
| Moisture Sensitivity Level | 3 |

**Table 37-7. 100-ball TFBGA Package Reference**

| | |
|---|---|
| JEDEC Drawing Reference | MO-275-DDAC-01 |
| JESD97 Classification | e8 |

**Figure 37-3. 64- and 48-lead LQFP Package Drawing**

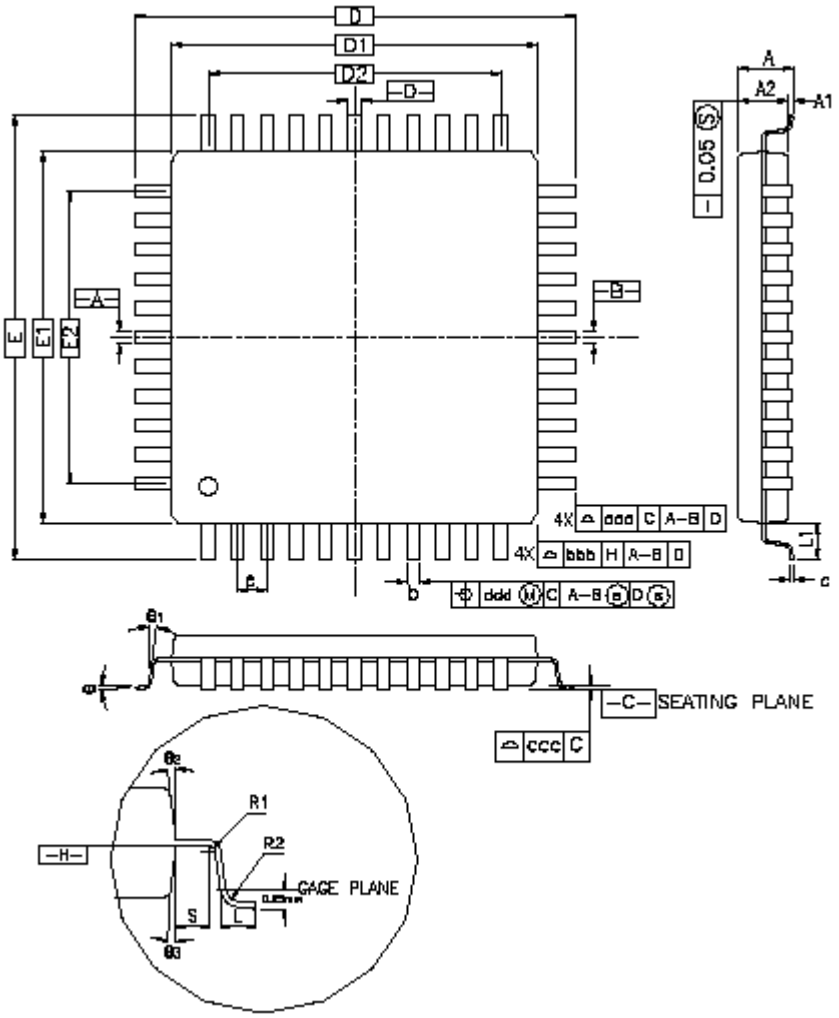


**Table 37-8. 48-lead LQFP Package Dimensions (in mm)**

| 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 | 9.00 BSC | | | 0.354 BSC | | |
| D1 | 7.00 BSC | | | 0.276 BSC | | |
| E | 9.00 BSC | | | 0.354 BSC | | |
| E1 | 7.00 BSC | | | 0.276 BSC | | |
| R2 | 0.08 | – | 0.20 | 0.003 | – | 0.008 |
| R1 | 0.08 | – | – | 0.003 | – | – |
| q | 0° | 3.5° | 7° | 0° | 3.5° | 7° |
| $\theta_1$ | 0° | – | – | 0° | – | – |
| $\theta_2$ | 11° | 12° | 13° | 11° | 12° | 13° |
| $\theta_3$ | 11° | 12° | 13° | 11° | 12° | 13° |
| c | 0.09 | – | 0.20 | 0.004 | – | 0.008 |
| L | 0.45 | 0.60 | 0.75 | 0.018 | 0.024 | 0.030 |
| L1 | 1.00 REF | | | 0.039 REF | | |
| S | 0.20 | – | – | 0.008 | – | – |
| b | 0.17 | 0.20 | 0.27 | 0.007 | 0.008 | 0.011 |
| e | 0.50 BSC. | | | 0.020 BSC. | | |
| D2 | 5.50 | | | 0.217 | | |
| E2 | 5.50 | | | 0.217 | | |
| Tolerances of Form and Position | | | | | | |
| aaa | 0.20 | | | 0.008 | | |
| bbb | 0.20 | | | 0.008 | | |
| ccc | 0.08 | | | 0.003 | | |
| ddd | 0.08 | | | 0.003 | | |

**Table 37-9. 64-lead LQFP Package Dimensions (in mm)**

| 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 | | | 0.383 BSC | | |
| E | 12.00 BSC | | | 0.472 BSC | | |
| E1 | 10.00 BSC | | | 0.383 BSC | | |
| R2 | 0.08 | – | 0.20 | 0.003 | – | 0.008 |
| R1 | 0.08 | – | – | 0.003 | – | – |
| q | 0° | 3.5° | 7° | 0° | 3.5° | 7° |
| $\theta_1$ | 0° | – | – | 0° | – | – |
| $\theta_2$ | 11° | 12° | 13° | 11° | 12° | 13° |
| $\theta_3$ | 11° | 12° | 13° | 11° | 12° | 13° |
| c | 0.09 | – | 0.20 | 0.004 | – | 0.008 |
| L | 0.45 | 0.60 | 0.75 | 0.018 | 0.024 | 0.030 |
| L1 | 1.00 REF | | | 0.039 REF | | |
| S | 0.20 | – | – | 0.008 | – | – |
| b | 0.17 | 0.20 | 0.27 | 0.007 | 0.008 | 0.011 |
| e | 0.50 BSC. | | | 0.020 BSC. | | |
| D2 | 7.50 | | | 0.285 | | |
| E2 | 7.50 | | | 0.285 | | |
| Tolerances of Form and Position | | | | | | |
| aaa | 0.20 | | | 0.008 | | |
| bbb | 0.20 | | | 0.008 | | |
| ccc | 0.08 | | | 0.003 | | |
| ddd | 0.08 | | | 0.003 | | |

**Table 37-10. Device and LQFP Package Maximum Weight**

| | | |
|---|---|---|
| SAM3N4/2/1 | 750 | mg |

**Table 37-11. LQFP Package Reference**

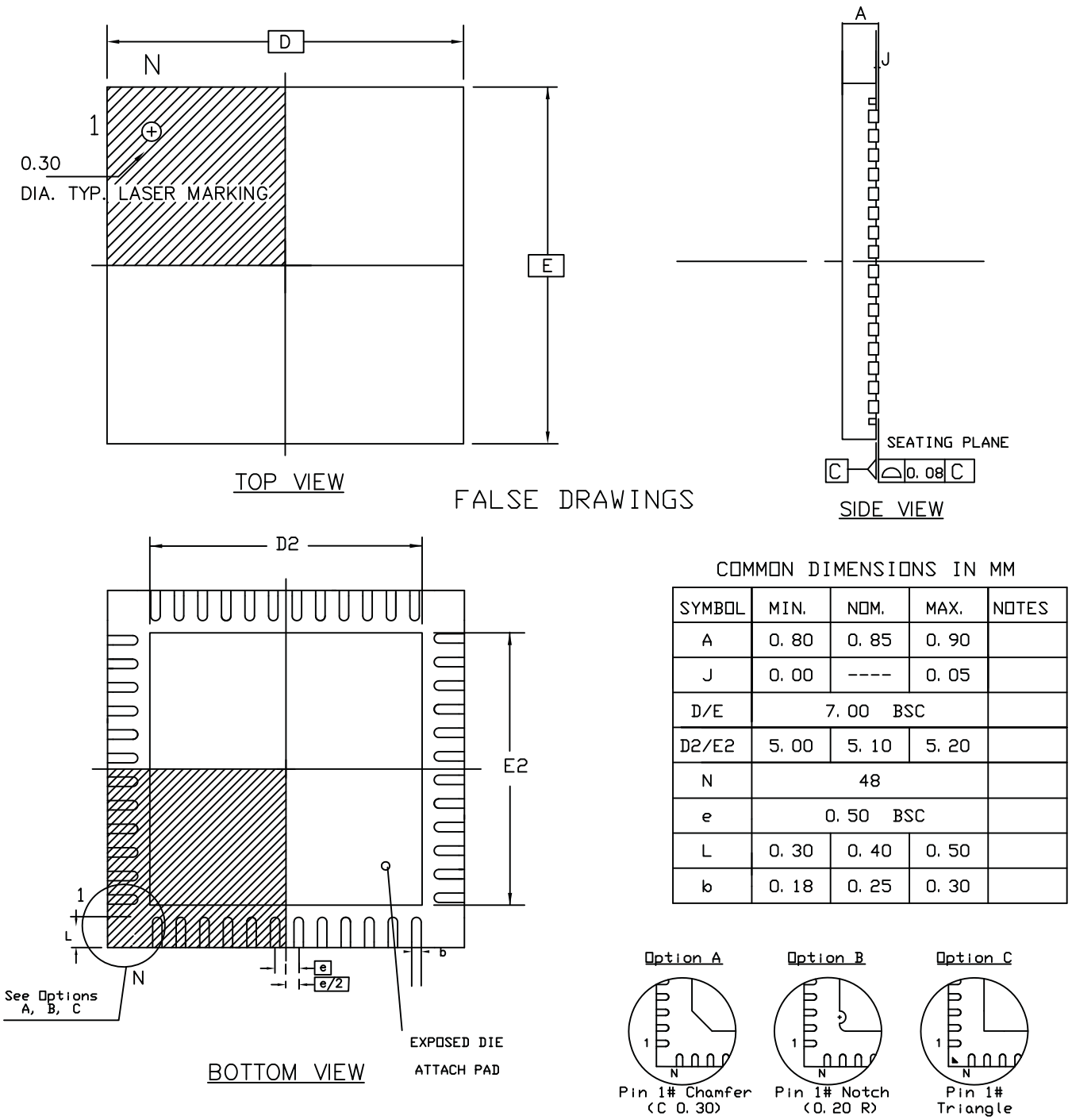
| | |
|---|---|
| JEDEC Drawing Reference | MS-026 |
| JESD97 Classification | e3 |

**Table 37-12. LQFP and QFN Package Characteristics**

| | |
|---|---|
| Moisture Sensitivity Level | 3 |

This package respects the recommendations of the NEMI User Group.

**Figure 37-4. 48-pad QFN Package**



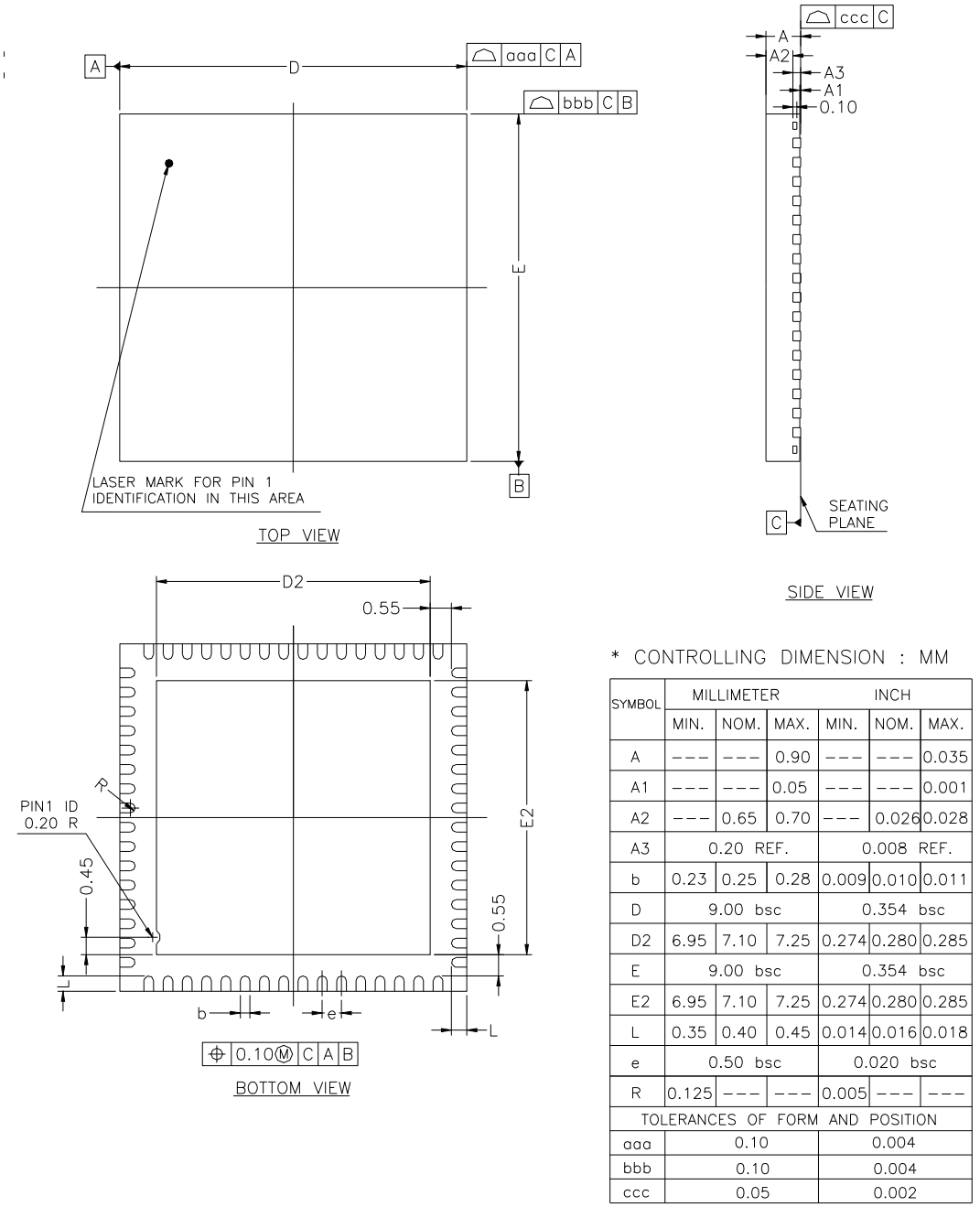

COMMON DIMENSIONS IN MM

| SYMBOL | MIN. | NOM. | MAX. | NOTES |
|---|---|---|---|---|
| A | 0.80 | 0.85 | 0.90 | |
| J | 0.00 | ---- | 0.05 | |
| D/E | 7.00 BSC | | | |
| D2/E2 | 5.00 | 5.10 | 5.20 | |
| N | 48 | | | |
| e | 0.50 BSC | | | |
| L | 0.30 | 0.40 | 0.50 | |
| b | 0.18 | 0.25 | 0.30 | |

**Table 37-13. 48-pad QFN Package Dimensions (in mm)**

| Symbol | Millimeter | | | Inch | | |
|---|---|---|---|---|---|---|
| | Min | Nom | Max | Min | Nom | Max |
| A | – | – | 090 | – | – | 0.035 |
| A1 | – | – | 0.050 | – | – | 0.002 |
| A2 | – | 0.65 | 0.70 | – | 0.026 | 0.028 |
| A3 | | 0.20 REF | | | 0.008 REF | |
| b | 0.18 | 0.20 | 0.23 | 0.007 | 0.008 | 0.009 |
| D | | 7.00 bsc | | | 0.276 bsc | |
| D2 | 5.45 | 5.60 | 5.75 | 0.215 | 0.220 | 0.226 |
| E | | 7.00 bsc | | | 0.276 bsc | |
| E2 | 5.45 | 5.60 | 5.75 | 0.215 | 0.220 | 0.226 |
| L | 0.35 | 0.40 | 0.45 | 0.014 | 0.016 | 0.018 |
| e | | 0.50 bsc | | | 0.020 bsc | |
| R | 0.09 | – | – | 0.004 | – | – |
| Tolerances of Form and Position | | | | | | |
| aaa | | 0.10 | | | 0.004 | |
| bbb | | 0.10 | | | 0.004 | |
| ccc | | 0.05 | | | 0.002 | |

**Figure 37-5. 64-pad QFN Package Drawing**




* CONTROLLING DIMENSION : MM

| SYMBOL | MILLIMETER | | | INCH | | |
|---|---|---|---|---|---|---|
| | MIN. | NOM. | MAX. | MIN. | NOM. | MAX. |
| A | --- | --- | 0.90 | --- | --- | 0.035 |
| A1 | --- | --- | 0.05 | --- | --- | 0.001 |
| A2 | --- | 0.65 | 0.70 | --- | 0.026 | 0.028 |
| A3 | 0.20 REF. | | | 0.008 REF. | | |
| b | 0.23 | 0.25 | 0.28 | 0.009 | 0.010 | 0.011 |
| D | 9.00 bsc | | | 0.354 bsc | | |
| D2 | 6.95 | 7.10 | 7.25 | 0.274 | 0.280 | 0.285 |
| E | 9.00 bsc | | | 0.354 bsc | | |
| E2 | 6.95 | 7.10 | 7.25 | 0.274 | 0.280 | 0.285 |
| L | 0.35 | 0.40 | 0.45 | 0.014 | 0.016 | 0.018 |
| e | 0.50 bsc | | | 0.020 bsc | | |
| R | 0.125 | --- | --- | 0.005 | --- | --- |
| TOLERANCES OF FORM AND POSITION | | | | | | |
| aaa | 0.10 | | | 0.004 | | |
| bbb | 0.10 | | | 0.004 | | |
| ccc | 0.05 | | | 0.002 | | |

**Table 37-14. Device and QFN Package Maximum Weight (Preliminary)**

| | | |
|---|---|---|
| SAM3N4/2/1 | 280 | mg |

**Table 37-15. QFN Package Reference**

| | |
|---|---|
| JEDEC Drawing Reference | MO-220 |
| JESD97 Classification | e3 |

**Table 37-16. QFN Package Characteristics**

| | |
|---|---|
| Moisture Sensitivity Level | 3 |

This package respects the recommendations of the NEMI User Group.

## 37.1 Soldering Profile

Table 37-17 gives the recommended soldering profile from J-STD-020C.

**Table 37-17. 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.

## 37.2 Packaging Resources

Land Pattern Definition.

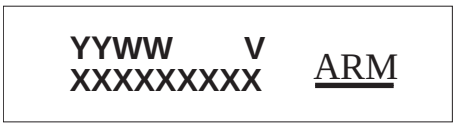
Refer to the following IPC Standards:

- IPC-7351A and IPC-782 (*Generic Requirements for Surface Mount Design and Land Pattern Standards*) http://landpatterns.ipc.org/default.asp
- Atmel Green and RoHS Policy and Package Material Declaration Data Sheet http://www.atmel.com/green/

# 38. 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

# 39. Ordering Information

**Table 39-1. SAM3N Ordering Information**

| Ordering Code | MRL | Flash (Kbytes) | Package | Operating Temperature Range |
|---|---|---|---|---|
| ATSAM3N4CA-AU | A | 256 | LQFP100 | Industrial -40°C to 85°C |
| ATSAM3N4CA-CU | A | 256 | TFBGA100 | Industrial -40°C to 85°C |
| ATSAM3N4BA-AU | A | 256 | LQFP64 | Industrial -40°C to 85°C |
| ATSAM3N4BA-MU | A | 256 | QFN64 | Industrial -40°C to 85°C |
| ATSAM3N4AA-AU | A | 256 | LQFP48 | Industrial -40°C to 85°C |
| ATSAM3N4AA-MU | A | 256 | QFN48 | Industrial -40°C to 85°C |
| ATSAM3N2CA-AU | A | 128 | LQFP100 | Industrial -40°C to 85°C |
| ATSAM3N2CA-CU | A | 128 | TFBGA100 | Industrial -40°C to 85°C |
| ATSAM3N2BA-AU | A | 128 | LQFP64 | Industrial -40°C to 85°C |
| ATSAM3N2BA-MU | A | 128 | QFN64 | Industrial -40°C to 85°C |
| ATSAM3N2AA-AU | A | 128 | LQFP48 | Industrial -40°C to 85°C |
| ATSAM3N2AA-MU | A | 128 | QFN48 | Industrial -40°C to 85°C |
| ATSAM3N1CA-AU | A | 64 | LQFP100 | Industrial -40°C to 85°C |
| ATSAM3N1CB-AU | B | 64 | LQFP100 | Industrial -40°C to 85°C |
| ATSAM3N1CA-CU | A | 64 | TFBGA100 | Industrial -40°C to 85°C |
| ATSAM3N1CB-CU | B | 64 | TFBGA100 | Industrial -40°C to 85°C |
| ATSAM3N1BA-AU | A | 64 | LQFP64 | Industrial -40°C to 85°C |
| ATSAM3N1BB-AU | B | 64 | LQFP64 | Industrial -40°C to 85°C |
| ATSAM3N1BA-MU | A | 64 | QFN 64 | Industrial -40°C to 85°C |
| ATSAM3N1BB-MU | B | 64 | QFN 64 | Industrial -40°C to 85°C |
| ATSAM3N1AA-AU | A | 64 | LQFP48 | Industrial -40°C to 85°C |

**Table 39-1. SAM3N Ordering Information (Continued)**

| Ordering Code | MRL | Flash (Kbytes) | Package | Operating Temperature Range |
|---|---|---|---|---|
| ATSAM3N1AB-AU | B | 64 | LQFP48 | Industrial -40°C to 85°C |
| ATSAM3N1AA-MU | A | 64 | QFN48 | Industrial -40°C to 85°C |
| ATSAM3N1AB-MU | B | 64 | QFN48 | Industrial -40°C to 85°C |
| ATSAM3N0CA-AU | A | 32 | LQFP100 | Industrial -40°C to 85°C |
| ATSAM3N0CA-CU | A | 32 | TFBGA100 | Industrial -40°C to 85°C |
| ATSAM3N0BA-AU | A | 32 | LQFP64 | Industrial -40°C to 85°C |
| ATSAM3N0BA-MU | A | 32 | QFN64 | Industrial -40°C to 85°C |
| ATSAM3N0AA-AU | A | 32 | LQFP48 | Industrial -40°C to 85°C |
| ATSAM3N0AA-MU | A | 32 | QFN48 | Industrial -40°C to 85°C |
| ATSAM3N00BA-AU | A | 16 | LQFP64 | Industrial -40°C to 85°C |
| ATSAM3N00BA-MU | A | 16 | QFN64 | Industrial -40°C to 85°C |
| ATSAM3N00AA-AU | A | 16 | LQFP48 | Industrial -40°C to 85°C |
| ATSAM3N00AA-MU | A | 16 | QFN48 | Industrial -40°C to 85°C |

# 40. SAM3N Series Errata

## 40.1 SAM3N4/2/1 Errata - Rev. A Parts

The table below lists the chip IDs for SAM3N4/2/1 revision A parts.

**Table 40-1. Chip IDs: SAM3N4/2/1 Rev. A Parts**

| Chip Name | Revision | Chip ID |
|---|---|---|
| ATSAM3N4C | A | 0x29540960 |
| ATSAM3N2C | A | 0x29590760 |
| ATSAM3N1C | A | 0x29580560 |
| ATSAM3N4B | A | 0x29440960 |
| ATSAM3N2B | A | 0x29490760 |
| ATSAM3N1B | A | 0x29480560 |
| ATSAM3N4A | A | 0x29340960 |
| ATSAM3N2A | A | 0x29390760 |
| ATSAM3N1A | A | 0x29380560 |

## 40.2 Flash Memory

### 40.2.1 Flash: Flash Programming

When writing data into the Flash memory plane (either through the EEFC, using the IAP function or FFPI), the data may not be correctly written (i.e the data written is not the one expected).

**Problem Fix/Workaround**

Set the number of Wait States (WS) at 6 (FWS = 6) during the programming.

### 40.2.2 Flash: Fetching Error after Reading the Unique Identifier

After reading the Unique Identifier (or using the STUI/SPUI command), the processor may fetch wrong instructions. It depends on the code and on the region of the code.

**Problem Fix/Workaround**

In order to avoid this problem, follow the steps below:

1. Set bit 16 of EEFC Flash Mode Register to 1
2. Send the Start Read Unique Identifier command (STUI) by writing the Flash Command Register with the STUI command
3. Wait for the FRDY bit to fall
4. Read the Unique ID (and next bits if required)
5. Send the Stop Read Unique Identifier command (SPUI) by writing the Flash Command Register with the SPUI command
6. Wait for the FRDY bit to rise
7. Clear bit 16 of EEFC Flash Mode Register

Note: During the sequence, the software cannot run out of Flash (so it has to run out of SRAM).

## 40.3 SAM3N1 Errata - Rev. B Parts / SAM3N0/00 - Rev. A Parts

The table below lists the chip IDs for SAM3N1 revision B parts and SAM3N0/00 revision A parts.

**Table 40-2. Chip IDs: SAM3N1 Rev. B Parts / SAM3N0/00 Rev. A Parts**

| Chip Name | Revision | Chip ID |
|---|---|---|
| ATSAM3N1C | B | 0x29580561 |
| ATSAM3N1B | B | 0x29480561 |
| ATSAM3N1A | B | 0x29380561 |
| ATSAM3N0C | A | 0x29580361 |
| ATSAM3N0B | A | 0x29480361 |
| ATSAM3N0A | A | 0x29380361 |
| ATSAM3N00B | A | 0x29450261 |
| ATSAM3N00A | A | 0x29350261 |

### 40.3.1 Flash: Fetching Error after Reading the Unique Identifier

After reading the Unique Identifier (or using the STUI/SPUI command), the processor may fetch wrong instructions. It depends on the code and on the region of the code.

**Problem Fix/Workaround**

In order to avoid this problem, follow the steps below:

1. Set bit 16 of EEFC Flash Mode Register to 1
2. Send the Start Read Unique Identifier command (STUI) by writing the Flash Command Register with the STUI command
3. Wait for the FRDY bit to fall
4. Read the Unique ID (and next bits if required)
5. Send the Stop Read Unique Identifier command (SPUI) by writing the Flash Command Register with the SPUI command
6. Wait for the FRDY bit to rise
7. Clear bit 16 of EEFC Flash Mode Register

Note: During the sequence, the software cannot run out of Flash (so it has to run out of SRAM).

# 41. Revision History

**Table 41-1. 11011C Revision History**

| Date | Comments |
|---|---|
| 16-Apr-15 | General formatting and editorial changes throughout |
| | Section "Description"<br>Added paragraph relating to low-power modes<br>Added paragraph relating to Real-time Event Managment |
| | Section 1. "Features"<br>Updated description of "Low Power Modes"<br>Changed "Up to 6 Three-Channel 16-bit Timer/Counter" to "Up to two 3-channel 16-bit Timer Counters"<br>Added bullet "Register Write Protection" |
| | Section 2. "SAM3N Block Diagram"<br>In Figure 2-1 "SAM3N 100-pin version Block Diagram", Figure 2-2 "SAM3N 64-pin version Block Diagram" and Figure 2-3 "SAM3N 48-pin version Block Diagram", updated System Controller block, added Real-Time Event block, and renumbered two Timer Counter blocks as 0–1 (were previously A–B) |
| | Section 4. "Package and Pinout"<br>Table 4-2 "100-ball TFBGA SAM3N4/2/1/0/00C Pinout": updated to add FFPI signal names |
| | Section 5. "Power Considerations"<br>Added Section 5.2 "Power-up Considerations"<br>Table 5-1 "Low Power Mode Configuration Summary": updated potential wake-up sources (removed "USB wake-up" and "BOD alarm"; added "SM alarm")<br>Updated Figure 5-4 "Core Externally Supplied (Backup Battery)"<br>Updated Section 5.6.1 "Backup Mode", Section 5.7 "Wake-up Sources", and Section 5.8 "Fast Startup" |
| | Section 6. "Input/Output Lines"<br>Table 6-1 "System I/O Configuration Pin List": renamed column header "SYSTEM_IO bit number" to "CCFG_SYSIO Bit No."<br>Section 6.5 "ERASE Pin": in first paragraph, added details relative to reprogramming Flash content; in last sentence, changed "asserting the pin to low does not erase the Flash" to "asserting the pin to high does not erase the Flash" |
| | Added Section 8. "Real-time Event Management" |
| | Section 9. "System Controller"<br>Removed Figure 8-1. System Controller Block Diagram (redundant with Figure 17-1 "Supply Controller Block Diagram") |
| | Section 10. "Peripherals"<br>Table 10-2 "Multiplexing on PIO Controller A (PIOA)": added footnotes on selecting extra functions and system functions<br>Table 10-3 "Multiplexing on PIO Controller B (PIOB)": added footnotes on selecting extra functions and system functions<br>Table 10-4 "Multiplexing on PIO Controller C (PIOC)": added footnotes on selecting extra functions |
| | Section 12. "Debug and Test Features"<br>Section 12.5.2 "Debug Architecture": corrected "Cortex-M3 embeds four functional units" to "Cortex-M3 embeds five functional units" |
| | Section 21. "SAM3N Boot Program"<br>Section 21.5.3 "In Application Programming (IAP) Feature": modified content to reference two arguments instead of a single argument; replaced two instances of "MC_FSR register" with "EEFC_FSR" |
| | Section 25. "Power Management Controller (PMC)"<br>Updated Section 25.10 "Fast Startup" |

**Table 41-1. 11011C Revision History (Continued)**

| Date | Comments |
|---|---|
| 16-Apr-15 | Section 32. “Timer Counter (TC)”<br>Replaced instances of “Master clock” or “MCK” with “peripheral clock” throughout<br>Replaced instances of ‘quadrature decoder logic’ with ‘quadrature decoder’ or ‘QDEC’ throughout<br>Section 32.1 “Description”; updated text and corrected instance of input “TIOA1” to “TIOB1”<br>Section 32.2 “Embedded Characteristics”: corrected total number of TC channels from “Three” to “6”; deleted bullet “Two Global Registers that Act on All Three TC Channels”<br>Section 32.3 “Block Diagram”: added Table 32-1 ”Timer Counter Clock Assignment”; updated descriptions of INT and SYNC in Table 32-2 ”Signal Name Description”<br>Added Table 32-5 ”Peripheral IDs”<br>Section 32.6.3 “Clock Selection”: updated names of internal clock signals<br>Section 32.6.11.1 “WAVSEL = 00”: replaced “0xFFFF” with “$2^{16}$-1” in first paragraph<br>In Figure 32-9 ”WAVSEL = 10 without Trigger” and Figure 32-10 ”WAVSEL = 10 with Trigger”, replaced “0xFFFF” with “$2^{n}$-1” (with “n” representing counter size)<br>Section 32.6.11.3 “WAVSEL = 01”: replaced “0xFFFF” with “$2^{16}$-1” in first paragraph<br>In Figure 32-13 ”WAVSEL = 11 without Trigger” and Figure 32-14 ”WAVSEL = 11 with Trigger”, replaced “0xFFFF” with “$2^{n}$-1” (with “n” representing counter size)<br>Section 32.6.14.1 “Description”: in first paragraph, changed TIOA1 into TIOB1 and corrected link to Figure 32-15<br>Section 32.6.14.2 “Input Pre-processing”: deleted sentence “Filters can be disabled using the FILTER bit in the TC_BMR”<br>Figure 32-16 ”Input Stage”: replaced “FILTER” with “MAXFILTER > 0”<br>Section 32.6.14.3 “Direction Status and Change Detection”: rewrote sixth paragraph for clarity<br>Section 32.6.14.4 “Position and Rotation Measurement”: rewrote first paragraph for clarity and changed TIOA1 into TIOB1; at end of second paragraph, defined External Trigger Edge and External Trigger configuration in TC_CMR<br>Section 32.6.14.5 “Speed Measurement”: in fifth paragraph, replaced “EDGTRG can be set to 0x01” with “ETRGEDG must be set to 0x01”; in seventh paragraph, replaced sentence “The speed can be read on TC_RA0 register in TC_CMR0” with “The speed can be read on field RA in register TC_RA0”<br>Revised Section 32.6.16 “Register Write Protection”<br>Table 32-6 ”Register Mapping”: defined offset 0xD8 and offset range 0xE8–0xFC as reserved<br>Section 32.7.2 “TC Channel Mode Register: Capture Mode”: in ‘Name’ line, replaced “(WAVE = 0)” with “(CAPTURE_MODE)”; updated TCCLKS field values 0–4<br>Section 32.7.3 “TC Channel Mode Register: Waveform Mode”: in ‘Name’ line, replaced “(WAVE = 1)” with “(WAVEFORM_MODE)”; updated TCCLKS field values 0–4; added note to ENETRG bit description description<br>Section 32.7.5 “TC Counter Value Register”: in CV field description, added notation “IMPORTANT: For 16-bit channels, CV field size is limited to register bits 15:0”<br>Section 32.7.6 “TC Register A”: in RA field description, added notation “IMPORTANT: For 16-bit channels, RA field size is limited to register bits 15:0”<br>Section 32.7.7 “TC Register B”: in RB field description, added notation “IMPORTANT: For 16-bit channels, RB field size is limited to register bits 15:0”<br>Section 32.7.8 “TC Register C”: in RC field description, added notation “IMPORTANT: For 16-bit channels, RC field size is limited to register bits 15:0”<br>Section 32.7.9 “TC Status Register”: updated bit descriptions<br>Section 32.7.14 “TC Block Mode Register”:<br>- removed FILTER bit (register bit 19 now reserved)<br>- corrected TC2XC2S field configuration values: value 2 is TIOA0 (was TIOA1); value 3 is TIOA1 (was TIOA2)<br>Section 32.7.19 “TC Write Protection Mode Register”: updated bit/field descriptions |

**Table 41-1. 11011C Revision History (Continued)**

| Date | Comments |
|---|---|
| 16-Apr-15 | Section 36. “Electrical Characteristics”<br>Updated and harmonized parameter symbols throughout<br>Table 36-2 ”DC Characteristics”: removed parameter “Input Capacitance“; updated footnotes<br>Table 36-3 ”1.8V Voltage Regulator Characteristics”: updated footnotes; updated conditions for $CD_{IN}$ and $CD_{OUT}$<br>Table 36-4 ”Core Power Supply Brownout Detector Characteristics”: added parameter “Reset Period”<br>Table 36-7 ”DC Flash Characteristics”: replaced TBDs with values for “Active Current” (64-bit Mode Read Access)<br>Updated Section 36.3.1.1 “Configuration A: Embedded Slow Clock RC Oscillator Enabled”<br>Updated Section 36.3.1.2 “Configuration B: 32.768 kHz Crystal Oscillator Enabled”<br>Table 36-8 ”Power Consumption for Backup Mode (SAM3N4/2/1 MRL A)”: replaced TBDs with values (for 85°C conditions)<br>Table 36-21 ”32.768 kHz Crystal Oscillator Characteristics”: removed parameter “Maximum external capacitor on XIN32 and XOUT32”; added parameter “Allowed Crystal Capacitance Load”<br>Updated Figure 36-12 ”XIN32 Clock Timing”<br>Table 36-24 ”3 to 20 MHz Crystal Oscillator Characteristics”: removed parameter “Maximum external capacitor on XIN and XOUT”; added parameter “Allowed Crystal Capacitance Load”; removed all footnotes<br>Table 36-25 ”Crystal Characteristics”: for ESR values, corrected unit ‘W’ to ‘Ω’<br>Updated Figure 36-14 ”XIN Clock Timing”<br>Updated Section 36.4.9 “Crystal Oscillators Design Consideration Information”<br>Merged PLL characteristics into single Table 36-27 ”PLL Characteristics”<br>Table 36-30 ”External Voltage Reference Input”: updated conditions for parameter “ADVREF Input Voltage Range” and deleted duplicated Table 35-33. “External Voltage Reference Input”<br>Updated Section 36.6.1 “Track and Hold Time versus Source Output Impedance”<br>Section 36.8.3.1 “Maximum SPI Frequency”: updated content under “Master Write Mode” and “Master Read Mode”<br>Table 36-41 ”Two-wire Serial Bus Requirements”: in bottom row, replaced duplicated parameter “Hold Time (repeated) START Condition” with new parameter “Bus free time between a STOP and START condition”<br>Section 36.8.6 “Embedded Flash Characteristics”: corrected “field FWS of the MC_FMR register” to “field FWS in the EEFC_FMR”; updated text and replaced two wait state tables with single ”Table 36-42 ”Embedded Flash Wait State - VDDCORE 1.65V/1.80V”<br>Table 36-43 ”AC Flash Characteristics”: changed unit ‘ms’ to ‘s’ for “Full Chip Erase” parameter |
| | Section 37. “Mechanical Characteristics”<br>Updated Table 37-4 ”100-ball TFBGA Soldering Information (Substrate Level)”<br>Updated Table 37-5 ”100-ball TFBGA Device Maximum Weight”<br>Updated Table 37-7 ”100-ball TFBGA Package Reference”<br>Updated Figure 37-5 ”64-pad QFN Package Drawing” and removed redundant Table 36-14. “64-pad QFN Package Dimensions (in mm)” |
| | Inserted Section 38. “Marking” (was previously subsection of Section 40. “SAM3N Series Errata”) |
| | Section 39. “Ordering Information”<br>Table 39-1 ”SAM3N Ordering Information”: removed “Package Type” column (this information is provided on the Atmel website) |

| Doc. Rev. 11011B | Comments | Change Request Ref. |
|---|---|---|
| | Overview: | |
| | All mentions of 100-ball LFBGA changed into 100-ball TFBGA | 8044 |
| | Numerous updates | 7634 |
| | Section 7. “Memories” on page 25, Heading was ‘Memories’. Changed to ‘Product Mapping’ | 7685 |
| | Several updates to clarify that only 1 USART has ISO7816 capability | 7857 |
| | Two typos corrected in chapter 12 and 32 | 7913 |
| | Section 5. “Power Considerations” on page 16: Figure 6., Changed from Edge detection to Level detection.<br>Section 25.10 “Fast Startup” on page 337, Added ‘SM’ for Fast Startup detection | 7922 |
| | Section “Features”, extented range for Flash (now from 16Kbytes) and SRAM (now from 4Kbytes) | 8106 |
| | Section 1.1 “Configuration Summary” on page 3, table extended | |
| | Section 2. “SAM3N Block Diagram” on page 4: Figure 2-1 and Figure 2-2 and Figure 2-3, updated FLASH and SRAM boxes | |
| | Figure 3-1, added table note for ’Internal pull-up disabled’ under ’’Comments’ in ‘ICE and JTAG’ secion | |
| | Whole doc.. Replaced ‘SAM3N4/2/1’ by ‘SAM3N4/2/1/0/00’ | |
| | Section 7.:: | |
| | Figure 7., added SAM3N0 and SAM3N00 product information | |
| | Figure 7.2.3, added SAM3N0 and SAM3N00 Flash bank information | |
| | Section 7.2.3.5 “Lock Regions” on page 27, added lock bit information for SAM3N0 and SAM3N00 | |
| | Section 4.1.4 “100-ball TFBGA Pinout” on page 12, whole pinout table updated | 7201 |
| | Updated package dimensions in ‘Features’ | 7965 |
| | Section 36-2 “DC Characteristics” on page 709, Pull-down Resistor values updated | rfo |
| | Section 36-7 “DC Flash Characteristics” on page 714, Max value for ‘25°C /VDDCORE = 1.95V’ updated | rfo |
| | Section 36-35 “Static Performance Characteristics” on page 731, updated values for Integral and Differential Non-linearity parameters | 8189 |
| | Section 36-3 “1.8V Voltage Regulator Characteristics” on page 711, updated values for ‘Dropout Voltage’ | |
| | Section 24.2 “Embedded Characteristics” on page 326, changed sentence “Processor Clock (HCLK), must be switched off...” | 8217 |
| | CHIPID: | 8106 |
| | Section 26. “Chip Identifier (CHIPID)” on page 369: Figure 26-1, table updated with new chip names | |
| | Debug and Test Features: | |
| | Section 12. “Debug and Test Features”: | |
| | Section 12.5.7 “IEEE® 1149.1 JTAG Boundary Scan” on page 203, Updated. | 7489 |
| | Section 12.4 “Debug and Test Pin Description” on page 199: Figure 12-1, added table note for TDO/TRACESWO | 8106 |

| Doc. Rev. 11011B | Comments | Change Request Ref. |
|---|---|---|
| | ELEC:<br>Section 36.2 “DC Characteristics” on page 709:<br>PULLUP Pull-up Resistor NRST: New values added<br>PULLDOWN Pull-down Resistor: Changed signal names and added one line for signal names PB10-PB11<br>Section 36. “Electrical Characteristics” on page 708:<br>Table 36-19, “32 kHz RC Oscillator Characteristics,” on page 722, changed parameter ‘Frequency Temperature Dependency’<br>Table 36-4, “Core Power Supply Brownout Detector Characteristics,” on page 712, changed MAX value of VTH+<br>Section 36.8.6 “Embedded Flash Characteristics” on page 740, added note regarding erasing Flash contents | 8077<br>8174<br>8223 |
| | Errata:<br>Section 40. “SAM3N Series Errata” on page 753:<br>Added section: Section 40.1 “SAM3N4/2/1 Errata - Rev. A Parts” on page 753<br>Added section: Section 40.3 “SAM3N1 Errata - Rev. B Parts / SAM3N0/00 - Rev. A Parts” on page 754<br>Section 40.1 “SAM3N4/2/1 Errata - Rev. A Parts” on page 753 and Section 40.3 “SAM3N1 Errata - Rev. B Parts / SAM3N0/00 - Rev. A Parts” on page 754, Added errata ‘Flash: Fetching Error after Reading the Unique Identifier’ | 8106<br>7978 |
| | FFPI:<br>Section 20-1 “Signal Description List” on page 285, Text for ‘Function’ changed to ‘Main Clock Input’ | 7851 |
| | Ordering Information:<br>Section 39. “Ordering Information” on page 751: Table 39-1:<br>Updated and added ordering codes<br>Corrected multiple instances of wrong Package types for 128 and 256Kbytes devices | 8106<br>RFO |
| | PMC:<br>Section 25.3 “Block Diagram” on page 334, figure updated with ‘PMC_PCKx’<br>Section 25.10 “Fast Startup” on page 337, SUPC_FSMR changed to PMC_FSMR and SUPC_FSPR changed to PMC_FSPR | 7915<br>8010 |
| | USART:<br>Table 36-40, “USART SPI Timings,” on page 738, Changed 'MCK' --> 'tCPMCK' and 'SCK' --> 'tCPSCK' | 7651 |

| Doc. Rev. 11011A | Comments | Change Request Ref. |
|---|---|---|
| | 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**