

## Features

- Incorporates the ARM7TDMI® ARM® Thumb® Processor
  - High-performance 32-bit RISC Architecture
  - High-density 16-bit Instruction Set
  - Leader in MIPS/Watt
- EmbeddedICE™ In-circuit Emulation, Debug Communication Channel Support
- 256 Kbytes of Internal High-speed Flash, Organized in 1024 Pages of 256 Bytes
  - Single Cycle Access at Up to 30 MHz in Worst Case Conditions
  - Prefetch Buffer Optimizing Thumb Instruction Execution at Maximum Speed
  - Page Programming Time: 6 ms, Including Page Auto-erase, Full Erase Time: 15 ms
  - 10,000 Write Cycles, 10-year Data Retention Capability, Sector Lock Capabilities
- 32K Bytes of Internal High-speed SRAM, Single-cycle Access at Maximum Speed
- Memory Controller (MC)
  - Embedded Flash Controller, Abort Status and Misalignment Detection
  - Memory Protection Unit
- Reset Controller (RSTC)
  - Based on Three Power-on Reset Cells
  - Provides External Reset Signal Shaping and Reset Sources Status
- Clock Generator (CKGR)
  - Low-power RC Oscillator, 3 to 20 MHz On-chip Oscillator and One PLL
- Power Management Controller (PMC)
  - Power Optimization Capabilities, including Slow Clock Mode (Down to 500 Hz), Idle Mode, Standby Mode and Backup Mode
  - Four Programmable External Clock Signals
- Advanced Interrupt Controller (AIC)
  - Individually Maskable, Eight-level Priority, Vectored Interrupt Sources
  - Four External Interrupt Sources and One Fast Interrupt Source, Spurious Interrupt Protected
- Debug Unit (DBGU)
  - 2-wire UART and Support for Debug Communication Channel interrupt
- Periodic Interval Timer (PIT)
  - 20-bit Programmable Counter plus 12-bit Interval Counter
- Windowed Watchdog (WDT)
  - 12-bit key-protected Programmable Counter
  - Provides Reset or Interrupt Signal to the System
  - Counter May Be Stopped While the Processor is in Debug Mode or in Idle State
- Real-time Timer (RTT)
  - 32-bit Free-running Counter with Alarm
  - Runs Off the Internal RC Oscillator
- Two Parallel Input/Output Controllers (PIO)
  - Sixty-two Programmable I/O Lines Multiplexed with up to Two Peripheral I/Os
  - Input Change Interrupt Capability on Each I/O Line
  - Individually Programmable Open-drain, Pull-up resistor and Synchronous Output
- Shutdown Controller (SHDWC)
  - Programmable Shutdown Pin and Wake-up Circuitry
- Two 32-bit Battery Backup Registers for a Total of 8 Bytes
- One 8-channel 20-bit PWM Controller (PWMC)
- One USB 2.0 Full Speed (12 Mbits per Second) Device Port
  - On-chip Transceiver, 2376-byte Configurable Integrated FIFOs



## AT91 ARM Thumb-based Microcontrollers

### AT91SAM7A3

## Preliminary





- Nineteen Peripheral DMA Controller (PDC) Channels
- Two CAN 2.0B Active Controllers, Supporting 11-bit Standard and 29-bit Extended Identifiers
  - 16 Fully Programmable Message Object Mailboxes, 16-bit Time Stamp Counter
- Two 8-channel 10-bit Analog-to-Digital Converter
- Three Universal Synchronous/Asynchronous Receiver Transmitters (USART)
  - Individual Baud Rate Generator, IrDA® Infrared Modulation/Demodulation
  - Support for ISO7816 T0/T1 Smart Card, Hardware Handshaking, RS485 Support
- Two Master/Slave Serial Peripheral Interfaces (SPI)
  - 8- to 16-bit Programmable Data Length, Four External Peripheral Chip Selects
- Three 3-channel 16-bit Timer/Counters (TC)
  - Three External Clock Inputs, Two Multi-purpose I/O Pins per Channel
  - Double PWM Generation, Capture/Waveform Mode, Up/Down Capability
- Two Synchronous Serial Controllers (SSC)
  - Independent Clock and Frame Sync Signals for Each Receiver and Transmitter
  - I<sup>2</sup>S Analog Interface Support, Time Division Multiplex Support
  - High-speed Continuous Data Stream Capabilities with 32-bit Data Transfer
- One Two-wire Interface (TWI)
  - Master Mode Support Only, All Two-wire Atmel EEPROM's Supported
- Multimedia Card Interface (MCI)
  - Compliant with Multimedia Cards and SD Cards
  - Automatic Protocol Control and Fast Automatic Data Transfers with PDC, MMC and SDCard Compliant
- IEEE® 1149.1 JTAG Boundary Scan on All Digital Pins
- Required Power Supplies
  - Embedded 1.8V Regulator, Drawing up to 130 mA for the Core and the External Components, Enables 3.3V Single Supply Mode
  - 3.3V VDD3V3 Regulator, I/O Lines and Flash Power Supply
  - 1.8V VDD1V8 Output of the Voltage Regulator and Core Power Supply
  - 3V to 3.6V VDDANA ADC Power Supply
  - 3V to 3.6V VDDBU Backup Power Supply
- 5V-tolerant I/Os
- Fully Static Operation: Up to 60 MHz at 1.65V and 85°C Worst Case Conditions
- Available in a 100-lead LQFP Green Package

## 1. Description

The AT91SAM7A3 is a member of a series of 32-bit ARM7™ microcontrollers with an integrated CAN controller. It features a 256-Kbyte high-speed Flash and 32-Kbyte SRAM, a large set of peripherals, including two 2.0B full CAN controllers, and a complete set of system functions minimizing the number of external components. The device is an ideal migration path for 8-bit microcontroller users looking for additional performance and extended memory.

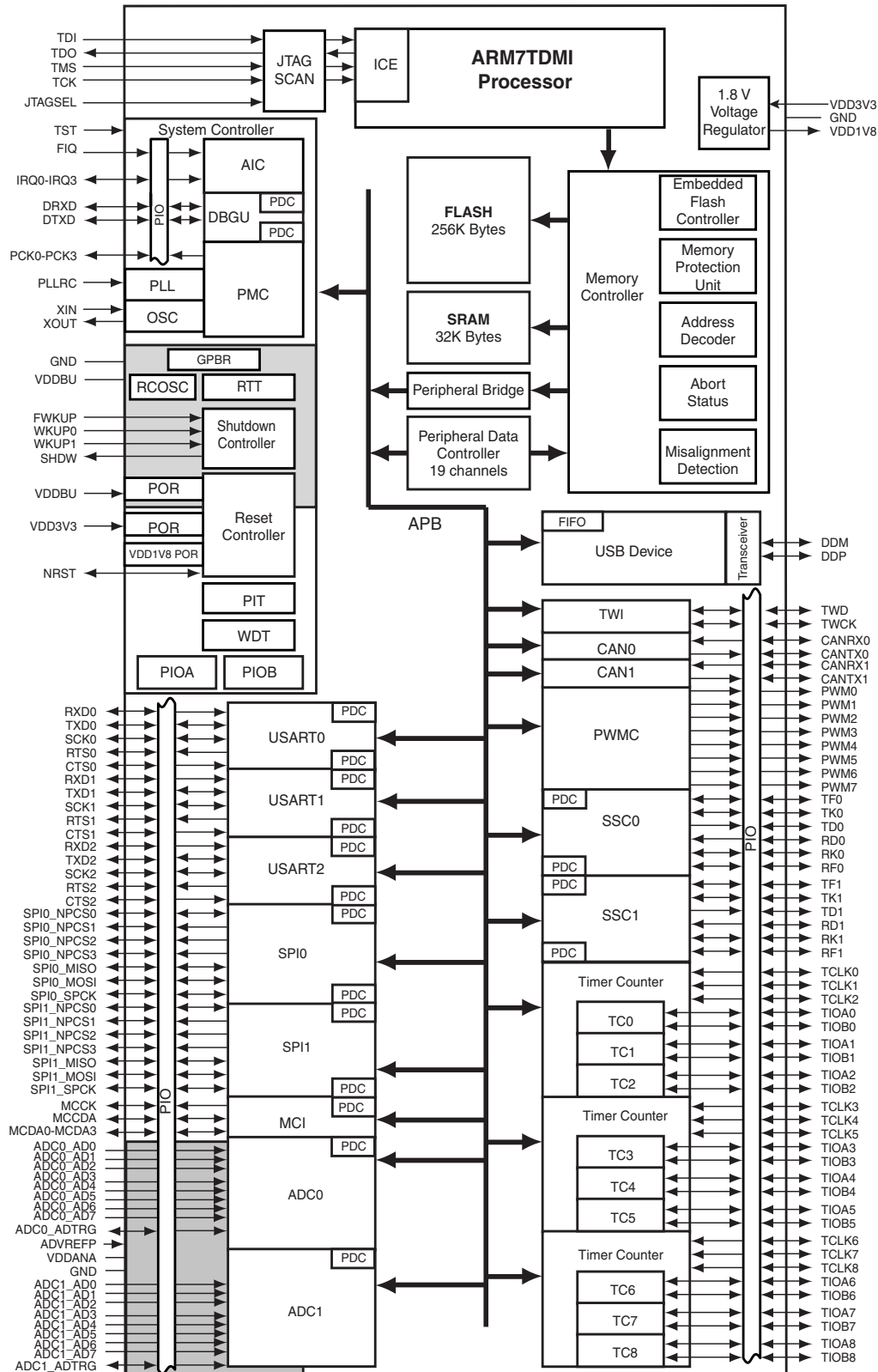
The embedded Flash memory can be programmed in-system via the JTAG-ICE interface. Built-in lock bits protect the firmware from accidental overwrite.

The AT91SAM7A3 integrates a complete set of features facilitating debug, including a JTAG Embedded ICE interface, misalignment detector, interrupt driven debug communication channel for user configurable trace on a console, and JTAG boundary scan for board level debug and test.

By combining a high-performance 32-bit RISC processor with a high-density 16-bit instruction set, Flash and SRAM memory, a wide range of peripherals including CAN controllers, 10-bit ADC, Timers and serial communication channels, on a monolithic chip, the AT91SAM7A3 is ideal for many compute-intensive embedded control applications.

## 2. Block Diagram

Figure 2-1. AT91SAM7A3 Block Diagram



## 3. Signal Description

**Table 3-1.** Signal Description

Signal Name	Function	Type	Active Level	Comments
<b>Power</b>				
VDD3V3	1.8V Voltage Regulator, I/O Lines and Flash Power Supply	Power		3.0V to 3.6V
VDDDBU	Backup I/O Lines Power Supply	Power		3V to 3.6V
VDDANA	Analog Power Supply	Power		3V to 3.6V
VDD1V8	1.8V Voltage Regulator Output and Core Power Supply	Power		1.85V typical
VDDPLL	1.8V PLL Power Supply	Power		1.65V to 1.95V
GND	Ground	Ground		
<b>Clocks, Oscillators and PLLs</b>				
XIN	Main Oscillator Input	Input		
XOUT	Main Oscillator Output	Output		
PLLRC	PLL Filter	Input		
PCK0 - PCK3	Programmable Clock Output	Output		
SHDW	Shut-Down Control	Output		Open Drain.
WKUP0 - WKUP1	Wake-Up Inputs	Input		Accept between 0V and VDDDBU
FWKUP	Force Wake Up	Input		Accept between 0V and VDDDBU External pull-up resistor needed.
<b>ICE and JTAG</b>				
TCK	Test Clock	Input		No pull-up resistor
TDI	Test Data In	Input		No pull-up resistor
TDO	Test Data Out	Output		
TMS	Test Mode Select	Input		No pull-up resistor
JTAGSEL	JTAG Selection	Input		Pull-down resistor
<b>Reset/Test</b>				
NRST	Microcontroller Reset	I/O	Low	
TST	Test Mode Select	Input	High	Pull-down resistor
<b>Debug Unit</b>				
DRXD	Debug Receive Data	Input		
DTXD	Debug Transmit Data	Output		

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

Signal Name	Function	Type	Active Level	Comments
<b>AIC</b>				
IRQ0 - IRQ3	External Interrupt Inputs	Input		
FIQ	Fast Interrupt Input	Input		
<b>PIO</b>				
PA0 - PA31	Parallel IO Controller A	I/O		Pulled-up input at reset
PB0 - PB29	Parallel IO Controller B	I/O		Pulled-up input at reset
<b>Multimedia Card Interface</b>				
MCKK	Multimedia Card Clock	Output		
MCCDA	Multimedia Card A Command	I/O		
MCDA0 - MCDA3	Multimedia Card A Data	I/O		
<b>USB Device Port</b>				
DDM	USB Device Port Data -	Analog		
DDP	USB Device Port Data +	Analog		
<b>USART</b>				
SCK0 - SCK1 - SCK2	Serial Clock	I/O		
TXD0 - TXD1 - TXD2	Transmit Data	I/O		
RXD0 - RXD1 - RXD2	Receive Data	Input		
RTS0 - RTS1 - RTS2	Request To Send	Output		
CTS0 - CTS1 - CTS2	Clear To Send	Input		
<b>Synchronous Serial Controller</b>				
TD0 - TD1	Transmit Data	Output		
RD0 - RD1	Receive Data	Input		
TK0 - TK1	Transmit Clock	I/O		
RK0 - RK1	Receive Clock	I/O		
TF0 - TF1	Transmit Frame Sync	I/O		
RF0 - RF1	Receive Frame Sync	I/O		
<b>Timer/Counter</b>				
TCLK0 - TCLK8	External Clock Input	Input		
TIOA0 - TIOA8	I/O Line A	I/O		
TIOB0 - TIOB8	I/O Line B	I/O		
<b>PWM Controller</b>				
PWM0 - PWM7	PWM Channels	Output		

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

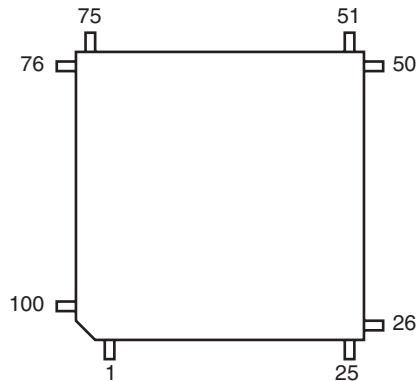
Signal Name	Function	Type	Active Level	Comments
<b>SPI</b>				
SPI0_MISO SPI1_MISO	Master In Slave Out	I/O		
SPI0_MOSI SPI1_MOSI	Master Out Slave In	I/O		
SPI0_SPCK SPI1_SPCK	SPI Serial Clock	I/O		
SPI0_NPCS0 SPI1_NPCS0	SPI Peripheral Chip Select 0	I/O	Low	
SPI0_NPCS1 - SPI0_NPCS3 SPI1_NPCS1 - SPI1_NPCS3	SPI Peripheral Chip Select	Output	Low	
<b>Two-wire Interface</b>				
TWD	Two-wire Serial Data	I/O		
TWCK	Two-wire Serial Clock	I/O		
<b>Analog-to-Digital Converter</b>				
ADC0_AD0 - ADC0_AD7 ADC1_AD0 - ADC1_AD7	Analog Inputs	Analog		Digital pulled-up inputs at reset
ADVREFP	Analog Positive Reference	Analog		
ADC0_ADTRG ADC1_ADTRG	ADC Trigger	Input		
<b>CAN Controller</b>				
CANRX0-CANRX1	CAN Inputs	Input		
CANTX0-CANTX1	CAN Outputs	Output		

## 4. Package

### 4.1 100-lead LQFP Package Outline

Figure 4-1 shows the orientation of the 100-lead LQFP package. A detailed mechanical description is given in the Mechanical Characteristics section of the full datasheet.

Figure 4-1. 100-lead LQFP Outline (Top View)





## 4.2 Pinout

**Table 4-1.** Pinout in 100-lead LQFP Package

1	GND	26	VDDBU	51	PA20	76	PLLRC
2	NRST	27	FWKUP	52	PA21	77	VDDANA
3	TST	28	WKUP0	53	PA22	78	ADVREFP
4	PB13	29	WKUP1	54	PA23	79	GND
5	PB12	30	SHDW	55	PA24	80	PB14/ADC0_AD0
6	PB11	31	GND	56	PA25	81	PB15/ADC0_AD1
7	PB10	32	PA4	57	PA26	82	PB16/ADC0_AD2
8	PB9	33	PA5	58	PA27	83	PB17/ADC0_AD3
9	PB8	34	PA6	59	VDD1V8	84	PB18/ADC0_AD4
10	PB7	35	PA7	60	GND	85	PB19/ADC0_AD5
11	PB6	36	PA8	61	VDD3V3	86	PB20/ADC0_AD6
12	PB5	37	PA9	62	PA28	87	PB21/ADC0_AD7
13	PB4	38	VDD3V3	63	PA29	88	VDD3V3
14	PB3	39	GND	64	PA30	89	PB22/ADC1_AD0
15	VDD3V3	40	VDD1V8	65	PA31	90	PB23/ADC1_AD1
16	GND	41	PA10	66	JTAGSEL	91	PB24/ADC1_AD2
17	VDD1V8	42	PA11	67	TDI	92	PB25/ADC1_AD3
18	PB2	43	PA12	68	TMS	93	PB26/ADC1_AD4
19	PB1	44	PA13	69	TCK	94	PB27/ADC1_AD5
20	PB0	45	PA14	70	TDO	95	PB28/ADC1_AD6
21	PA0	46	PA15	71	GND	96	PB29/ADC1_AD7
22	PA1	47	PA16	72	VDDPLL	97	DDM
23	PA2	48	PA17	73	XOUT	98	DDP
24	PA3	49	PA18	74	XIN	99	VDD1V8
25	GND	50	PA19	75	GND	100	VDD3V3

## 5. Power Considerations

### 5.1 Power Supplies

The AT91SAM7A3 has five types of power supply pins:

- VDD3V3 pins. They power the voltage regulator, the I/O lines, the Flash and the USB transceivers; voltage ranges from 3.0V to 3.6V, 3.3V nominal.
- VDD1V8 pins. They are the outputs of the 1.8V voltage regulator and they power the logic of the device.
- VDDPLL pin. It powers the PLL; voltage ranges from 1.65V to 1.95V, 1.8V typical. They can be connected to the VDD1V8 pin with decoupling capacitor.
- VDDBU pin. It powers the Slow Clock oscillator and the Real Time Clock, as well as a part of the System Controller; ranges from 3.0V and 3.6V, 3.3V nominal.
- VDDANA pin. It powers the ADC; ranges from 3.0V and 3.6V, 3.3V nominal.

No separate ground pins are provided for the different power supplies. Only GND pins are provided and should be connected as shortly as possible to the system ground plane.

### 5.2 Voltage Regulator

The AT91SAM7A3 embeds a voltage regulator that consumes less than 120  $\mu$ A static current and draws up to 130 mA of output current.

Adequate output supply decoupling is mandatory for VDD1V8 (pin 99) to reduce ripple and avoid oscillations. The best way to achieve this is to use two capacitors in parallel: one external 470 pF (or 1 nF) NPO capacitor must be connected between VDD1V8 and GND as close to the chip as possible. One external 3.3  $\mu$ F (or 4.7  $\mu$ F) X7R capacitor must be connected between VDD1V8 and GND.

All other VDD1V8 pins must be externally connected and have a proper decoupling capacitor (at least 100 nF).

Adequate input supply decoupling is mandatory for VDD3V3 (pin 100) in order to improve start-up stability and reduce source voltage drop. The input decoupling capacitor should be placed close to the chip. For example, two capacitors can be used in parallel: 100 nF NPO and 4.7  $\mu$ F X7R.

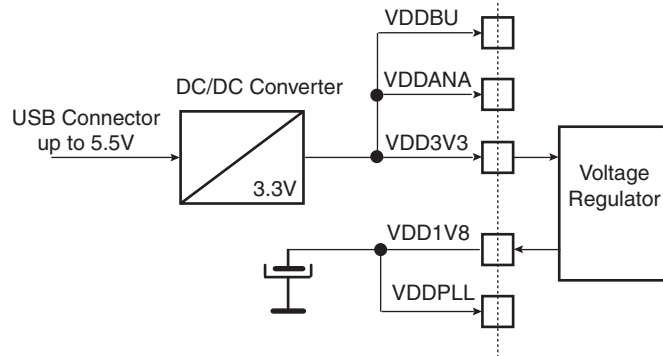
All other VDD3V3 pins must be externally connected and have a proper decoupling capacitor (at least 100 nF).

## 5.3 Typical Powering Schematics

### 5.3.1 3.3V Single Supply

The AT91SAM7A3 supports a 3.3V single supply mode. The internal regulator is connected to the 3.3V source and its output feeds VDDPLL. [Figure 5-1](#) shows the power schematics to be used for USB bus-powered systems.

**Figure 5-1.** 3.3V System Single Power Supply Schematics



## 6. I/O Lines Considerations

### 6.1 JTAG Port Pins

TMS, TDI and TCK are schmitt trigger inputs. TMS and TCK are 5V-tolerant, TDI is not. TMS, TDI and TCK do not integrate any resistors and have to be pulled-up externally.

TDO is an output, driven at up to VDD3V3.

The JTAGSEL pin is used to select the JTAG boundary scan when asserted at a high level.

The JTAGSEL pin integrates a permanent pull-down resistor so that it can be left unconnected for normal operations.

### 6.2 Test Pin

The TST pin is used for manufacturing tests and integrates a pull-down resistor so that it can be left unconnected for normal operations. Driving this line at a high level leads to unpredictable results.

### 6.3 Reset 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. There is no constraint on the length of the reset pulse, and the reset controller can guarantee a minimum pulse length. This allows connection of a simple push-button on the NRST pin as system user reset, and the use of the NRST signal to reset all the components of the system.

### 6.4 PIO Controller A and B Lines

All the I/O lines PA0 to PA31 and PB0 to PB29 are 5V-tolerant and all integrate a programmable pull-up resistor. Programming of this pull-up resistor is performed independently for each I/O line through the PIO Controllers.

5V-tolerant means that the I/O lines can drive voltage level according to VDD3V3, but can be driven with a voltage at up to 5.5V. However, driving an I/O line with a voltage over VDD3V3 while the programmable pull-up resistor is enabled creates a current path through the pull-up resistor from the I/O line to VDDIO. Care should be taken, especially at reset, as all the I/O lines default as inputs with pull-up resistor enabled at reset.

### 6.5 Shutdown Logic Pins

The SHDW pin is an open drain output. It can be tied to VDDBU with an external pull-up resistor.

The FWUP, WKUP0 and WKUP1 pins are input-only. They can accept voltages only between 0V and VDDBU. It is recommended to tie these pins either to GND or to VDDBU with an external resistor.

### 6.6 I/O Line Drive Levels

All the I/O lines can draw up to 2 mA.

## 7. Processor and Architecture

### 7.1 ARM7TDMI Processor

- RISC Processor Based on ARMv4T Von Neumann Architecture
  - Runs at up to 60 MHz, providing 0.9 MIPS/MHz
- Two instruction sets
  - ARM high-performance 32-bit Instruction Set
  - Thumb high code density 16-bit Instruction Set
- Three-stage pipeline architecture
  - Instruction Fetch (F)
  - Instruction Decode (D)
  - Execute (E)

### 7.2 Debug and Test Features

- Integrated EmbeddedICE™ (embedded in-circuit emulator)
  - Two watchpoint units
  - Test access port accessible through a JTAG protocol
  - Debug communication channel
- Debug Unit
  - Two-pin UART
  - Debug communication channel interrupt handling
  - Chip ID Register
- IEEE1149.1 JTAG Boundary-scan on all digital pins

### 7.3 Memory Controller

- Bus Arbiter
  - Handles requests from the ARM7TDMI and the Peripheral Data Controller
- Address Decoder Provides Selection Signals for
  - Three internal 1Mbyte memory areas
  - One 256 Mbyte embedded peripheral area
- Abort Status Registers
  - Source, Type and all parameters of the access leading to an abort are saved
  - Facilitates debug by detection of bad pointers
- Misalignment Detector
  - Alignment checking of all data accesses
  - Abort generation in case of misalignment
- Remap Command
  - Remaps the Internal SRAM in place of the embedded non-volatile memory
  - Allows handling of dynamic exception vectors
- 16-area Memory Protection Unit
  - Individually programmable size between 1K Bytes and 1M Bytes

- Individually programmable protection against write and/or user access
- Peripheral protection against write and/or user access
- Embedded Flash Controller
  - Embedded Flash interface, up to three programmable wait states
  - Read-optimized interface, buffering and anticipating the 16-bit requests, reducing the required wait states
  - Password-protected program, erase and lock/unlock sequencer
  - Automatic consecutive programming, erasing and locking operations
  - Interrupt generation in case of forbidden operation

## 7.4 Peripheral DMA Controller

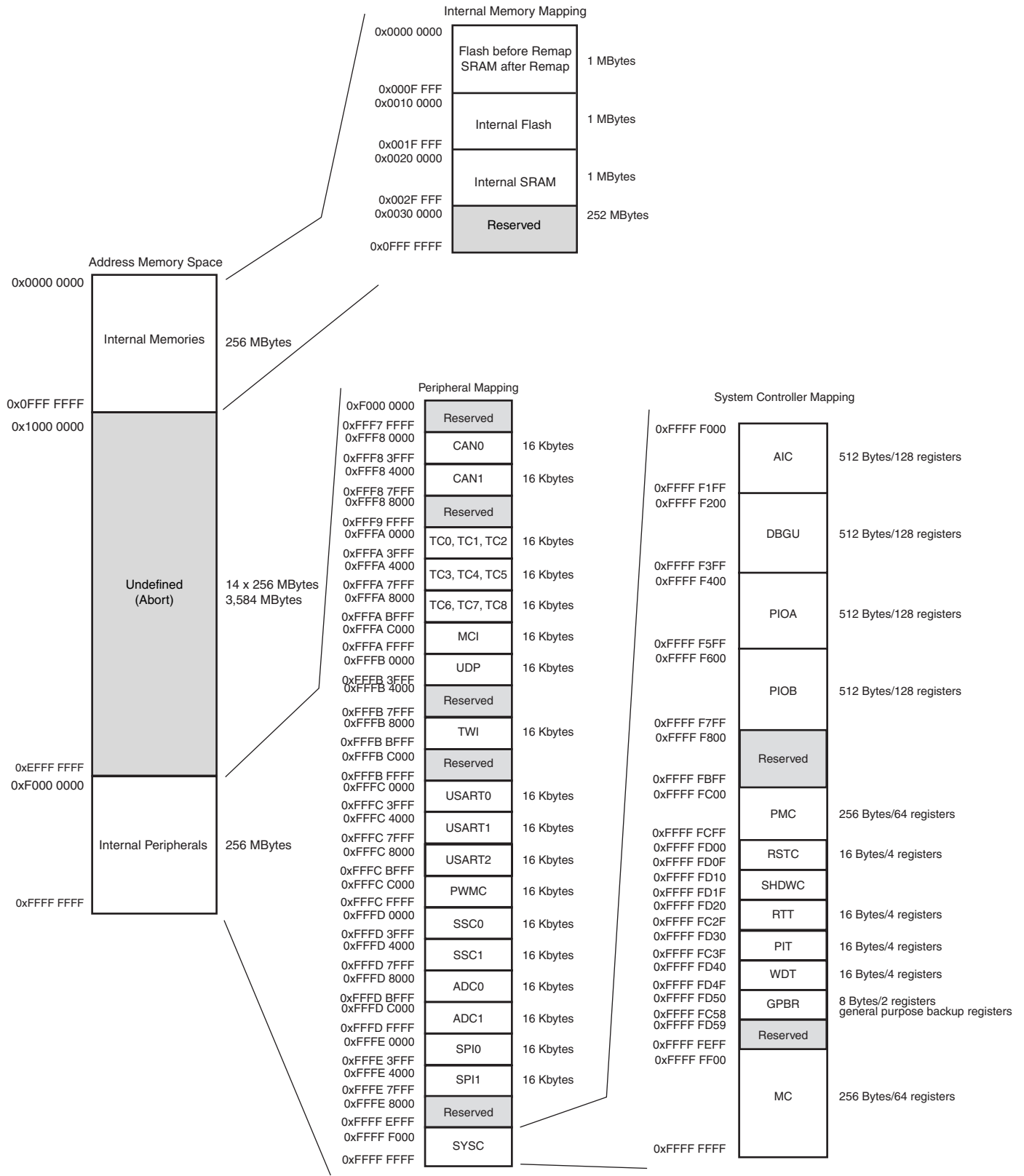
- Handles data transfer between peripherals and memories
- Nineteen Channels
  - Two for each USART
  - Two for the Debug Unit
  - Two for each Serial Synchronous Controller
  - Two for each Serial Peripheral Interface
  - One for the Multimedia Card Interface
  - One for each Analog-to-Digital Converter
- 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 requirements

## **8. Memory**

### **8.1 Embedded Memories**

- 256 Kbytes of Flash Memory
  - 1024 pages of 256 bytes.
  - Fast access time, 30 MHz single cycle access in worst case conditions.
  - Page programming time: 6 ms, including page auto-erase
  - Full erase time: 15 ms
  - 10,000 write cycles, 10-year data retention capability
  - 16 lock bits, each protecting 16 pages
- 32 Kbytes of Fast SRAM
  - Single-cycle access at full speed

**Figure 8-1. AT91SAM7A3 Memory Mapping**





## 8.2 Memory Mapping

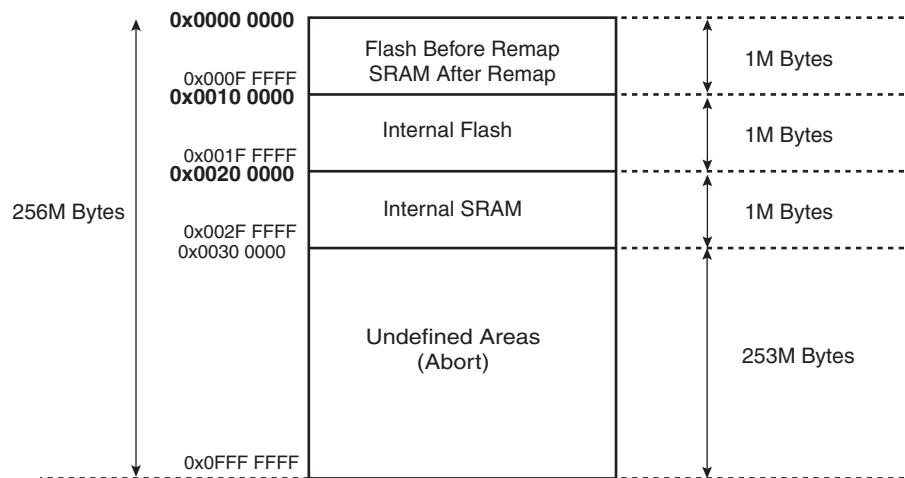
### 8.2.1 Internal SRAM

The AT91SAM7A3 embeds a high-speed 32-Kbyte SRAM bank. After reset and until the Remap Command is performed, the SRAM is only accessible at address 0x0020 0000. After Remap, the SRAM also becomes available at address 0x0.

### 8.2.2 Internal Flash

The AT91SAM7A3 features one bank of 256 Kbytes of Flash. The Flash is mapped to address 0x0010 0000. It is also accessible at address 0x0 after the reset and before the Remap Command.

**Figure 8-2.** Internal Memory Mapping



## 8.3 Embedded Flash

### 8.3.1 Flash Overview

The Flash block of the AT91SAM7A3 is organized in 1024 pages of 256 bytes. It reads as 65,536 32-bit words.

The Flash block contains a 256-byte write buffer, accessible through a 32-bit interface.

When Flash is not used (read or write access), it is automatically put into standby mode.

### 8.3.2 Embedded Flash Controller

The Embedded Flash Controller (EFC) 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 within the Memory Controller on the APB. The User Interface allows:

- programming of the access parameters of the Flash (number of wait states, timings, etc.)
- starting commands such as full erase, page erase, page program, NVM bit set, NVM bit clear, etc.
- getting the end status of the last command
- getting error status
- programming interrupts on the end of the last commands or on errors

The Embedded Flash Controller also provides a dual 32-bit Prefetch Buffer that optimizes 16-bit access to the Flash. This is particularly efficient when the processor is running in Thumb mode.

### 8.3.3 Lock Regions

The Embedded Flash Controller manages 16 lock bits to protect 16 regions of the Flash against inadvertent Flash erasing or programming commands.

The AT91SAM7A3 has 16 lock regions. Each lock region contains 16 pages of 256 bytes. Each lock region has a size of 4 Kbytes, thus only the first 64 Kbytes can be locked.

The 16 NVM bits are software programmable through the EFC User Interface. The command “Set Lock Bit” activates the protection. The command “Clear Lock Bit” unlocks the lock region.

## 9. System Controller

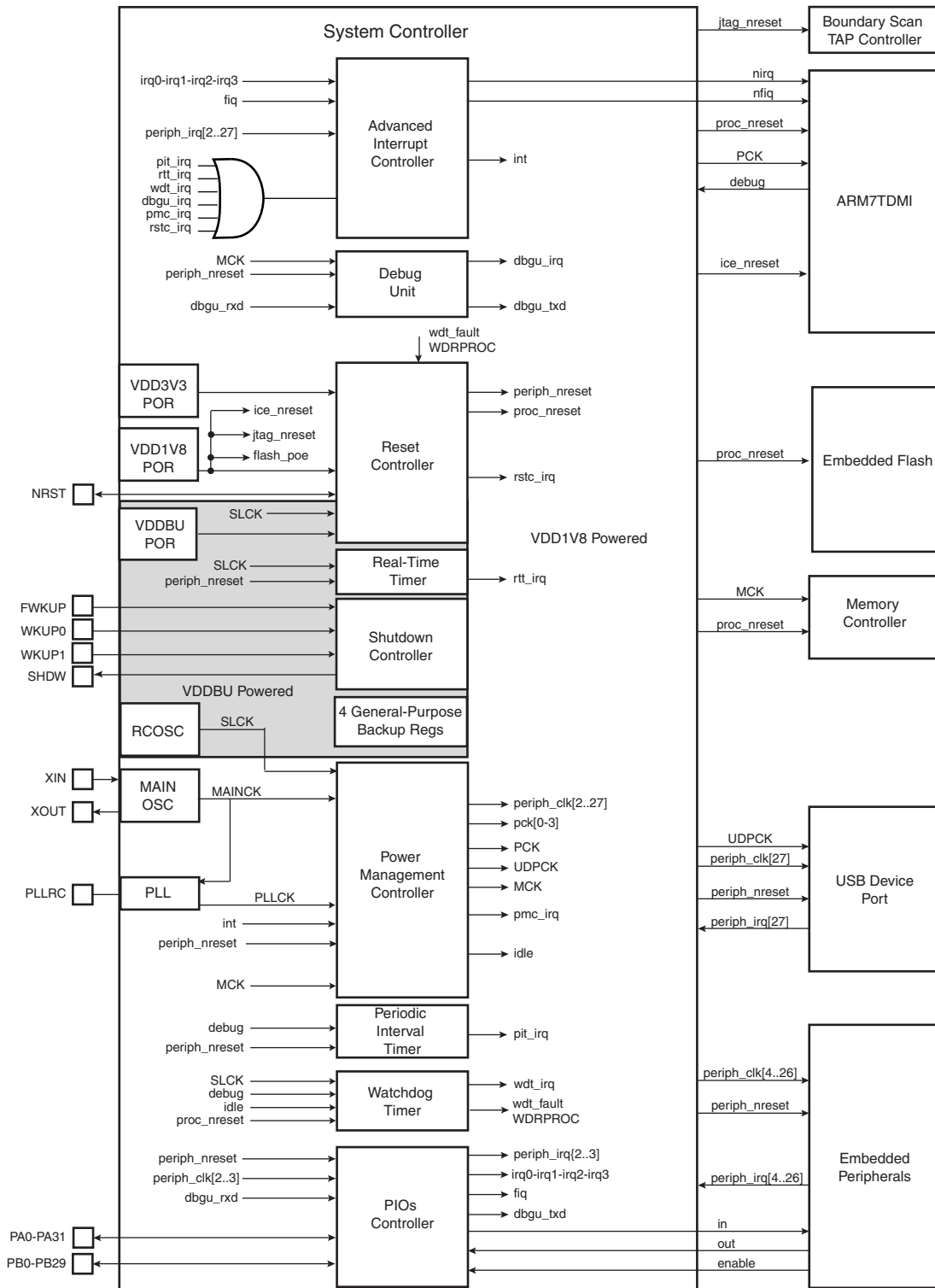
The System Controller manages all vital blocks of the microcontroller: interrupts, clocks, power, time, debug and reset.

The System Controller peripherals are all mapped to the highest 4K bytes of address space, between addresses 0xFFFF F000 and 0xFFFF FFFF. Each peripheral has an address space of up to 512 Bytes, representing up to 128 registers.

[Figure 9-1 on page 20](#) shows the System Controller Block Diagram.

[Figure 8-1 on page 16](#) shows the mapping of the User Interface of the System Controller peripherals. Note that the Memory Controller configuration user interface is also mapped within this address space.

**Figure 9-1. System Controller Block Diagram**



## 9.1 System Controller Mapping

## 9.2 Reset Controller

The Reset Controller is based on three power-on reset cells. It gives the status of the last reset, indicating whether it is a general reset, a wake-up reset, a software reset, a user reset or a watchdog reset. In addition, it controls the internal resets and the NRST pin output. It shapes a signal on the NRST line, guaranteeing that the length of the pulse meets any requirement.

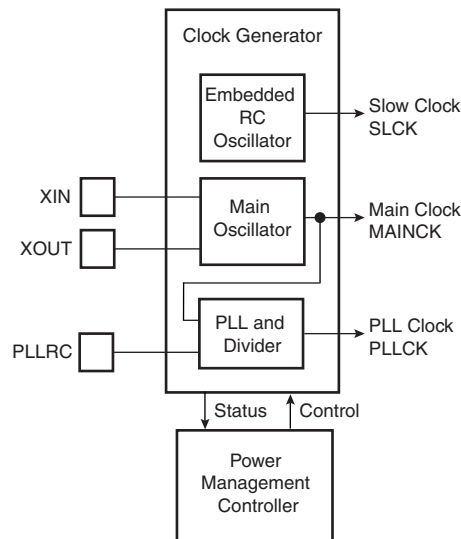
## 9.3 Clock Generator

The Clock Generator embeds one low-power RC Oscillator, one Main Oscillator and one PLL with the following characteristics:

- RC Oscillator ranges between 22 KHz and 42 KHz
- Main Oscillator frequency ranges between 3 and 20 MHz
- Main Oscillator can be bypassed
- PLL output ranges between 80 and 220 MHz

It provides SLCK, MAINCK and PLLCK.

**Figure 9-2.** Clock Generator Block Diagram



## 9.4 Power Management Controller

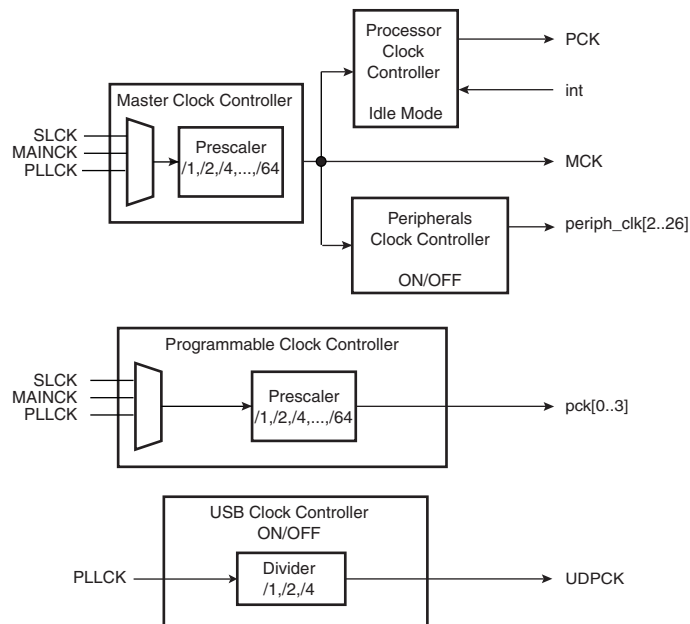
The Power Management Controller uses the Clock Generator outputs to provide:

- the Processor Clock PCK
- the Master Clock MCK
- the USB Clock UDPCK
- all the peripheral clocks, independently controllable
- four programmable clock outputs

The Master Clock (MCK) is programmable from a few hundred Hz to the maximum operating frequency of the device.

The Processor Clock (PCK) switches off when entering processor idle mode, thereby reducing power consumption while waiting an interrupt.

**Figure 9-3.** Power Management Controller Block Diagram



## 9.5 Advanced Interrupt Controller

- Controls the interrupt lines (nIRQ and nFIQ) of the ARM Processor
- Individually maskable and vectored interrupt sources
  - Source 0 is reserved for the Fast Interrupt Input (FIQ)
  - Source 1 is reserved for system peripherals (ST, PMC, DBGU, etc.)
  - Other sources control the peripheral interrupts or external interrupts
  - Programmable edge-triggered or level-sensitive internal sources
  - Programmable positive/negative edge-triggered or high/low level-sensitive external sources (FIQ, IRQ)
- 8-level Priority Controller
  - Drives the normal interrupt nIRQ of the processor
  - Handles priority of the interrupt sources
  - Higher priority interrupts can be served during service of a lower priority interrupt
- Vectoring
  - Optimizes interrupt service routine branch and execution
  - One 32-bit vector register per interrupt source
  - Interrupt vector register reads the corresponding current interrupt vector
- Protect Mode
  - Easy debugging by preventing automatic operations
- Fast Forcing
  - Permits redirecting any interrupt source on the fast interrupt
- General Interrupt Mask
  - Provides processor synchronization on events without triggering an interrupt

## 9.6 Debug Unit

- Comprises
  - One two-pin UART
  - One interface for the Debug Communication Channel (DCC) support
  - One set of chip ID registers
  - One interface allowing ICE access prevention
- Two-pin UART
  - USART-compatible user interface
  - Programmable baud rate generator
  - Parity, framing and overrun error
  - Automatic Echo, Local Loopback and Remote Loopback Channel Modes
- Debug Communication Channel Support
  - Offers visibility of COMMRX and COMMTX signals from the ARM Processor
- Chip ID Registers
  - Identification of the device revision, sizes of the embedded memories, set of peripherals
  - Chip ID is 0x260A0941 (Version 1)

## 9.7 Period Interval Timer

- 20-bit programmable counter plus 12-bit interval counter

## 9.8 Watchdog Timer

- 12-bit key-protected Programmable Counter running on prescaled SLCK
- Provides reset or interrupt signals to the system
- Counter may be stopped while the processor is in debug state or in idle mode

## 9.9 Real-time Timer

- 32-bit free-running counter with alarm
- Programmable 16-bit prescaler for SCLK accuracy compensation

## 9.10 Shutdown Controller

- Software programmable assertion of the SHDW open-drain pin
- De-assertion programmable with the pins WKUP0, WKUP1 and FWKUP

## 9.11 PIO Controllers A and B

- The PIO Controllers A and B respectively control 32 and 30 programmable I/O Lines
- Fully programmable through Set/Clear Registers
- Multiplexing of two 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
  - Half a clock period Glitch filter
  - Multi-drive option enables driving in open drain



- Programmable pull up on each I/O line
- Pin data status register, supplies visibility of the level on the pin at any time
- Synchronous output, provides Set and Clear of several I/O lines in a single write



## 10. Peripherals

### 10.1 Peripheral Mapping

Each User Peripheral is allocated 16K bytes of address space.

Figure 10-1. User Peripherals Mapping

Address	Peripheral	Peripheral Name	Size
0xF000 0000	Reserved		
0xFFFF7 FFFF 0xFFFF8 0000	CAN0	CAN Controller 0	16K Bytes
0xFFFF8 3FFF 0xFFFF8 4000	CAN1	CAN Controller 1	16K Bytes
0xFFFF8 7FFF 0xFFFF8 8000	Reserved		
0xFFFF9 FFFF 0xFFFFA 0000	TC0, TC1, TC2	Timer/Counter 0, 1 and 2	16K Bytes
0xFFFFA 3FFF 0xFFFFA 4000	TC3, TC4, TC5	Timer/Counter 3, 4 and 5	16K Bytes
0xFFFFA 7FFF 0xFFFFA 8000	TC6, TC7, TC8	Timer/Counter 6, 7 and 8	16K Bytes
0xFFFFA BFFF 0xFFFFA C000	MCI	Multimedia Card Interface	16K Bytes
0xFFFFA FFFF 0xFFFFB 0000	UDP	USB Device Port	16K Bytes
0xFFFFB 3FFF 0xFFFFB 4000	Reserved		
0xFFFFB 7FFF 0xFFFFB 8000	TWI	Two-Wire Interface	16K Bytes
0xFFFFB BFFF 0xFFFFB C000	Reserved		
0xFFFFB FFFF 0xFFFFC 0000	USART0	Universal Synchronous Asynchronous Receiver Transmitter 0	16K Bytes
0xFFFFC 3FFF 0xFFFFC 4000	USART1	Universal Synchronous Asynchronous Receiver Transmitter 1	16K Bytes
0xFFFFC 7FFF 0xFFFFC 8000	USART2	Universal Synchronous Asynchronous Receiver Transmitter 1	16K Bytes
0xFFFFC BFFF 0xFFFFC C000	PWMC	PWM Controller	16K Bytes
0xFFFFC FFFF 0xFFFFD 0000	SSC0	Serial Synchronous Controller 0	16K Bytes
0xFFFFD 3FFF 0xFFFFD 4000	SSC1	Serial Synchronous Controller 1	16K Bytes
0xFFFFD 7FFF 0xFFFFD 8000	ADC0	Analog-to-Digital Converter 0	16K Bytes
0xFFFFD BFFF 0xFFFFD C000	ADC1	Analog-to-Digital Converter 1	16K Bytes
0xFFFFD FFFF 0xFFFFE 0000	SPI0	Serial Peripheral Interface 0	16K Bytes
0xFFFFE 3FFF 0xFFFFE 4000	SPI1	Serial Peripheral Interface 1	16K Bytes
0xFFFFE 7FFF 0xFFFFE 8000	Reserved		
0xFFFFE FFFF	Reserved		

## 10.2 Peripheral Multiplexing on PIO Lines

The AT91SAM7A3 features two PIO controllers, PIOA and PIOB, which multiplex the I/O lines of the peripheral set.

PIO Controllers A and B control respectively 32 and 30 lines. Each line can be assigned to one of two peripheral functions, A or B. Some of them can also be multiplexed with Analog Input of both ADC Controllers.

[Table 10-1 on page 27](#) and [Table 10-2 on page 28](#) define how the I/O lines of the peripherals A, B or Analog Input are multiplexed on the PIO Controllers A and B. The two columns “Function” and “Comments” have been inserted for the user’s own comments; they may be used to track how pins are defined in an application.

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

At reset, all I/O lines are automatically configured as input with the programmable pull-up enabled, so that the device is maintained in a static state as soon as a reset occurs.

## 10.3 PIO Controller A Multiplexing

**Table 10-1.** Multiplexing on PIO Controller A

PIO Controller A				Application Usage	
I/O Line	Peripheral A	Peripheral B	Comment	Function	Comments
PA0	TWD	ADC0_ADTRG			
PA1	TWCK	ADC1_ADTRG			
PA2	RXD0				
PA3	TXD0				
PA4	SCK0	SPI1_NPSC0			
PA5	RTS0	SPI1_NPCS1			
PA6	CTS0	SPI1_NPCS2			
PA7	RXD1	SPI1_NPCS3			
PA8	TXD1	SPI1_MISO			
PA9	RXD2	SPI1_MOSI			
PA10	TXD2	SPI1_SPCK			
PA11	SPI0_NPCS0				
PA12	SPI0_NPCS1	MCDA1			
PA13	SPI0_NPCS2	MCDA2			
PA14	SPI0_NPCS3	MCDA3			
PA15	SPI0_MISO	MCDA0			
PA16	SPI0_MOSI	MCCDA			
PA17	SPI0_SPCK	MCCK			
PA18	PWM0	PCK0			
PA19	PWM1	PCK1			
PA20	PWM2	PCK2			
PA21	PWM3	PCK3			
PA22	PWM4	IRQ0			
PA23	PWM5	IRQ1			
PA24	PWM6	TCLK4			
PA25	PWM7	TCLK5			
PA26	CANRX0				
PA27	CANTX0				
PA28	CANRX1	TCLK3			
PA29	CANTX1	TCLK6			
PA30	DRXD	TCLK7			
PA31	DTXD	TCLK8			



## 10.4 PIO Controller B Multiplexing

Table 10-2. Multiplexing on PIO Controller B

PIO Controller B				Application Usage	
I/O Line	Peripheral A	Peripheral B	Comment	Function	Comments
PB0	IRQ2	PWM5			
PB1	IRQ3	PWM6			
PB2	TF0	PWM7			
PB3	TK0	PCK0			
PB4	TD0	PCK1			
PB5	RD0	PCK2			
PB6	RK0	PCK3			
PB7	RF0	CANTX1			
PB8	FIQ	TF1			
PB9	TCLK0	TK1			
PB10	TCLK1	RK1			
PB11	TCLK2	RF1			
PB12	TIOA0	TD1			
PB13	TIOB0	RD1			
PB14	TIOA1	PWM0	ADC0_AD0		
PB15	TIOB1	PWM1	ADC0_AD1		
PB16	TIOA2	PWM2	ADC0_AD2		
PB17	TIOB2	PWM3	ADC0_AD3		
PB18	TIOA3	PWM4	ADC0_AD4		
PB19	TIOB3	SPI1_NPCS1	ADC0_AD5		
PB20	TIOA4	SPI1_NPCS2	ADC0_AD6		
PB21	TIOB4	SPI1_NPCS3	ADC0_AD7		
PB22	TIOA5		ADC1_AD0		
PB23	TIOB5		ADC1_AD1		
PB24	TIOA6	RTS1	ADC1_AD2		
PB25	TIOB6	CTS1	ADC1_AD3		
PB26	TIOA7	SCK1	ADC1_AD4		
PB27	TIOB7	RTS2	ADC1_AD5		
PB28	TIOA8	CTS2	ADC1_AD6		
PB29	TIOB8	SCK2	ADC1_AD7		

## 11. Peripheral Identifiers

The AT91SAM7A3 embeds a wide range of peripherals. [Table 11-1](#) defines the Peripheral Identifiers of the AT91SAM7A3. Unique peripheral identifiers are defined for both the AIC and the PMC.

**Table 11-1.** Peripheral Identifiers

Peripheral ID	Peripheral Mnemonic	Peripheral Name	External Interrupt
0	AIC	Advanced Interrupt Controller	FIQ
1	SYSC <sup>(1)</sup>		
2	PIOA	Parallel I/O Controller A	
3	PIOB	Parallel I/O Controller B	
4	CAN0	CAN Controller 0	
5	CAN1	CAN Controller 1	
6	US0	USART 0	
7	US1	USART 1	
8	US2	USART 2	
9	MCI	Multimedia Card Interface	
10	TWI	Two-wire Interface	
11	SPI0	Serial Peripheral Interface 0	
12	SPI1	Serial Peripheral Interface 1	
13	SSC0	Synchronous Serial Controller 0	
14	SSC1	Synchronous Serial Controller 1	
15	TC0	Timer/Counter 0	
16	TC1	Timer/Counter 1	
17	TC2	Timer/Counter 2	
18	TC3	Timer/Counter 3	
19	TC4	Timer/Counter 4	
20	TC5	Timer/Counter 5	
21	TC6	Timer/Counter 6	
22	TC7	Timer/Counter 7	
23	TC8	Timer/Counter 8	
24	ADC0 <sup>(1)</sup>	Analog-to Digital Converter 0	
25	ADC1 <sup>(1)</sup>	Analog-to Digital Converter 1	
26	PWMC	PWM Controller	
27	UDP	USB Device Port	
28	AIC	Advanced Interrupt Controller	IRQ0
29	AIC	Advanced Interrupt Controller	IRQ1
30	AIC	Advanced Interrupt Controller	IRQ2
31	AIC	Advanced Interrupt Controller	IRQ3

Note: 1. Setting SYSC and ADC bits in the clock set/clear registers of the PMC has no effect. The System Controller and ADC are continuously clocked.

## 11.1 Serial Peripheral Interface

- Supports communication with external serial devices
  - Four chip selects with external decoder allow communication with up to 15 peripherals
  - Serial memories, such as DataFlash<sup>®</sup> 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 per chip select between consecutive transfers and between clock and data
  - Programmable delay between consecutive transfers
  - Selectable mode fault detection
  - Maximum frequency at up to Master Clock

## 11.2 Two-wire Interface

- Master Mode only
- Compatibility with standard two-wire serial memories
- One, two or three bytes for slave address
- Sequential read/write operations

## 11.3 USART

- 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
  - Hardware handshaking RTS-CTS
  - Receiver time-out and transmitter timeguard
  - Optional Multi-drop Mode with address generation and detection
- RS485 with driver control signal
- ISO7816, T = 0 or T = 1 Protocols for interfacing with smart cards
  - NACK handling, error counter with repetition and iteration limit
- IrDA modulation and demodulation
  - Communication at up to 115.2 Kbps
- Test Modes

- Remote Loopback, Local Loopback, Automatic Echo

## 11.4 Serial Synchronous Controller

- Provides serial synchronous communication links used in audio and telecom applications
- Contains an independent receiver and transmitter and a common clock divider
- Offers a configurable frame sync and data length
- Receiver and transmitter can be programmed to start automatically or on detection of different event on the frame sync signal
- Receiver and transmitter include a data signal, a clock signal and a frame synchronization signal

## 11.5 Timer Counter

- Three 16-bit Timer Counter Channels
- Wide range of functions including:
  - Frequency Measurement
  - Event Counting
  - Interval Measurement
  - Pulse Generation
  - Delay Timing
  - Pulse Width Modulation
  - Up/down Capabilities
- Each channel is user-configurable and contains:
  - Three external clock inputs
  - Five internal clock inputs as defined in [Table 11-2](#).

**Table 11-2.** Timer Counter Clock Assignment

TC Clock input	Clock
TIMER_CLOCK1	MCK/2
TIMER_CLOCK2	MCK/8
TIMER_CLOCK3	MCK/32
TIMER_CLOCK4	MCK/128
TIMER_CLOCK5	MCK/1024

- Two multi-purpose input/output signals
- Two global registers that act on all three TC Channels

## 11.6 PWM Controller

- Eight channels, one 20-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 channel programming

- Independent enable/disable commands
- Independent clock selection
- Independent period and duty cycle, with double buffering
- Programmable selection of the output waveform polarity
- Programmable center or left aligned output waveform

## 11.7 USB Device Port

- USB V2.0 full-speed compliant, 12 Mbits per second.
- Embedded USB V2.0 full-speed transceiver
- Six endpoints
  - Endpoint 0: 8 bytes
  - Endpoint 1 and 2: 64 bytes ping-pong
  - Endpoint 3: 64 bytes
  - Endpoint 4 and 5: 512 bytes ping-pong
- Embedded 2,376-byte dual-port RAM for endpoints
  - Ping-pong Mode (two memory banks) for bulk endpoints
- Suspend/resume logic

## 11.8 Multimedia Card Interface

- Compatibility with MultiMedia card specification version 2.2
- Compatibility with SD Memory card specification version 1.0
- Cards clock rate up to Master Clock divided by 2
- Embeds power management to slow down clock rate when not used
- Supports up to sixteen slots (through multiplexing)
  - One slot for one MultiMedia card bus (up to 30 cards) or one SD memory card
- Supports stream, block and multi-block data read and write
- Supports connection to Peripheral Data Controller
  - Minimizes processor intervention for large buffer transfers

## 11.9 CAN Controller

- Fully compliant with CAN 2.0B active controllers
- Bit rates up to 1Mbit/s
- 16 object-oriented mailboxes, each with the following properties:
  - CAN specification 2.0 Part A or 2.0 Part B programmable for each message
  - Object-configurable as receive (with overwrite or not) or transmit
  - Local tag and mask filters up to 29-bit identifier/channel
  - 32-bit access to data registers for each mailbox data object
  - Uses a 16-bit time stamp on receive and transmit messages
  - Hardware concatenation of ID unmasked bit fields to speed up family ID processing
  - 16-bit internal timer for Time Stamping and Network synchronization



- Programmable reception buffer length up to 16 mailbox object
- Priority management between transmission mailboxes
- Autobaud and listening mode
- Low power mode and programmable wake-up on bus activity or by the application
- Data, remote, error and overload frame handling

## 11.10 Analog-to-Digital Converter

- 8-channel ADC
- 10-bit 384K, or 8-bit 533K, samples/sec Successive Approximation Register ADC
- -3/+3 LSB Integral Non Linearity, -2/+2 LSB Differential Non Linearity
- Integrated 8-to-1 multiplexer, offering eight independent 3.3V analog inputs
- Individual enable and disable of each channel
- External voltage reference for better accuracy on low-voltage inputs
- Multiple trigger sources
  - Hardware or software trigger
  - External pins: ADTRG0 and ADTRG1
  - Timer Counter 0 to 5 outputs: TIOA0 to TIOA5
- Sleep Mode and conversion sequencer
  - Automatic wakeup on trigger and back to sleep mode after conversions of all enabled channels
- All analog inputs are shared with digital signals



## 12. ARM7TDMI Processor

### 12.1 Overview

The ARM7TDMI core executes both the 32-bit ARM and 16-bit Thumb instruction sets, allowing the user to trade off between high performance and high code density. The ARM7TDMI processor implements Von Neuman architecture, using a three-stage pipeline consisting of Fetch, Decode, and Execute stages.

The main features of the ARM7TDMI processor are:

- ARM7TDMI Based on ARMv4T Architecture
- Two Instruction Sets
  - ARM High-performance 32-bit Instruction Set
  - Thumb High Code Density 16-bit Instruction Set
- Three-Stage Pipeline Architecture
  - Instruction Fetch (F)
  - Instruction Decode (D)
  - Execute (E)

### 12.2 ARM7TDMI Processor

For further details on ARM7TDMI, refer to the following ARM documents:

ARM Architecture Reference Manual (DDI 0100E)

ARM7TDMI Technical Reference Manual (DDI 0210B)

#### 12.2.1 Instruction Type

Instructions are either 32 bits long (in ARM state) or 16 bits long (in THUMB state).

#### 12.2.2 Data Type

ARM7TDMI supports byte (8-bit), half-word (16-bit) and word (32-bit) data types. Words must be aligned to four-byte boundaries and half words to two-byte boundaries.

Unaligned data access behavior depends on which instruction is used where.

#### 12.2.3 ARM7TDMI Operating Mode

The ARM7TDMI, based on ARM architecture v4T, supports seven processor modes:

**User:** The normal ARM program execution state

**FIQ:** Designed to support high-speed data transfer or channel process

**IRQ:** Used for general-purpose interrupt handling

**Supervisor:** Protected mode for the operating system

**Abort mode:** Implements virtual memory and/or memory protection

**System:** A privileged user mode for the operating system

**Undefined:** Supports software emulation of hardware coprocessors

Mode changes may be made under software control, or may be brought about by external interrupts or exception processing. Most application programs execute in User mode. The



non-user modes, or privileged modes, are entered in order to service interrupts or exceptions, or to access protected resources.

### 12.2.4 ARM7TDMI Registers

The ARM7TDMI processor has a total of 37 registers:

- 31 general-purpose 32-bit registers
- 6 status registers

These registers are not accessible at the same time. The processor state and operating mode determine which registers are available to the programmer.

At any one time 16 registers are visible to the user. The remainder are synonyms used to speed up exception processing.

Register 15 is the Program Counter (PC) and can be used in all instructions to reference data relative to the current instruction.


R14 holds the return address after a subroutine call.

R13 is used (by software convention) as a stack pointer

**Table 12-1.** ARM7TDMI ARM Modes and Registers Layout

User and System Mode	Supervisor Mode	Abort Mode	Undefined Mode	Interrupt Mode	Fast Interrupt Mode
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8_FIQ
R9	R9	R9	R9	R9	R9_FIQ
R10	R10	R10	R10	R10	R10_FIQ
R11	R11	R11	R11	R11	R11_FIQ
R12	R12	R12	R12	R12	R12_FIQ
R13	R13_SVC	R13_ABORT	R13_UNDEF	R13_IRQ	R13_FIQ
R14	R14_SVC	R14_ABORT	R14_UNDEF	R14_IRQ	R14_FIQ
PC	PC	PC	PC	PC	PC

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_SVC	SPSR_ABORT	SPSR_UNDEF	SPSR_IRQ	SPSR_FIQ

 Mode-specific banked registers

Registers R0 to R7 are unbanked registers. This means that each of them refers to the same 32-bit physical register in all processor modes. They are general-purpose registers, with no special uses managed by the architecture, and can be used wherever an instruction allows a general-purpose register to be specified.

Registers R8 to R14 are banked registers. This means that each of them depends on the current mode of the processor.

## 12.2.4.1 Modes and Exception Handling

All exceptions have banked registers for R14 and R13.

After an exception, R14 holds the return address for exception processing. This address is used to return after the exception is processed, as well as to address the instruction that caused the exception.

R13 is banked across exception modes to provide each exception handler with a private stack pointer.

The fast interrupt mode also banks registers 8 to 12 so that interrupt processing can begin without having to save these registers.

A seventh processing mode, System Mode, does not have any banked registers. It uses the User Mode registers. System Mode runs tasks that require a privileged processor mode and allows them to invoke all classes of exceptions.

## 12.2.4.2 Status Registers

All other processor states are held in status registers. The current operating processor status is in the Current Program Status Register (CPSR). The CPSR holds:

- four ALU flags (Negative, Zero, Carry, and Overflow)
- two interrupt disable bits (one for each type of interrupt)
- one bit to indicate ARM or Thumb execution
- five bits to encode the current processor mode

All five exception modes also have a Saved Program Status Register (SPSR) that holds the CPSR of the task immediately preceding the exception.

## 12.2.4.3 Exception Types

The ARM7TDMI supports five types of exception and a privileged processing mode for each type. The types of exceptions are:

- fast interrupt (FIQ)
- normal interrupt (IRQ)
- memory aborts (used to implement memory protection or virtual memory)
- attempted execution of an undefined instruction
- software interrupts (SWIs)

Exceptions are generated by internal and external sources.

More than one exception can occur in the same time.

When an exception occurs, the banked version of R14 and the SPSR for the exception mode are used to save state.



To return after handling the exception, the SPSR is moved to the CPSR, and R14 is moved to the PC. This can be done in two ways:

- by using a data-processing instruction with the S-bit set, and the PC as the destination
- by using the Load Multiple with Restore CPSR instruction (LDM)

## 12.2.5 ARM Instruction Set Overview

The ARM instruction set is divided into:

- Branch instructions
- Data processing instructions
- Status register transfer instructions
- Load and Store instructions
- Coprocessor instructions
- Exception-generating instructions

ARM instructions can be executed conditionally. Every instruction contains a 4-bit condition code field (bit[31:28]).

Table 12-2 gives the ARM instruction mnemonic list.

**Table 12-2.** ARM Instruction Mnemonic List

Mnemonic	Operation
MOV	Move
ADD	Add
SUB	Subtract
RSB	Reverse Subtract
CMP	Compare
TST	Test
AND	Logical AND
EOR	Logical Exclusive OR
MUL	Multiply
SMULL	Sign Long Multiply
SMLAL	Signed Long Multiply Accumulate
MSR	Move to Status Register
B	Branch
BX	Branch and Exchange
LDR	Load Word
LDRSH	Load Signed Halfword
LDRSB	Load Signed Byte
LDRH	Load Half Word
LDRB	Load Byte
LDRBT	Load Register Byte with Translation
LDRT	Load Register with Translation
LDM	Load Multiple
SWP	Swap Word
MCR	Move To Coprocessor
LDC	Load To Coprocessor

Mnemonic	Operation
CDP	Coprocessor Data Processing
MVN	Move Not
ADC	Add with Carry
SBC	Subtract with Carry
RSC	Reverse Subtract with Carry
CMN	Compare Negated
TEQ	Test Equivalence
BIC	Bit Clear
ORR	Logical (inclusive) OR
MLA	Multiply Accumulate
UMULL	Unsigned Long Multiply
UMLAL	Unsigned Long Multiply Accumulate
MRS	Move From Status Register
BL	Branch and Link
SWI	Software Interrupt
STR	Store Word
STRH	Store Half Word
STRB	Store Byte
STRBT	Store Register Byte with Translation
STRT	Store Register with Translation
STM	Store Multiple
SWPB	Swap Byte
MRC	Move From Coprocessor
STC	Store From Coprocessor

## 12.2.6 Thumb Instruction Set Overview

The Thumb instruction set is a re-encoded subset of the ARM instruction set.

The Thumb instruction set is divided into:

- Branch instructions
- Data processing instructions
- Load and Store instructions
- Load and Store Multiple instructions
- Exception-generating instruction

In Thumb mode, eight general-purpose registers, R0 to R7, are available that are the same physical registers as R0 to R7 when executing ARM instructions. Some Thumb instructions also access to the Program Counter (ARM Register 15), the Link Register (ARM Register 14) and the Stack Pointer (ARM Register 13). Further instructions allow limited access to the ARM registers 8 to 15.

Table 12-3 gives the Thumb instruction mnemonic list.

**Table 12-3.** Thumb Instruction Mnemonic List

Mnemonic	Operation	Mnemonic	Operation
MOV	Move	MVN	Move Not
ADD	Add	ADC	Add with Carry
SUB	Subtract	SBC	Subtract with Carry
CMP	Compare	CMN	Compare Negated
TST	Test	NEG	Negate
AND	Logical AND	BIC	Bit Clear
EOR	Logical Exclusive OR	ORR	Logical (inclusive) OR
LSL	Logical Shift Left	LSR	Logical Shift Right
ASR	Arithmetic Shift Right	ROR	Rotate Right
MUL	Multiply		
B	Branch	BL	Branch and Link
BX	Branch and Exchange	SWI	Software Interrupt
LDR	Load Word	STR	Store Word
LDRH	Load Half Word	STRH	Store Half Word
LDRB	Load Byte	STRB	Store Byte
LDRSH	Load Signed Halfword	LDRSB	Load Signed Byte
LDMIA	Load Multiple	STMIA	Store Multiple
PUSH	Push Register to stack	POP	Pop Register from stack



## 13. AT91SAM7A3 Debug and Test Features

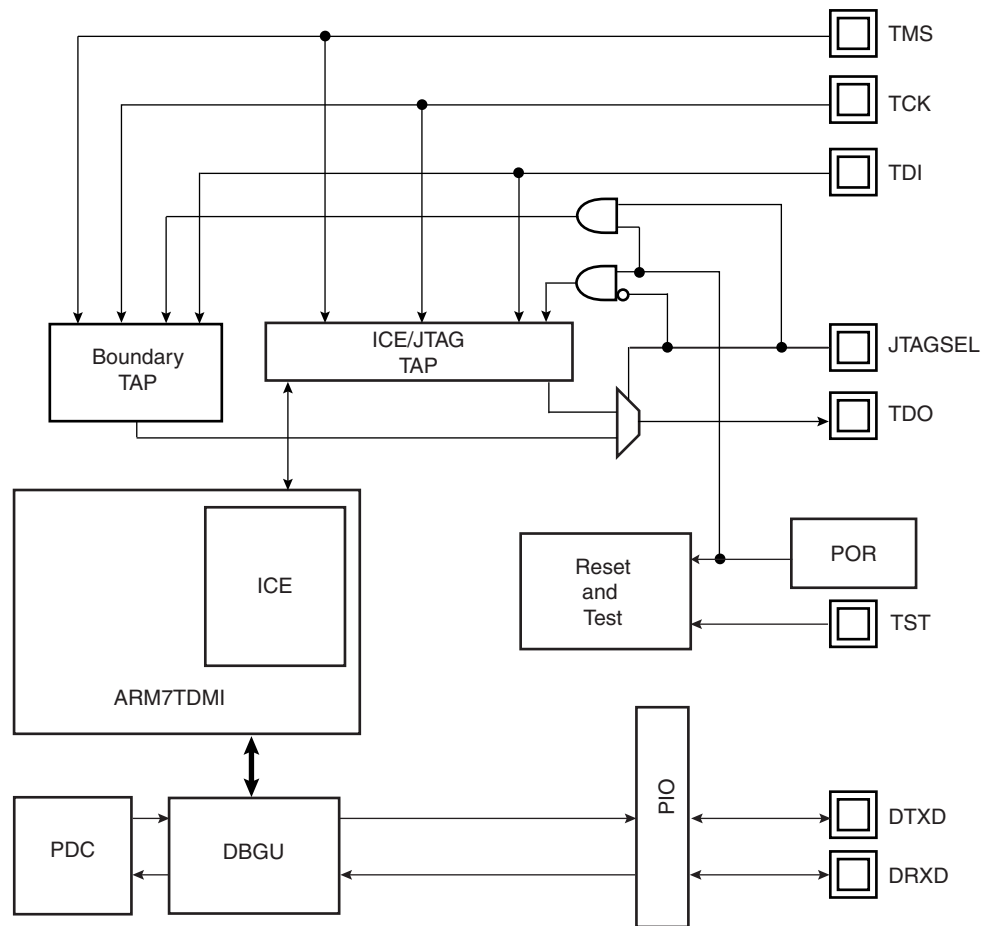
### 13.1 Overview

The AT91SAM7A3 features a number of complementary debug and test capabilities. A common JTAG/ICE (Embedded ICE) port is used for standard debugging functions, such as downloading code and single-stepping through programs. The Debug Unit provides a two-pin UART that can be used to upload an application into internal SRAM. It manages the interrupt handling of the internal COMMTX and COMMRX signals that trace the activity of the Debug Communication Channel.

A set of dedicated debug and test input/output pins gives direct access to these capabilities from a PC-based test environment.

### 13.2 Block Diagram

Figure 13-1. Debug and Test Block Diagram

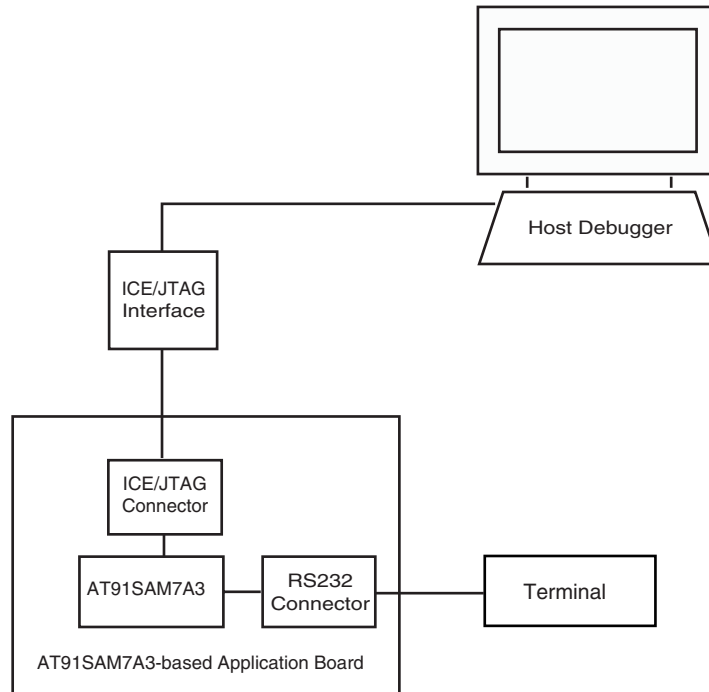


## 13.3 Application Examples

### 13.3.1 Debug Environment

Figure 13-2 on page 42 shows a complete debug environment example. The ICE/JTAG interface is used for standard debugging functions, such as downloading code and single-stepping through the program.

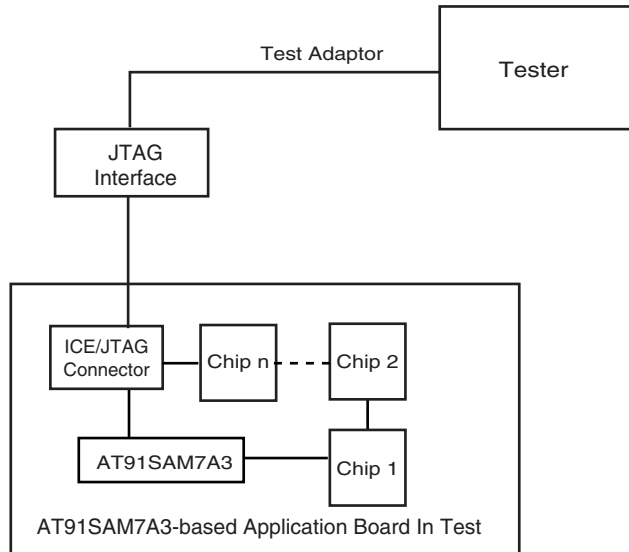
**Figure 13-2.** Application Debug Environment Example



## 13.4 Test Environment

Figure 13-3 on page 43 shows a test environment example. Test vectors are sent and interpreted by the tester. In this example, the “board in test” is designed using a number of JTAG-compliant devices. These devices can be connected to form a single scan chain.

**Figure 13-3.** Application Test Environment Example



## 13.5 Debug and Test Pin Description

**Table 13-1.** Debug and Test Pin List

Pin Name	Function	Type	Active Level
<b>Reset/Test</b>			
NRST	Microcontroller Reset	Input/Output	Low
TST	Test Mode Select	Input	High
<b>ICE and JTAG</b>			
TCK	Test Clock	Input	
TDI	Test Data In	Input	
TDO	Test Data Out	Output	
TMS	Test Mode Select	Input	
JTAGSEL	JTAG Selection	Input	
<b>Debug Unit</b>			
DRXD	Debug Receive Data	Input	
DTXD	Debug Transmit Data	Output	

## 13.6 Functional Description

### 13.6.1 Test Pin

One dedicated pin, TST, is used to define the device operating mode. The user must make sure that this pin is tied at low level to ensure normal operating conditions. Other values associated with this pin are reserved for manufacturing test.

### 13.6.2 Embedded ICE™ (Embedded In-circuit Emulator)

The ARM7TDMI Embedded ICE is supported via the ICE/JTAG port. The internal state of the ARM7TDMI is examined through an ICE/JTAG port.

The ARM7TDMI processor contains hardware extensions for advanced debugging features:

- In halt mode, a store-multiple (STM) can be inserted into the instruction pipeline. This exports the contents of the ARM7TDMI registers. This data can be serially shifted out without affecting the rest of the system.
- In monitor mode, the JTAG interface is used to transfer data between the debugger and a simple monitor program running on the ARM7TDMI processor.

There are three scan chains inside the ARM7TDMI processor that support testing, debugging, and programming of the Embedded ICE. The scan chains are controlled by the ICE/JTAG port.

Embedded ICE mode is selected when JTAGSEL is low. It is not possible to switch directly between ICE and JTAG operations. A chip reset must be performed after JTAGSEL is changed.

For further details on the Embedded ICE, see the ARM7TDMI (Rev4) Technical Reference Manual (DDI0210B).

### 13.6.3 Debug Unit

The Debug Unit provides a two-pin (DXRD and TXRD) USART that can be used for several debug and trace purposes and offers an ideal means for in-situ programming solutions and debug monitor communication. Moreover, the association with two peripheral data controller channels permits packet handling of these tasks with processor time reduced to a minimum.

The Debug Unit also manages the interrupt handling of the COMMTX and COMMRX signals that come from the ICE and that trace the activity of the Debug Communication Channel. The Debug Unit allows blockage of access to the system through the ICE interface.

The Debug Unit can be used to upload an application into the internal SRAM. It is activated by the boot program when no valid application is detected. The protocol used to load the application is XMODEM.

A specific register, the Debug Unit Chip ID Register, gives information about the product version and its internal configuration.

The AT91SAM7A3 Debug Unit Chip ID value is 0x260A0941 on 32-bit width.

For further details on the Debug Unit, see the Debug Unit section.

## 13.6.4 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 JTAGSEL is high. The SAMPLE, EXTEST and BYPASS functions are implemented. In ICE debug mode, the ARM processor responds with a non-JTAG chip ID that identifies the processor to the ICE system. This is not IEEE 1149.1 JTAG-compliant.

It is not possible to switch directly between JTAG and ICE operations. A chip reset must be performed after JTAGSEL is changed.

A Boundary-scan Descriptor Language (BSDL) file is provided to set up test.

### 13.6.4.1 JTAG Boundary-scan Register

The Boundary-scan Register (BSR) contains 186 bits that correspond to active pins and associated control signals.

Each AT91SAM7A3 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.

**Table 13-2.** AT91SAM7A3 JTAG Boundary Scan Register

Bit Number	Pin Name	Pin Type	Associated BSR Cells
185	PB13	IN/OUT	INPUT
184			OUTPUT
183			CONTROL
182	PB12	IN/OUT	INPUT
181			OUTPUT
180			CONTROL
179	PB11	IN/OUT	INPUT
178			OUTPUT
177			CONTROL
176	PB10	IN/OUT	INPUT
175			OUTPUT
174			CONTROL
173	PB9	IN/OUT	INPUT
172			OUTPUT
171			CONTROL
170	PB8	IN/OUT	INPUT
169			OUTPUT
168			CONTROL

**Table 13-2.** AT91SAM7A3 JTAG Boundary Scan Register (Continued)

Bit Number	Pin Name	Pin Type	Associated BSR Cells
167	PB7	IN/OUT	INPUT
166			OUTPUT
165			CONTROL
164	PB6	IN/OUT	INPUT
163			OUTPUT
162			CONTROL
161	PB5	IN/OUT	INPUT
160			OUTPUT
159			CONTROL
158	PB4	IN/OUT	INPUT
157			OUTPUT
156			CONTROL
155	PB3	IN/OUT	INPUT
154			OUTPUT
153			CONTROL
152	PB2	IN/OUT	INPUT
151			OUTPUT
150			CONTROL
149	PB1	IN/OUT	INPUT
148			OUTPUT
147			CONTROL
146	PB0	IN/OUT	INPUT
145			OUTPUT
144			CONTROL
143	PA0	IN/OUT	INPUT
142			OUTPUT
141			CONTROL
140	PA1	IN/OUT	INPUT
139			OUTPUT
138			CONTROL
137	PA2	IN/OUT	INPUT
136			OUTPUT
135			CONTROL

**Table 13-2.** AT91SAM7A3 JTAG Boundary Scan Register (Continued)

Bit Number	Pin Name	Pin Type	Associated BSR Cells
134	PA3	IN/OUT	INPUT
133			OUTPUT
132			CONTROL
131	PA4	IN/OUT	INPUT
130			OUTPUT
129			CONTROL
128	PA5	IN/OUT	INPUT
127			OUTPUT
126			CONTROL
125	PA6	IN/OUT	INPUT
124			OUTPUT
123			CONTROL
122	PA7	IN/OUT	INPUT
121			OUTPUT
120			CONTROL
119	PA8	IN/OUT	INPUT
118			OUTPUT
117			CONTROL
116	PA9	IN/OUT	INPUT
115			OUTPUT
114			CONTROL
113	PA10	IN/OUT	INPUT
112			OUTPUT
111			CONTROL
110	PA11	IN/OUT	INPUT
109			OUTPUT
108			CONTROL
107	PA12	IN/OUT	INPUT
106			OUTPUT
105			CONTROL
104	PA13	IN/OUT	INPUT
103			OUTPUT
102			CONTROL

**Table 13-2. AT91SAM7A3 JTAG Boundary Scan Register (Continued)**

Bit Number	Pin Name	Pin Type	Associated BSR Cells
101	PA14	IN/OUT	INPUT
100			OUTPUT
99			CONTROL
98	PA15	IN/OUT	INPUT
97			OUTPUT
96			CONTROL
95	PA16	IN/OUT	INPUT
94			OUTPUT
93			CONTROL
92	PA17	IN/OUT	INPUT
91			OUTPUT
90			CONTROL
89	PA18	IN/OUT	INPUT
88			OUTPUT
87			CONTROL
86	PA19	IN/OUT	INPUT
85			OUTPUT
84			CONTROL
83	PA20	IN/OUT	INPUT
82			OUTPUT
81			CONTROL
80	PA21	IN/OUT	INPUT
79			OUTPUT
78			CONTROL
77	PA22	IN/OUT	INPUT
76			OUTPUT
75			CONTROL
74	PA23	IN/OUT	INPUT
73			OUTPUT
72			CONTROL
71	PA24	IN/OUT	INPUT
70			OUTPUT
69			CONTROL



**Table 13-2.** AT91SAM7A3 JTAG Boundary Scan Register (Continued)

Bit Number	Pin Name	Pin Type	Associated BSR Cells
68	PA25	IN/OUT	INPUT
67			OUTPUT
66			CONTROL
65	PA26	IN/OUT	INPUT
64			OUTPUT
63			CONTROL
62	PA27	IN/OUT	INPUT
61			OUTPUT
60			CONTROL
59	PA28	IN/OUT	INPUT
58			OUTPUT
57			CONTROL
56	PA29	IN/OUT	INPUT
55			OUTPUT
54			CONTROL
53	PA30	IN/OUT	INPUT
52			OUTPUT
51			CONTROL
50	PA31	IN/OUT	INPUT
49			OUTPUT
48			CONTROL
47	PB14	IN/OUT	INPUT
46			OUTPUT
45			CONTROL
44	PB15	IN/OUT	INPUT
43			OUTPUT
42			CONTROL
41	PB16	IN/OUT	INPUT
40			OUTPUT
39			CONTROL
38	PB17	IN/OUT	INPUT
37			OUTPUT
36			CONTROL



**Table 13-2. AT91SAM7A3 JTAG Boundary Scan Register (Continued)**

Bit Number	Pin Name	Pin Type	Associated BSR Cells
35	PB18	IN/OUT	INPUT
34			OUTPUT
33			CONTROL
32	PB19	IN/OUT	INPUT
31			OUTPUT
30			CONTROL
29	PB20	IN/OUT	INPUT
28			OUTPUT
27			CONTROL
26	PB21	IN/OUT	INPUT
25			OUTPUT
24			CONTROL
23	PB22	IN/OUT	INPUT
22			OUTPUT
21			CONTROL
20	PB23	IN/OUT	INPUT
19			OUTPUT
18			CONTROL
17	PB24	IN/OUT	INPUT
16			OUTPUT
15			CONTROL
14	PB25	IN/OUT	INPUT
13			OUTPUT
12			CONTROL
11	PB26	IN/OUT	INPUT
10			OUTPUT
9			CONTROL
8	PB27	IN/OUT	INPUT
7			OUTPUT
6			CONTROL
5	PB28	IN/OUT	INPUT
4			OUTPUT
3			CONTROL

**Table 13-2.** AT91SAM7A3 JTAG Boundary Scan Register (Continued)

Bit Number	Pin Name	Pin Type	Associated BSR Cells
2	PB29	IN/OUT	INPUT
1			OUTPUT
0			CONTROL



### 13.6.5 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 0x1.

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

Product part Number is 0x5B05

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

Set to 0x01F.

Bit[0] Required by IEEE Std. 1149.1.

Set to 0x1.

JTAG ID Code value is 0x05B0503F

## 14. Reset Controller (RSTC)

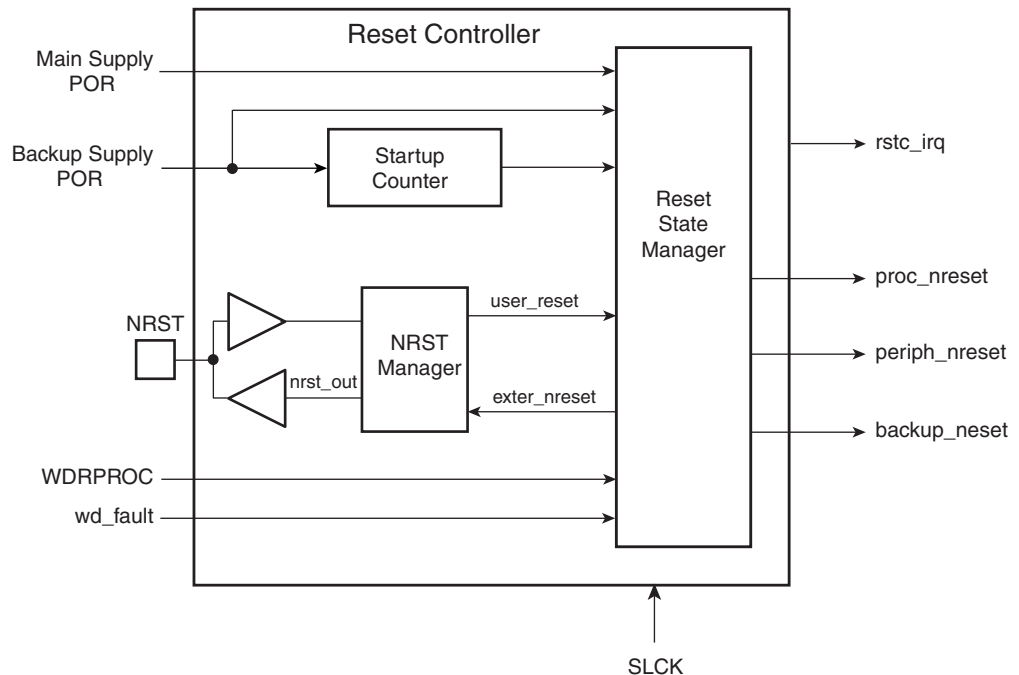
### 14.1 Overview

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.

### 14.2 Block Diagram

Figure 14-1. Reset Controller Block Diagram



### 14.3 Functional Description

#### 14.3.1 Reset Controller Overview

The Reset Controller is made up of an NRST Manager, a Startup Counter 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.
- **backup\_nreset**: Affects all the peripherals powered by VDDBU.
- **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.

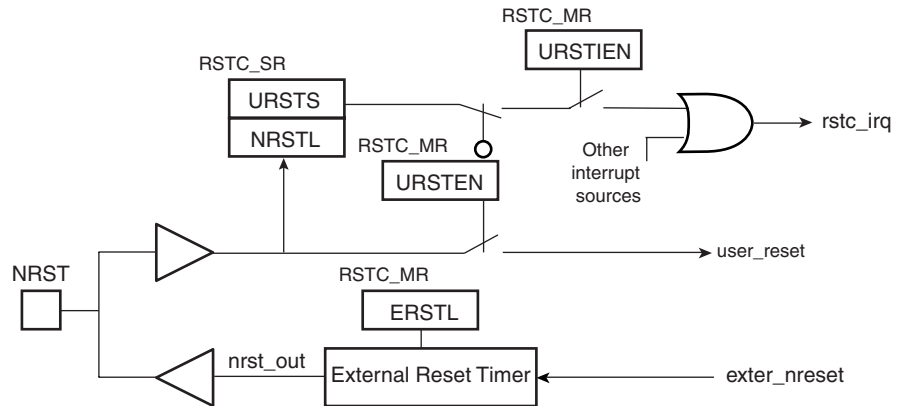
The startup counter waits for the complete crystal oscillator startup. The wait delay is given by the crystal oscillator startup time maximum value that can be found in the section Crystal Oscillator Characteristics in the Electrical Characteristics section of the product documentation.

The Reset Controller Mode Register (RSTC\_MR), allowing the configuration of the Reset Controller, is powered with VDDBU, so that its configuration is saved as long as VDDBU is on.

### 14.3.2 NRST Manager

The NRST Manager samples the NRST input pin and drives this pin low when required by the Reset State Manager. Figure 14-2 shows the block diagram of the NRST Manager.

Figure 14-2. NRST Manager



#### 14.3.2.1 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.

#### 14.3.2.2 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  $\mu$ s 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 field is within RSTC\_MR, which is backed-up, this field can be used to shape the system power-up reset for devices requiring a longer startup time than the Slow Clock Oscillator.

### 14.3.3 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.

#### 14.3.3.1 General Reset

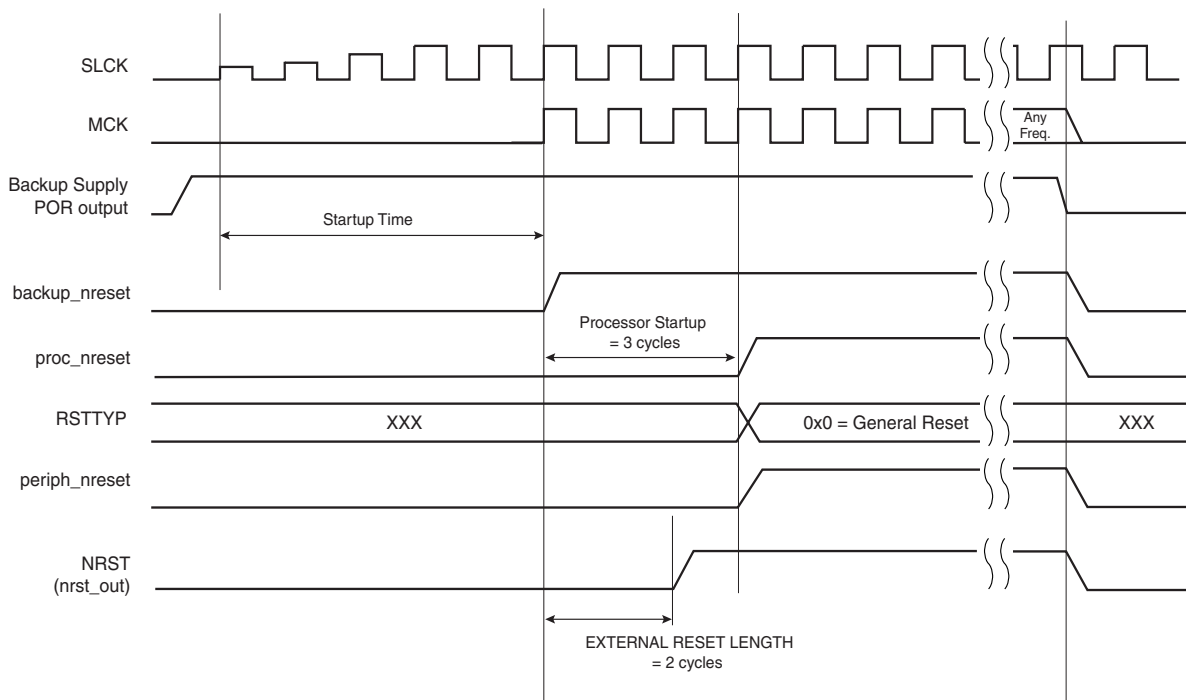
A general reset occurs when VDDDBU is powered on. The backup supply POR cell output rises and is filtered with a Startup Counter, which operates at Slow Clock. The purpose of this counter is to make sure the Slow Clock oscillator is stable before starting up the device. The length of startup time is hardcoded to comply with the Slow Clock Oscillator startup time.

After this time, the processor clock is released at Slow Clock and all the other signals remains valid for 3 cycles for proper processor and logic reset. Then, 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 backup\_nreset, as ERSTL defaults at value 0x0.

When VDDDBU is detected low by the Backup Supply POR Cell, all resets signals are immediately asserted, even if the Main Supply POR Cell does not report a Main Supply shut down.

Figure 14-3 shows how the General Reset affects the reset signals.

**Figure 14-3. General Reset State**



### 14.3.3.2 Wake-up Reset

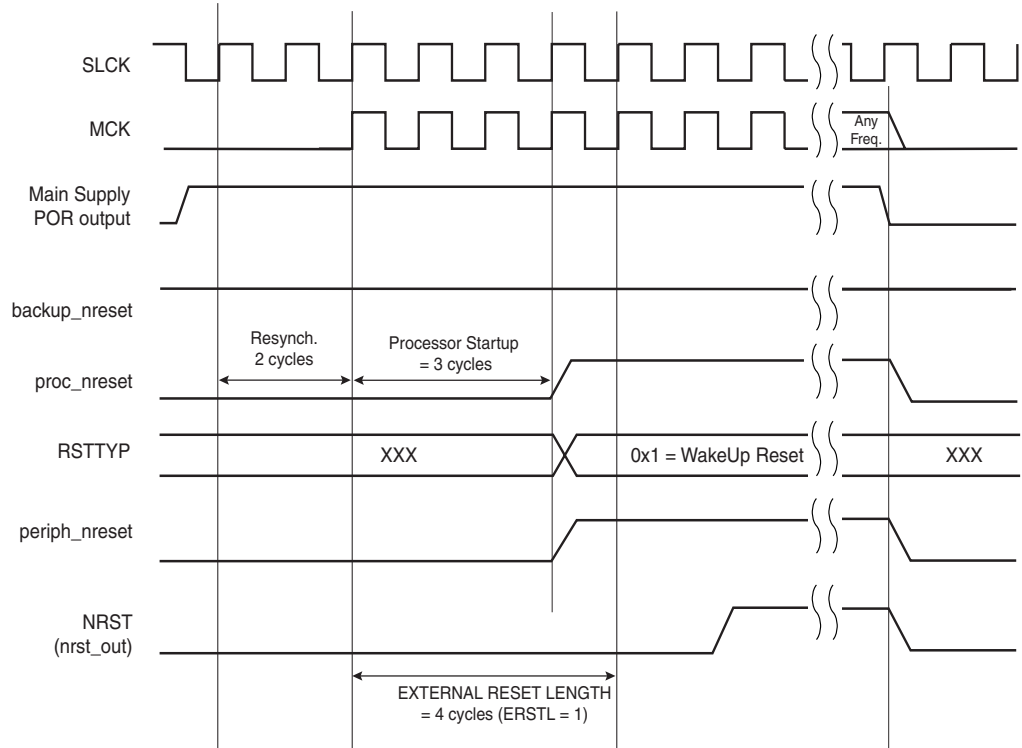
The Wake-up Reset occurs when the Main Supply is down. When the Main Supply POR output is active, all the reset signals are asserted except backup\_nreset. When the Main Supply powers up, the POR output is resynchronized on Slow Clock. The processor clock is then re-enabled during 3 Slow Clock cycles, depending on the requirements of the ARM processor.

At the end of this delay, the processor and other reset signals rise. The field RSTTYP in RSTC\_SR is updated to report a Wake-up Reset.

The “nrst\_out” remains asserted for EXTERNAL\_RESET\_LENGTH cycles. As RSTC\_MR is backed-up, the programmed number of cycles is applicable.

When the Main Supply is detected falling, the reset signals are immediately asserted. This transition is synchronous with the output of the Main Supply POR.

**Figure 14-4. Wake-up State**



### 14.3.3.3 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.

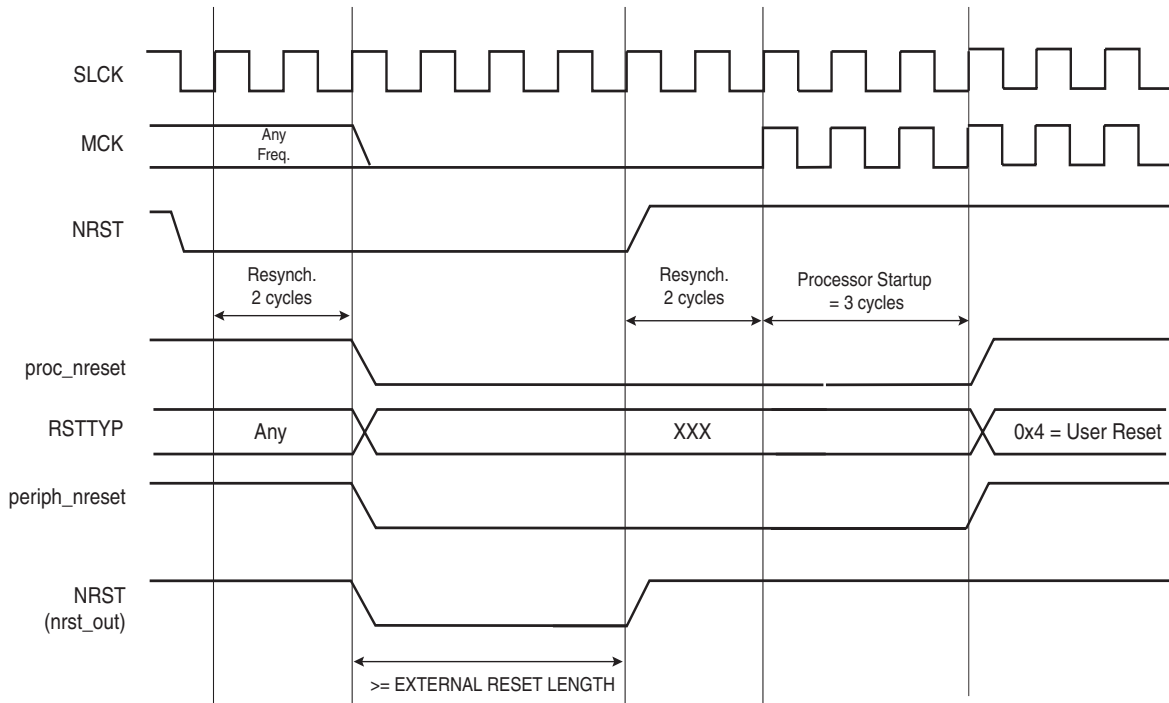
The User Reset is left when NRST rises, after a two-cycle resynchronization time and a three-cycle processor startup. The processor clock is re-enabled as soon as NRST is confirmed high.

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 14-5.** User Reset State



### 14.3.3.4 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.
- **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.

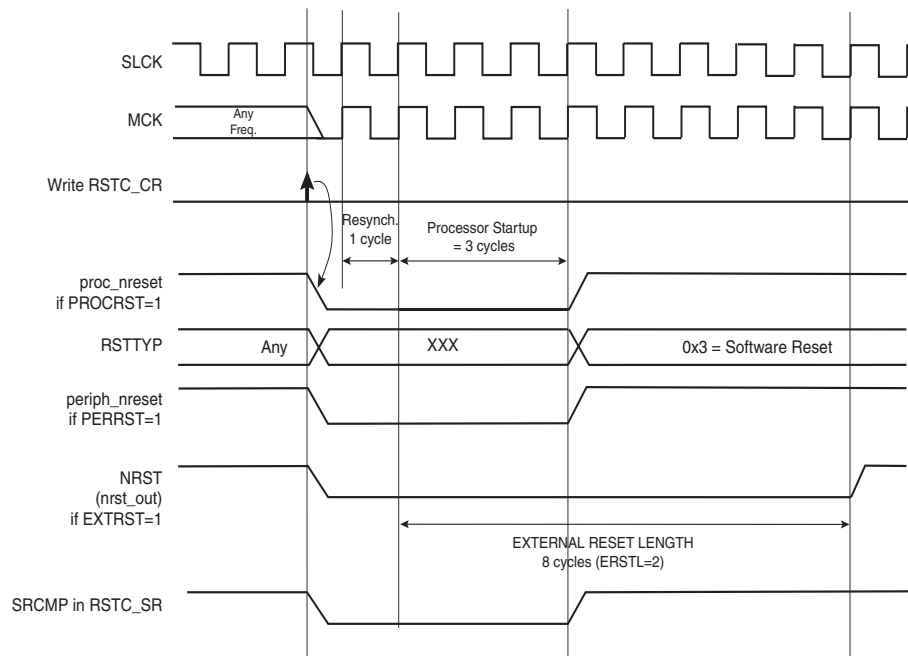
The internal reset signals are asserted as soon as the register write is performed. This is detected on the Master Clock (MCK). They are released when the software reset is left, i.e.; synchronously to SLCK.

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 14-6. Software Reset**



### 14.3.3.5 Watchdog Reset

The Watchdog Reset is entered when a watchdog fault occurs. This state lasts 3 Slow Clock cycles.

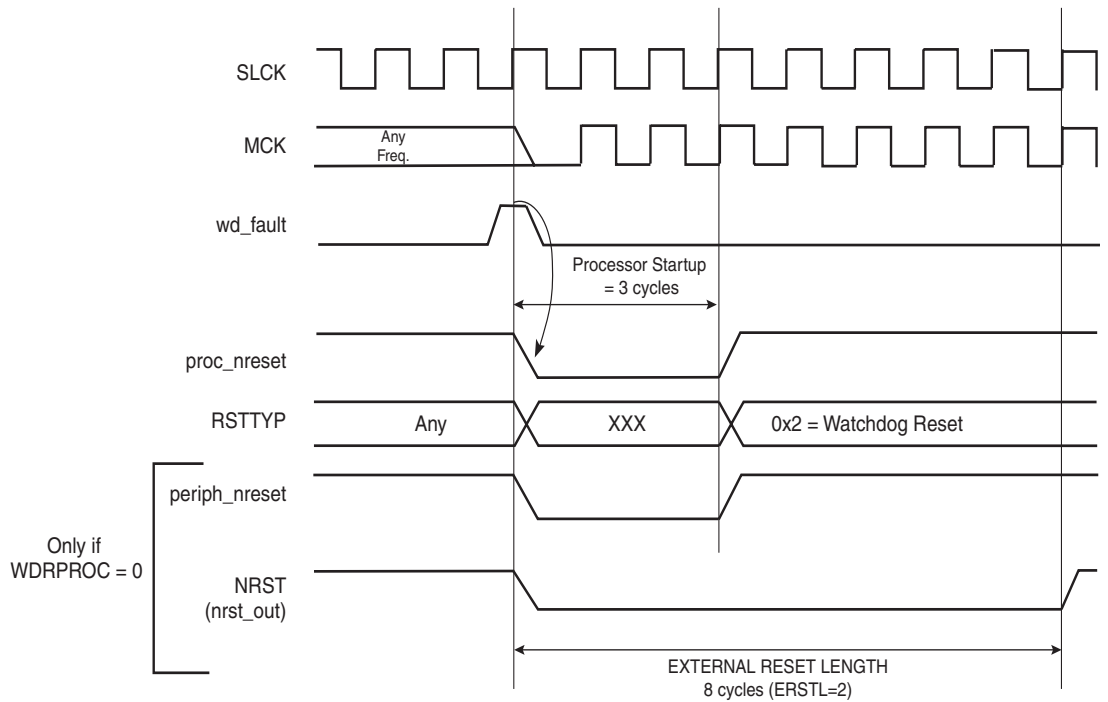
When in Watchdog Reset, assertion of the reset signals depends on the WDRPROC bit in WDT\_MR:

- 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 14-7. Watchdog Reset**



### 14.3.4 Reset State Priorities

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

- Backup Reset
- Wake-up Reset
- Watchdog Reset
- Software Reset
- User Reset

Particular cases are listed below:

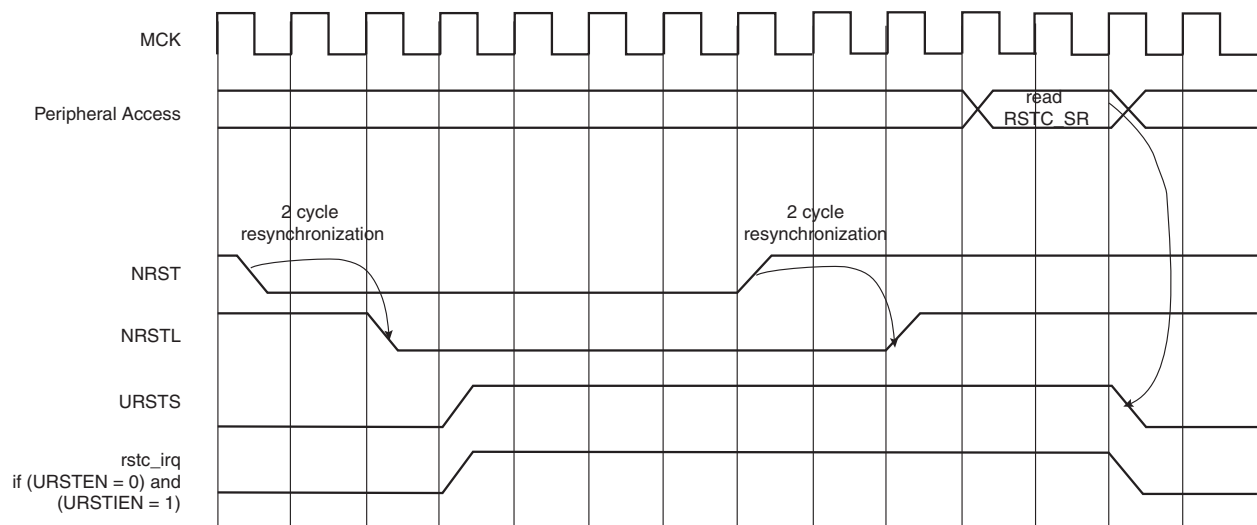
- When in User Reset:
  - A watchdog event is impossible because the Watchdog Timer is being reset by the `proc_nreset` signal.
  - A software reset is impossible, since the processor reset is being activated.
- When in Software Reset:
  - A watchdog event has priority over the current state.
  - The 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.

### 14.3.5 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 14-8](#)). 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 14-8.** Reset Controller Status and Interrupt



## 14.4 Reset Controller (RSTC) User Interface

**Table 14-1.** Reset Controller (RSTC) Register Mapping

Offset	Register	Name	Access	Reset Value	Back-up Reset Value
0x00	Control Register	RSTC_CR	Write-only	-	
0x04	Status Register	RSTC_SR	Read-only	0x0000_0001	0x0000_0000
0x08	Mode Register	RSTC_MR	Read/Write	-	0x0000_0000

Note: 1. The reset value of RSTC\_SR either reports a General Reset or a Wake-up Reset depending on last rising power supply.

### 14.4.1 Reset Controller Control Register

**Register Name:** RSTC\_CR

**Access Type:** 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.

## 14.4.2 Reset Controller Status Register

**Register Name:** RSTC\_SR

**Access Type:** 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	Both VDD1V8 and VDDBU rising
0	0	1	Wake Up Reset	VDD1V8 rising
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.

### 14.4.3 Reset Controller Mode Register

**Register Name:** RSTC\_MR

**Access Type:** 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  $\mu$ s and 2 seconds.

- **KEY: Password**

Should be written at value 0xA5. Writing any other value in this field aborts the write operation.



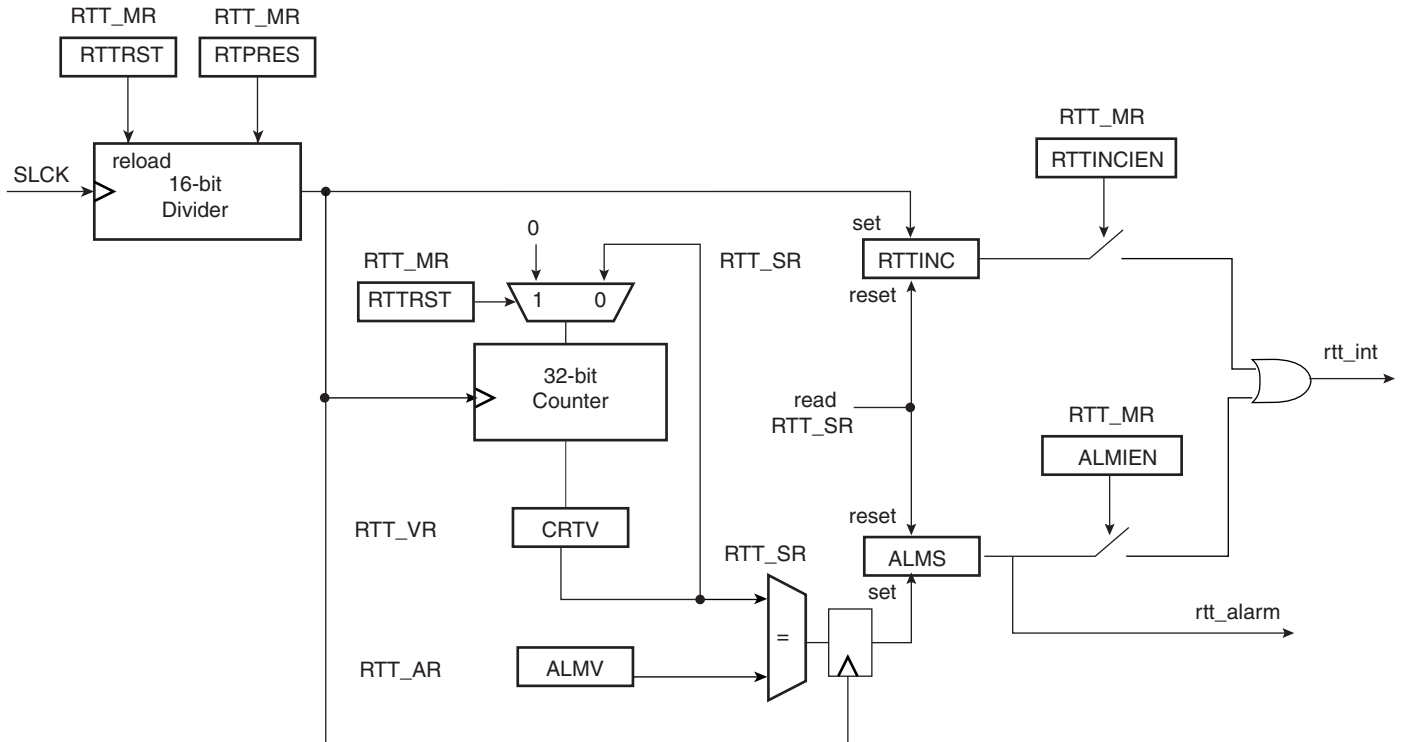
## 15. Real-time Timer (RTT)

### 15.1 Overview

The Real-time Timer is built around a 32-bit counter and used to count elapsed seconds. It generates a periodic interrupt or/and triggers an alarm on a programmed value.

### 15.2 Block Diagram

Figure 15-1. Real-time Timer



### 15.3 Functional Description

The Real-time Timer is 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 Hz). 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.

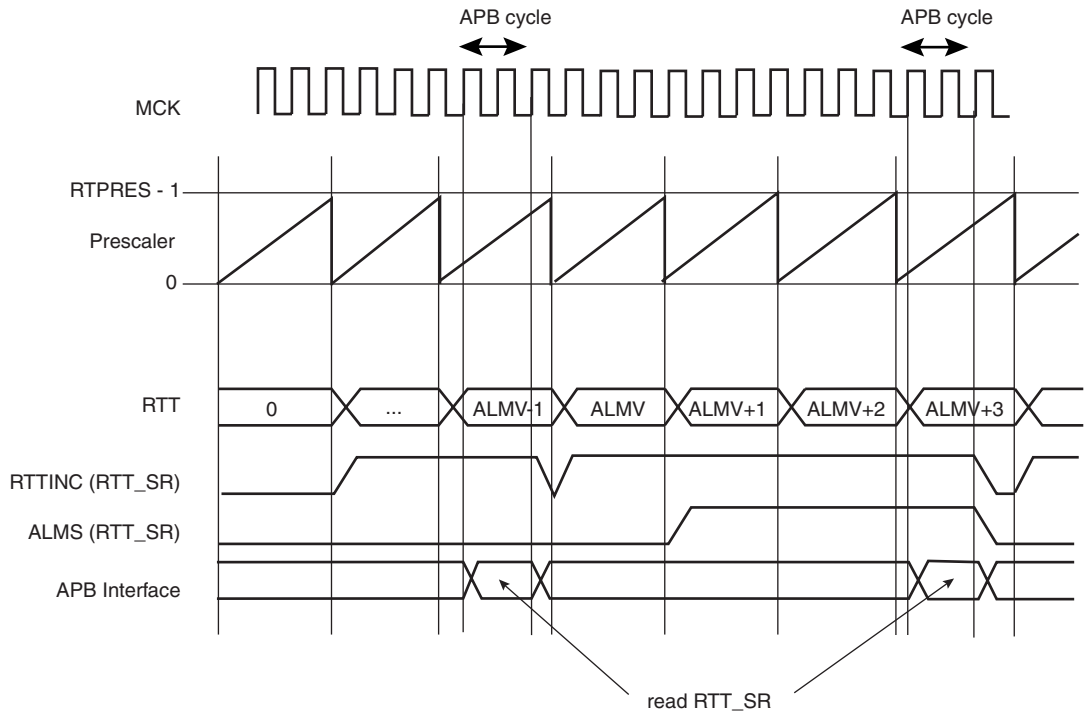
The bit RTTINC in RTT\_SR is set each time the Real-time Timer counter is incremented. This bit can be used to start a periodic interrupt, the period being one second when the RTPRES is programmed with 0x8000 and Slow Clock equal to 32.768 Hz.

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 15-2.** RTT Counting



**15.4 Real-time Timer (RTT) User Interface****Table 15-1.** Real-time Timer (RTT) Register Mapping

<b>Offset</b>	<b>Register</b>	<b>Name</b>	<b>Access</b>	<b>Reset Value</b>
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

### 15.4.1 Real-time Timer Mode Register

**Register Name:** RTT\_MR

**Access Type:** Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	RTRRST	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  $2^{16}$

RTPRES  $\neq$  0: The Prescaler Period is equal to RTPRES.

- **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.

- **RTRRST: Real-time Timer Restart**

1 = Reloads and restarts the clock divider with the new programmed value. This also resets the 32-bit counter.

## 15.4.2 Real-time Timer Alarm Register

**Register Name:** RTT\_AR

**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.

## 15.4.3 Real-time Timer Value Register

**Register Name:** RTT\_VR

**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.



#### 15.4.4 Real-time Timer Status Register

Register Name: RTT\_SR

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.

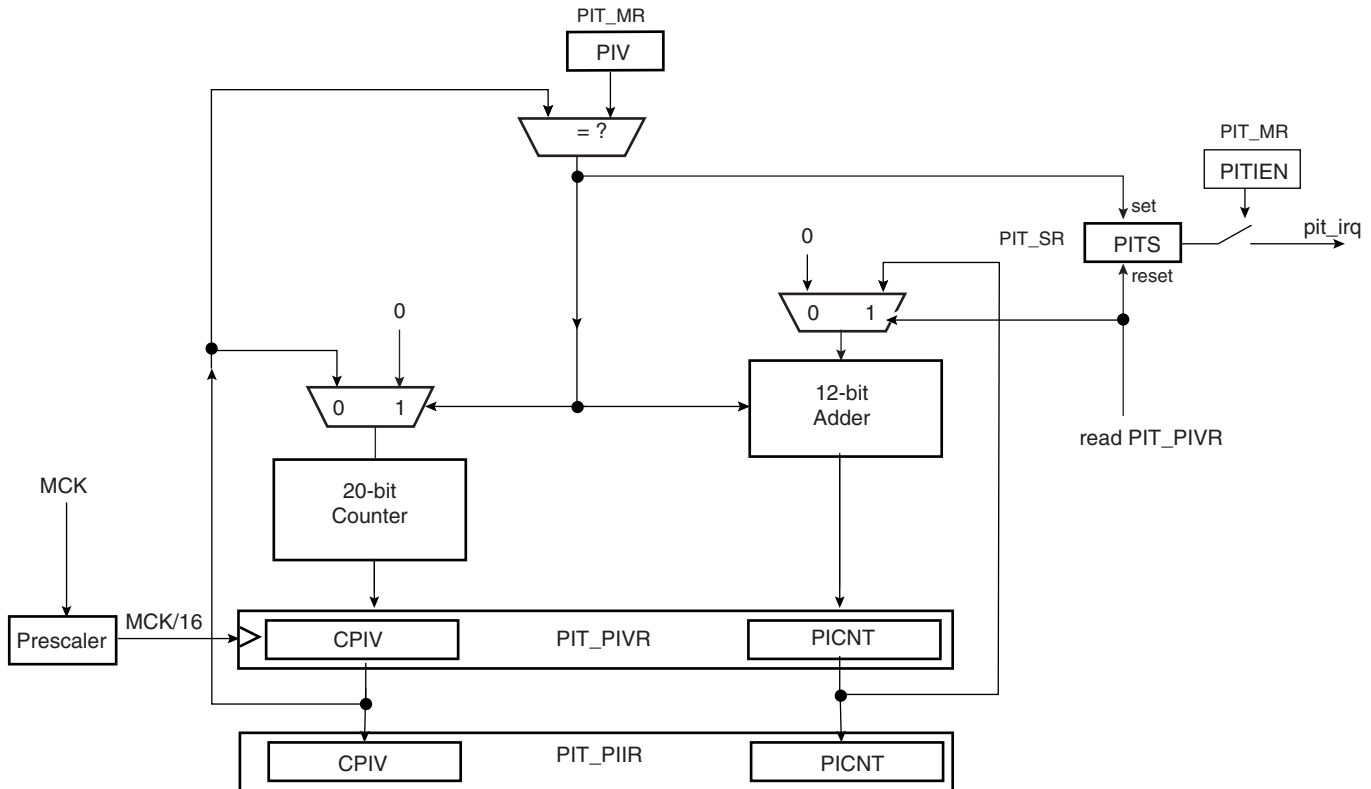
## 16. Periodic Interval Timer (PIT)

### 16.1 Overview

The Periodic Interval Timer (PIT) provides the operating system's scheduler interrupt. It is designed to offer maximum accuracy and efficient management, even for systems with long response time.

### 16.2 Block Diagram

Figure 16-1. Periodic Interval Timer



### 16.3 Functional Description

The Periodic Interval Timer aims at providing periodic interrupts for use by operating systems.

The PIT provides a programmable overflow counter and a reset-on-read feature. It is built around two counters: a 20-bit CPIV counter and a 12-bit PICNT counter. Both counters work at Master Clock /16.

The first 20-bit CPIV counter increments from 0 up to a programmable overflow value set in the field PIV of the Mode Register (PIT\_MR). When the counter CPIV reaches this value, it resets to 0 and increments the Periodic Interval Counter, PICNT. The status bit PITS in the Status Register (PIT\_SR) rises and triggers an interrupt, provided the interrupt is enabled (PITIEN in PIT\_MR).

Writing a new PIV value in PIT\_MR does not reset/restart the counters.

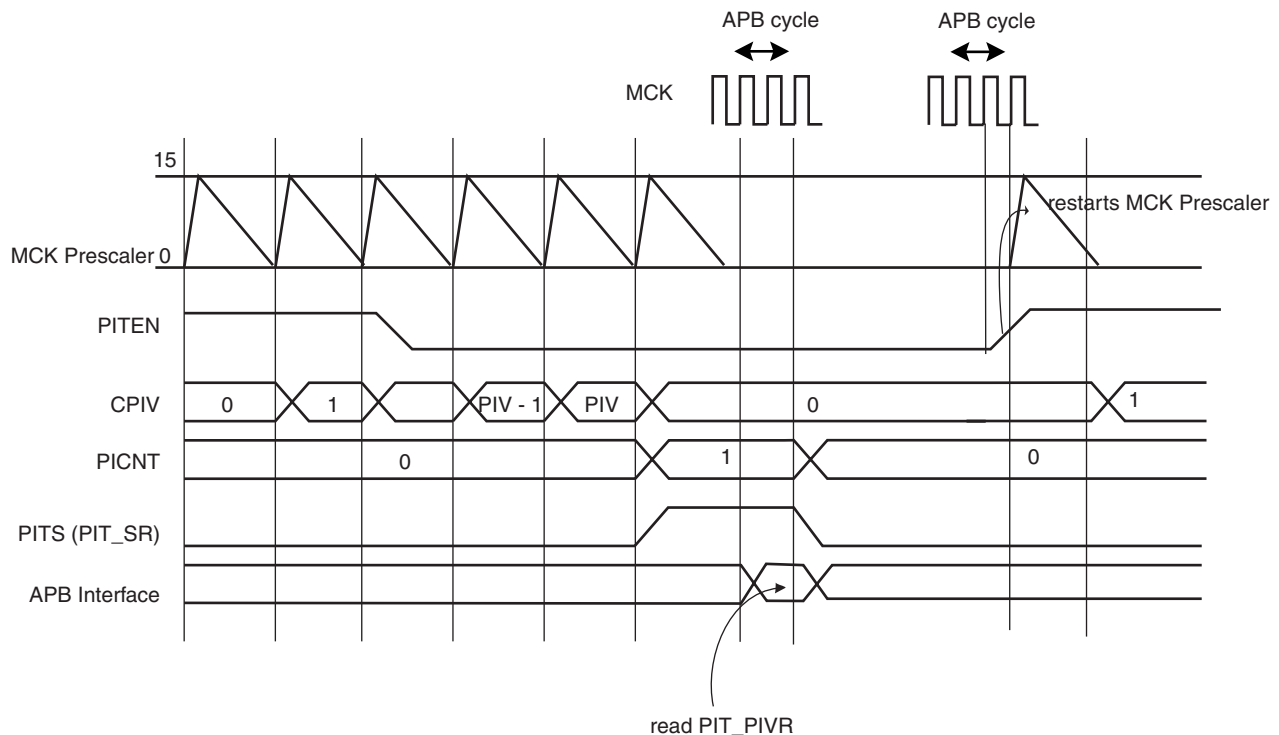
When CPIV and PICNT values are obtained by reading the Periodic Interval Value Register (PIT\_PIVR), the overflow counter (PICNT) is reset and the PITS is cleared, thus acknowledging the interrupt. The value of PICNT gives the number of periodic intervals elapsed since the last read of PIT\_PIVR.

When CPIV and PICNT values are obtained by reading the Periodic Interval Image Register (PIT\_PIIIR), there is no effect on the counters CPIV and PICNT, nor on the bit PITS. For example, a profiler can read PIT\_PIIIR without clearing any pending interrupt, whereas a timer interrupt clears the interrupt by reading PIT\_PIVR.

The PIT may be enabled/disabled using the PITEN bit in the PIT\_MR register (disabled on reset). The PITEN bit only becomes effective when the CPIV value is 0. Figure 16-2 illustrates the PIT counting. After the PIT Enable bit is reset (PITEN= 0), the CPIV goes on counting until the PIV value is reached, and is then reset. PIT restarts counting, only if the PITEN is set again.

The PIT is stopped when the core enters debug state.

**Figure 16-2.** Enabling/Disabling PIT with PITEN





## 16.4 Periodic Interval Timer (PIT) User Interface

**Table 16-1.** Periodic Interval Timer (PIT) Register Mapping

Offset	Register	Name	Access	Reset Value
0x00	Mode Register	PIT_MR	Read/Write	0x000F_FFFF
0x04	Status Register	PIT_SR	Read-only	0x0000_0000
0x08	Periodic Interval Value Register	PIT_PIVR	Read-only	0x0000_0000
0x0C	Periodic Interval Image Register	PIT_PIIR	Read-only	0x0000_0000

### 16.4.1 Periodic Interval Timer Mode Register

**Register Name:** PIT\_MR

**Access Type:** Read/Write

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

- **PIV: Periodic Interval Value**

Defines the value compared with the primary 20-bit counter of the Periodic Interval Timer (CPIV). The period is equal to (PIV + 1).

- **PITEN: Period Interval Timer Enabled**

0 = The Periodic Interval Timer is disabled when the PIV value is reached.

1 = The Periodic Interval Timer is enabled.

- **PITIEN: Periodic Interval Timer Interrupt Enable**

0 = The bit PITS in PIT\_SR has no effect on interrupt.

1 = The bit PITS in PIT\_SR asserts interrupt.

### 16.4.2 Periodic Interval Timer Status Register

**Register Name:** PIT\_SR

**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
-	-	-	-	-	-	-	PITS

- **PITS: Periodic Interval Timer Status**

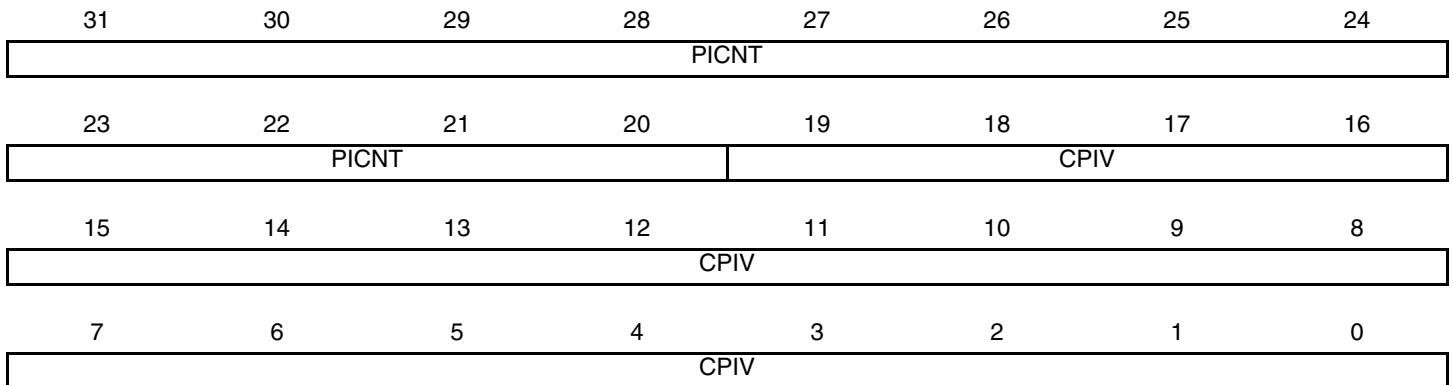
0 = The Periodic Interval timer has not reached PIV since the last read of PIT\_PIVR.

1 = The Periodic Interval timer has reached PIV since the last read of PIT\_PIVR.

## 16.4.3 Periodic Interval Timer Value Register

**Register Name:** PIT\_PIVR

**Access Type:** Read-only



Reading this register clears PITS in PIT\_SR.

- **CPIV: Current Periodic Interval Value**

Returns the current value of the periodic interval timer.

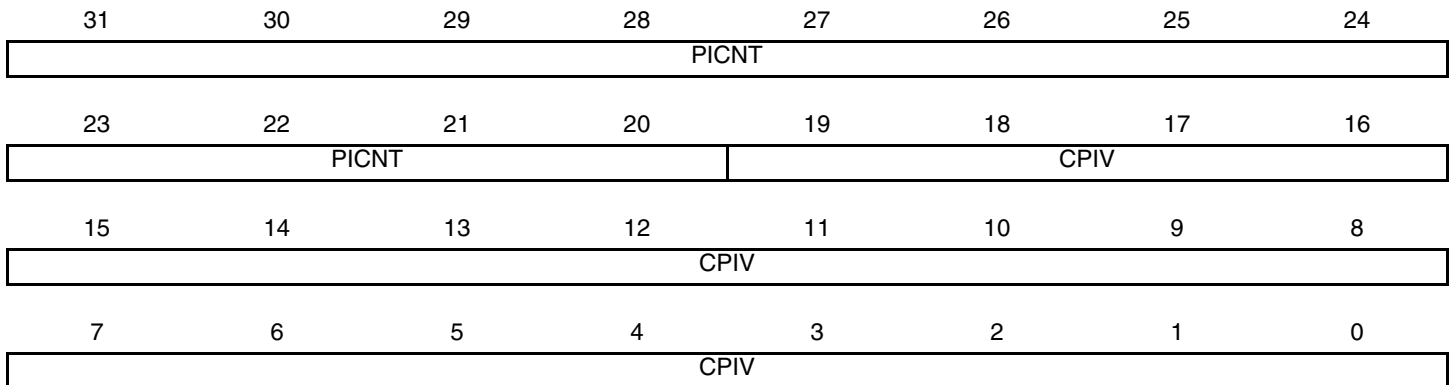
- **PICNT: Periodic Interval Counter**

Returns the number of occurrences of periodic intervals since the last read of PIT\_PIVR.

#### 16.4.4 Periodic Interval Timer Image Register

**Register Name:** PIT\_PIIR

**Access Type:** Read-only



- **CPIV: Current Periodic Interval Value**

Returns the current value of the periodic interval timer.

- **PICNT: Periodic Interval Counter**

Returns the number of occurrences of periodic intervals since the last read of PIT\_PIVR.

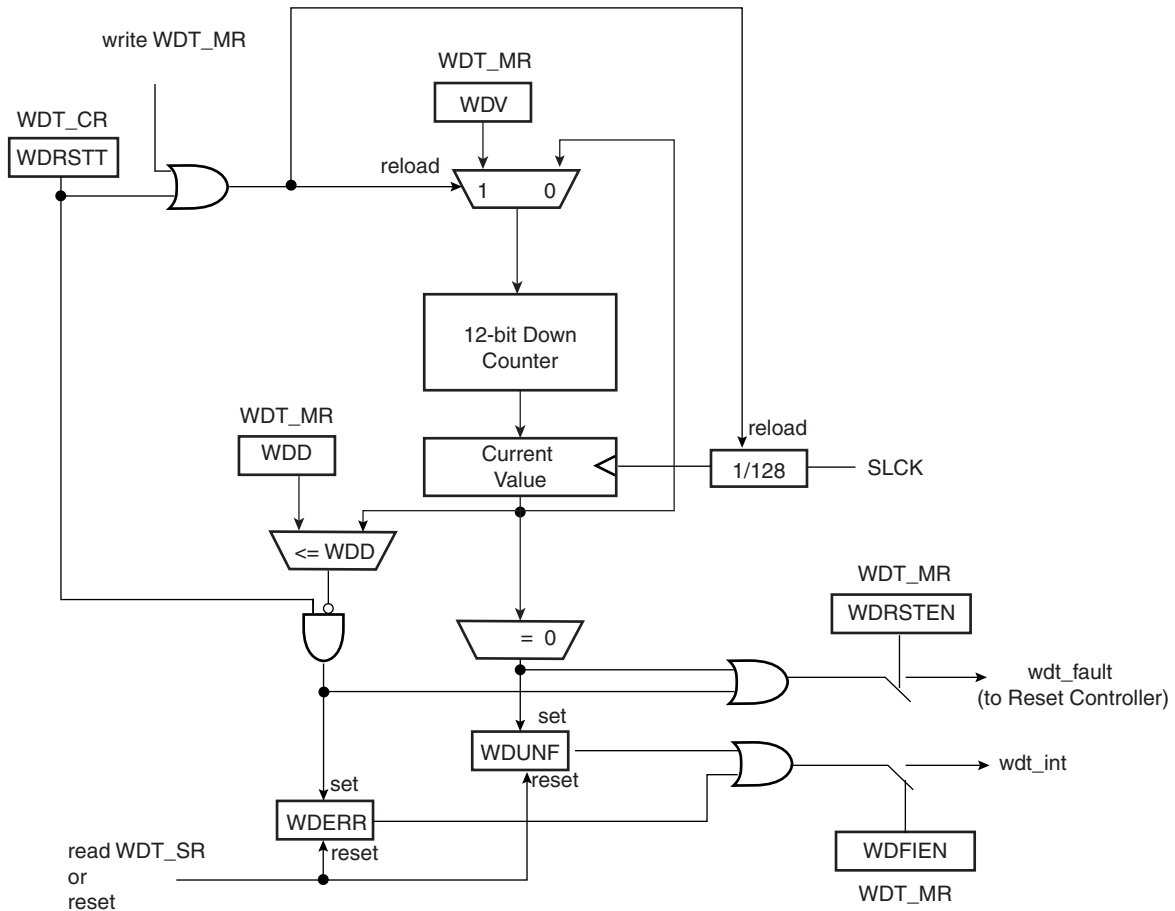
## 17. Watchdog Timer (WDT)

### 17.1 Overview

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.

### 17.2 Block Diagram

Figure 17-1. Watchdog Timer Block Diagram



## 17.3 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 0xFFFF, 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).

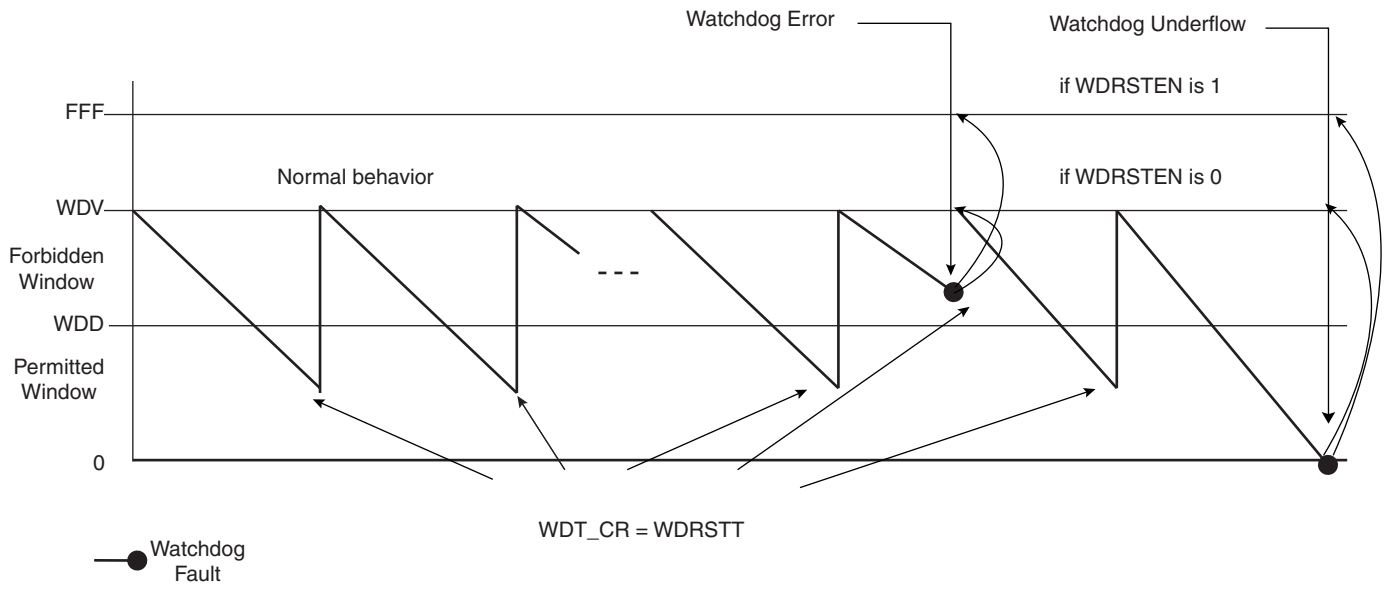
The status bits WDUNF (Watchdog Underflow) and WDERR (Watchdog Error) trigger an interrupt, provided the bit WDFIEN is set in the mode register. The signal “wdt\_fault” to the reset controller causes a Watchdog reset if the WDRSTEN bit is set as already explained in the reset controller programmer Datasheet. 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 17-2. Watchdog Behavior



## 17.4 Watchdog Timer (WDT) User Interface

**Table 17-1.** Watchdog Timer (WDT) Register Mapping

Offset	Register	Name	Access	Reset Value
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

### 17.4.1 Watchdog Timer Control Register

**Register Name:** WDT\_CR

**Access Type:** 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.



## 17.4.2 Watchdog Timer Mode Register

**Register Name:** WDT\_MR

**Access Type:** 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.

### 17.4.3 Watchdog Timer Status Register

**Register Name:** WDT\_SR

**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
–	–	–	–	–	–	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.

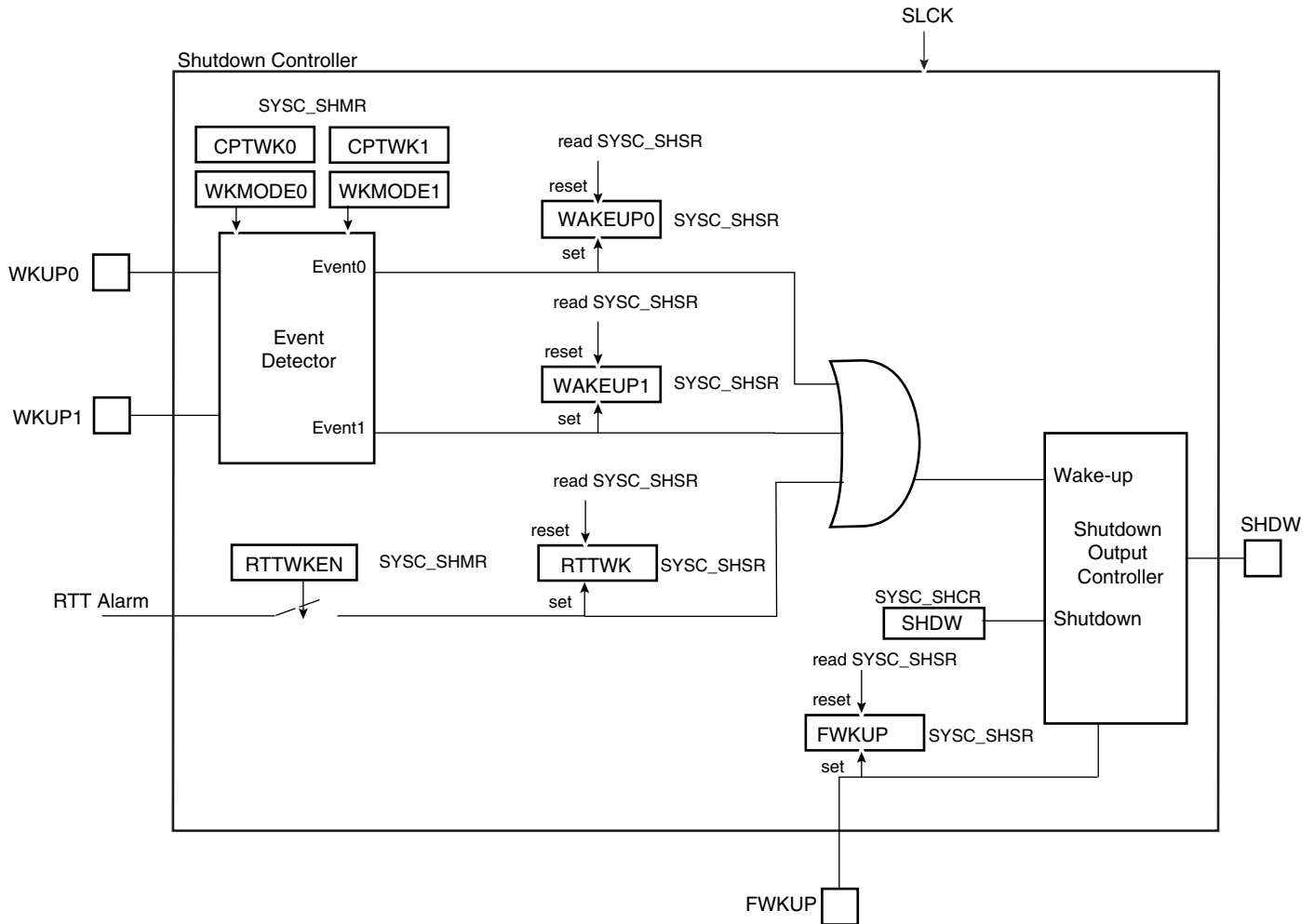
## 18. Shutdown Controller (SHDWC)

### 18.1 Overview

The Shutdown Controller controls the power supplies VDD3V3 and VDD1V8 and the wake-up detection on debounced input lines. A dedicated input, Force Wake Up, is also available.

### 18.2 Block Diagram

Figure 18-1. Shutdown Controller Block Diagram



## 18.3 I/O Lines Description

**Table 18-1.** I/O Lines Description

Name	Description	Type
FWKUP	Force Wake Up input for the Shutdown Controller	Input
WKUP0	Wake-up 0 input	Input
WKUP1	Wake-up 1 input	Input
SHDW	Shutdown output	Output

## 18.4 Product Dependencies

### 18.4.1 Power Management

The Shutdown Controller is continuously clocked by Slow Clock. The Power Management Controller has no effect on the behavior of the Shutdown Controller.

## 18.5 Functional Description

The Shutdown Controller manages the main power supply. To do so, it is supplied with VDDBU and manages wake-up input pins and one output pin, SHDW.

A typical application connects the pin SHDW to the shutdown input of the DC/DC Converter providing the main power supplies of the system, and especially VDD1V8 and/or VDD3V3. The wake-up inputs (WKUP0, WKUP1, FWKUP) connect to any push-buttons or signal that wake up the system.

The software is able to control the pin SHDW by writing the Shutdown Control Register (SHDW\_CR) with the bit SHDW at 1. This register is password-protected and so the value written should contain the correct key for the command to be taken into account. As a result, the system should be powered down.

A level change on WKUP0 or WKUP1 is used as wake-up. Wake-up is configured in the Shutdown Mode Register (SHDW\_MR). The transition detector can be programmed to detect either a positive or negative transition or any level change on WKUP0 and WKUP1. The detection can also be disabled. Programming is performed by defining WKMODE0 and WKMODE1.

Moreover, a debouncing circuit can be programmed for WKUP0 or WKUP1. The debouncing circuit filters pulses on WKUP0 or WKUP1 shorter than the programmed number of 16 SLCK cycles in CPTWK0 or CPTWK1 of the SHDW\_MR register. If the programmed level change is detected on a pin, a counter starts. When the counter reaches the value programmed in the corresponding field, CPTWK0 or CPTWK1, the SHDW pin is released. If a new input change is detected before the counter reaches the corresponding value, the counter is stopped and cleared. WAKEUP0 and/or WAKEUP1 of the Status Register (SHDW\_SR) reports the detection of the programmed events on WKUP0 or WKUP1, with a reset after the read of SHDW\_SR.

The Shutdown Controller can be programmed so as to activate the wake-up using the RTT alarm (the detection of the rising edge of the RTT alarm is synchronized with SLCK). This is done by writing the SHDW\_MR register using the RTTWKEN fields. When enabled, the detection of the RTT alarm is reported in the RTTWK bit of the SHDW\_SR Status register. It is reset

after the read of SHDW\_SR. When using the RTT alarm to wake up the system, the user must ensure that the RTT alarm status flag is cleared before shutting down the system. Otherwise, no rising edge of the status flag may be detected and the wake-up fails.

The pin FWKUP is treated differently and a low level on this pin forces a de-assertion of the SHDW pin, regardless of the presence of the Slow Clock. The bit FWKUP in the status register reports a Forced Wakeup Event after internal resynchronization of the event with the Slow Clock.

## 18.6 Shutdown Controller (SHDWC) User Interface

### 18.6.1 Register Mapping

**Table 18-2.** Shutdown Controller (SHDWC) Registers

Offset	Register	Name	Access	Reset Value
0x00	Shutdown Control Register	SHDW_CR	Write-only	-
0x04	Shutdown Mode Register	SHDW_MR	Read-Write	0x0000_0303
0x18	Shutdown Status Register	SHDW_SR	Read-only	0x0000_0000

### 18.6.2 Shutdown Control Register

**Register Name:** SHDW\_CR

**Access Type:** 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
-	-	-	-	-	-	-	SHDW

- **SHDW: Shut Down Command**

0 = No effect.

1 = If KEY is correct, asserts the SHDW pin.

- **KEY: Password**

Should be written at value 0xA5. Writing any other value in this field aborts the write operation.

## 18.6.3 Shutdown Mode Register

**Register Name:** SHDW\_MR

**Access Type:** Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	RTTWKEN
15	14	13	12	11	10	9	8
CPTWK1				–	–	WKMODE1	
7	6	5	4	3	2	1	0
CPTWK0				–	–	WKMODE0	

- **WKMODE0: Wake-up Mode 0**
- **WKMODE1: Wake-up Mode 1**

WKMODE[1:0]		Wake-up Input Transition Selection
0	0	None. No detection is performed on the wake-up input
0	1	Low to high level
1	0	High to low level
1	1	Both levels change

- **CPTWK0: Counter on Wake-up 0**
- **CPTWK1: Counter on Wake-up 1**

Defines the number of 16 Slow Clock cycles, the level detection on the corresponding input pin shall last before the wake-up event occurs. Because of the internal synchronization of WKUP0 and WKUP1, the SHDW pin is released (CPTWK x 16 + 1) Slow Clock cycles after the event on WKUP.

- **RTTWKEN: Real-time Timer Wake-up Enable**

0 = The RTT Alarm signal has no effect on the Shutdown Controller.

1 = The RTT Alarm signal forces the de-assertion of the SHDW pin.

### 18.6.4 Shutdown Status Register

**Register Name:** SHDW\_SR

**Access Type:** Read-only

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	RTTWK
15	14	13	12	11	10	9	8
-	-	-	-	-	-	-	-
7	6	5	4	3	2	1	0
-	-	-	-	-	FWKUP	WAKEUP1	WAKEUP0

- **WAKEUP0: Wake-up 0 Status**

- **WAKEUP1: Wake-up 1 Status**

0 = No wake-up event occurred on the corresponding wake-up input since the last read of SHDW\_SR.

1 = At least one wake-up event occurred on the corresponding wake-up input since the last read of SHDW\_SR.

- **FWKUP: Force Wake Up Status**

0 = No wake-up event occurred on the Force Wake Up input since the last read of SHDW\_SR.

1 = At least one wake-up event occurred on the Force Wake Up input since the last read of SHDW\_SR.

- **RTTWK: Real-time Timer Wake-up**

0 = No wake-up alarm from the RTT occurred since the last read of SHDW\_SR.

1 = At least one wake-up alarm from the RTT occurred since the last read of SHDW\_SR.



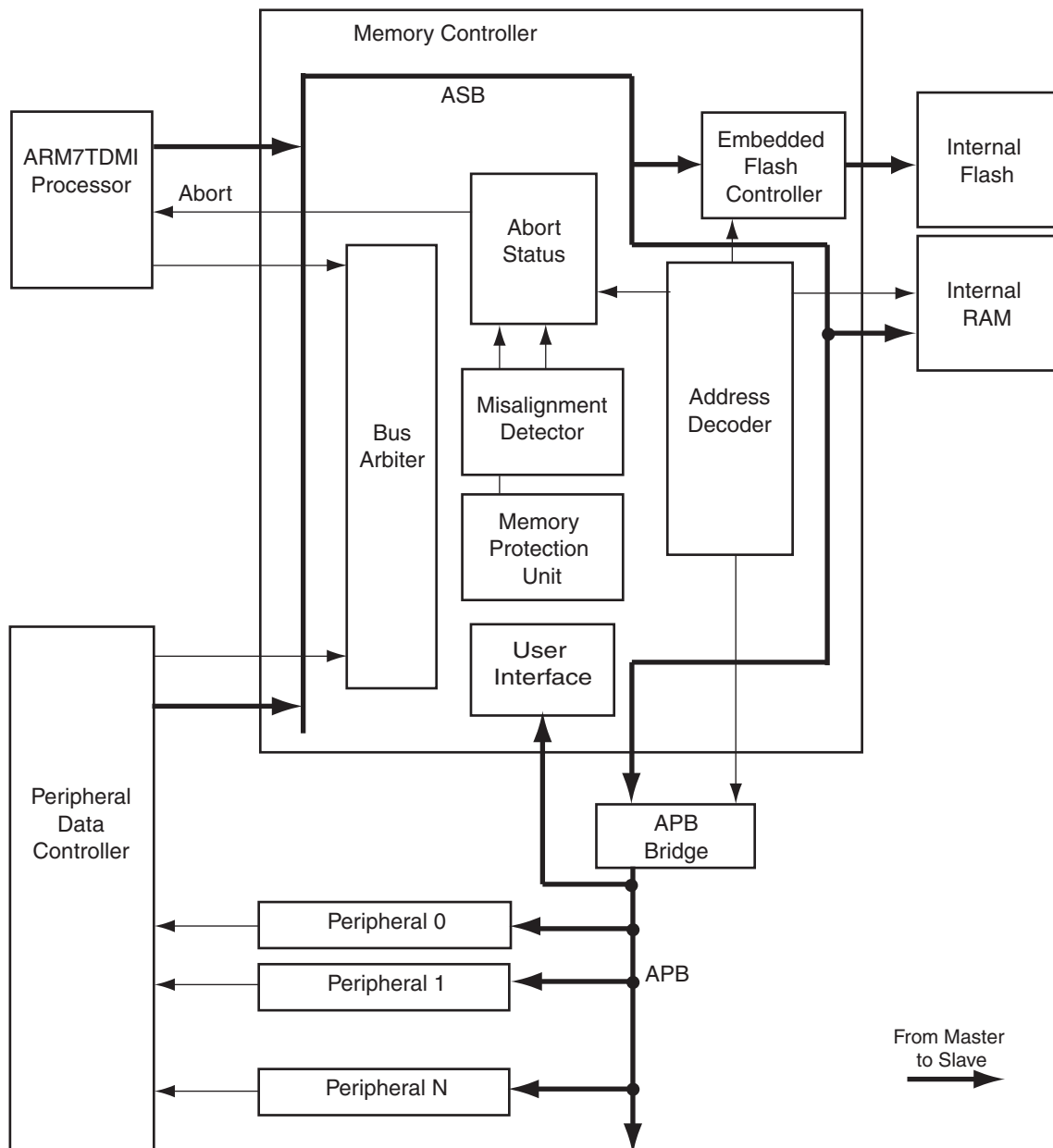
## 19. Memory Controller (MC)

### 19.1 Overview

The Memory Controller (MC) manages the ASB bus and controls accesses requested by the masters, typically the ARM7TDMI processor and the Peripheral Data Controller. It features a simple bus arbiter, an address decoder, an abort status, a misalignment detector and an Embedded Flash Controller. In addition, the MC contains a Memory Protection Unit (MPU) consisting of 16 areas that can be protected against write and/or user accesses. Access to peripherals can be protected in the same way.

### 19.2 Block Diagram

Figure 19-1. Memory Controller Block Diagram



### 19.3 Functional Description

The Memory Controller handles the internal ASB bus and arbitrates the accesses of both masters.

It is made up of:

- A bus arbiter
- An address decoder
- An abort status
- A misalignment detector
- A memory protection unit
- An Embedded Flash Controller

The MC handles only little-endian mode accesses. The masters work in little-endian mode only.

#### 19.3.1 Bus Arbiter

The Memory Controller has a simple, hard-wired priority bus arbiter that gives the control of the bus to one of the two masters. The Peripheral Data Controller has the highest priority; the ARM processor has the lowest one.

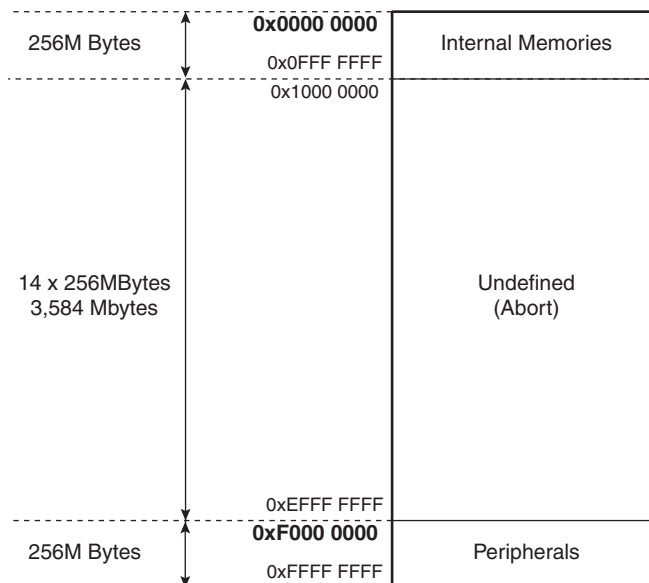
#### 19.3.2 Address Decoder

The Memory Controller features an Address Decoder that first decodes the four highest bits of the 32-bit address bus and defines three separate areas:

- One 256-Mbyte address space for the internal memories
- One 256-Mbyte address space reserved for the embedded peripherals
- An undefined address space of 3584M bytes representing fourteen 256-Mbyte areas that return an Abort if accessed

Figure 19-2 shows the assignment of the 256-Mbyte memory areas.

**Figure 19-2.** Memory Areas



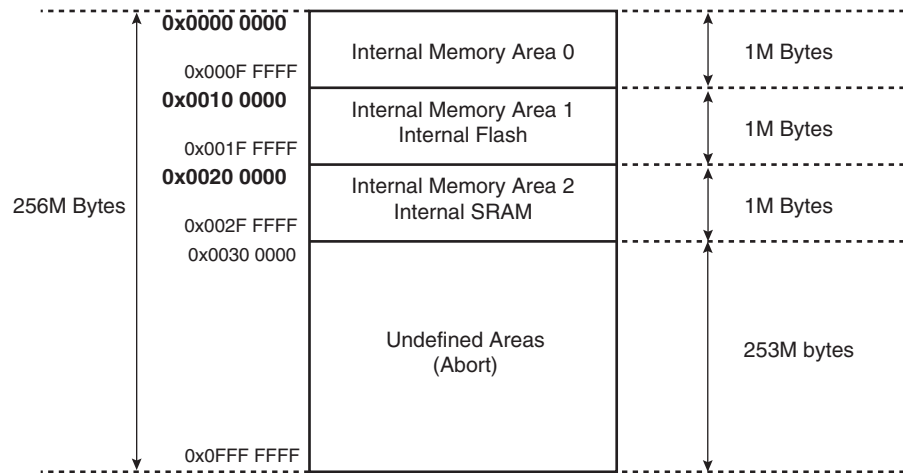
## 19.3.2.1 Internal Memory Mapping

Within the Internal Memory address space, the Address Decoder of the Memory Controller decodes eight more address bits to allocate 1-Mbyte address spaces for the embedded memories.

The allocated memories are accessed all along the 1-Mbyte address space and so are repeated n times within this address space, n equaling 1M bytes divided by the size of the memory.

When the address of the access is undefined within the internal memory area, the Address Decoder returns an Abort to the master.

**Figure 19-3.** Internal Memory Mapping



## 19.3.2.2 Internal Memory Area 0

The first 32 bytes of Internal Memory Area 0 contain the ARM processor exception vectors, in particular, the Reset Vector at address 0x0.

Before execution of the remap command, the on-chip Flash is mapped into Internal Memory Area 0, so that the ARM7TDMI reaches an executable instruction contained in Flash. After the remap command, the internal SRAM at address 0x0020 0000 is mapped into Internal Memory Area 0. The memory mapped into Internal Memory Area 0 is accessible in both its original location and at address 0x0.

## 19.3.3 Remap Command

After execution, the Remap Command causes the Internal SRAM to be accessed through the Internal Memory Area 0.

As the ARM vectors (Reset, Abort, Data Abort, Prefetch Abort, Undefined Instruction, Interrupt, and Fast Interrupt) are mapped from address 0x0 to address 0x20, the Remap Command allows the user to redefine dynamically these vectors under software control.

The Remap Command is accessible through the Memory Controller User Interface by writing the MC\_RCR (Remap Control Register) RCB field to one.

The Remap Command can be cancelled by writing the MC\_RCR RCB field to one, which acts as a toggling command. This allows easy debug of the user-defined boot sequence by offering a simple way to put the chip in the same configuration as after a reset.

### 19.3.4 Abort Status

There are three reasons for an abort to occur:

- access to an undefined address
- access to a protected area without the permitted state
- an access to a misaligned address.

When an abort occurs, a signal is sent back to all the masters, regardless of which one has generated the access. However, only the ARM7TDMI can take an abort signal into account, and only under the condition that it was generating an access. The Peripheral Data Controller does not handle the abort input signal. Note that the connection is not represented in [Figure 19-1](#).

To facilitate debug or for fault analysis by an operating system, the Memory Controller integrates an Abort Status register set.

The full 32-bit wide abort address is saved in MC\_AASR. Parameters of the access are saved in MC\_ASR and include:

- the size of the request (field ABTSZ)
- the type of the access, whether it is a data read or write, or a code fetch (field ABTTYP)
- whether the access is due to accessing an undefined address (bit UNDADD), a misaligned address (bit MISADD) or a protection violation (bit MPU)
- the source of the access leading to the last abort (bits MST0 and MST1)
- whether or not an abort occurred for each master since the last read of the register (bit SVMST0 and SVMST1) unless this information is loaded in MST bits

In the case of a Data Abort from the processor, the address of the data access is stored. This is useful, as searching for which address generated the abort would require disassembling the instructions and full knowledge of the processor context.

In the case of a Prefetch Abort, the address may have changed, as the prefetch abort is pipelined in the ARM processor. The ARM processor takes the prefetch abort into account only if the read instruction is executed and it is probable that several aborts have occurred during this time. Thus, in this case, it is preferable to use the content of the Abort Link register of the ARM processor.

### 19.3.5 Memory Protection Unit

The Memory Protection Unit allows definition of up to 16 memory spaces within the internal memories.

After reset, the Memory Protection Unit is disabled. Enabling it requires writing the Protection Unit Enable Register (MC\_PUER) with the PUEB at 1.

Programming of the 16 memory spaces is done in the registers MC\_PUIA0 to MC\_PUIA15.

The size of each of the memory spaces is programmable by a power of 2 between 1K bytes and 4M bytes. The base address is also programmable on a number of bits according to the size.

The Memory Protection Unit also allows the protection of the peripherals by programming the Protection Unit Peripheral Register (MC\_PUP) with the field PROT at the appropriate value.

The peripheral address space and each internal memory area can be protected against write and non-privileged access of one of the masters. When one of the masters performs a forbidden access, an Abort is generated and the Abort Status traces what has happened.

There is no priority in the protection of the memory spaces. In case of overlap between several memory spaces, the strongest protection is taken into account. If an access is performed to an address which is not contained in any of the 16 memory spaces, the Memory Protection Unit generates an abort. To prevent this, the user can define a memory space of 4M bytes starting at 0 and authorizing any access.

### 19.3.6 Embedded Flash Controller

The Embedded Flash Controller is added to the Memory Controller and ensures the interface of the flash block with the 32-bit internal bus. It allows an increase of performance in Thumb Mode for Code Fetch with its system of 32-bit buffers. It also manages with the programming, erasing, locking and unlocking sequences thanks to a full set of commands.

### 19.3.7 Misalignment Detector

The Memory Controller features a Misalignment Detector that checks the consistency of the accesses.

For each access, regardless of the master, the size of the access and the bits 0 and 1 of the address bus are checked. If the type of access is a word (32-bit) and the bits 0 and 1 are not 0, or if the type of the access is a half-word (16-bit) and the bit 0 is not 0, an abort is returned to the master and the access is cancelled. Note that the accesses of the ARM processor when it is fetching instructions are not checked.

The misalignments are generally due to software bugs leading to wrong pointer handling. These bugs are particularly difficult to detect in the debug phase.

As the requested address is saved in the Abort Status Register and the address of the instruction generating the misalignment is saved in the Abort Link Register of the processor, detection and fix of this kind of software bugs is simplified.

## 19.4 Memory Controller (MC) User Interface

Base Address: 0xFFFFF00

**Table 19-1.** Memory Controller (MC) Memory Mapping

Offset	Register	Name	Access	Reset State
0x00	MC Remap Control Register	MC_RCR	Write-only	
0x04	MC Abort Status Register	MC_ASR	Read-only	0x0
0x08	MC Abort Address Status Register	MC_AASR	Read-only	0x0
0x0C	Reserved			
0x10	MC Protection Unit Area 0	MC_PUIA0	Read/Write	0x0
0x14	MC Protection Unit Area 1	MC_PUIA1	Read/Write	0x0
0x18	MC Protection Unit Area 2	MC_PUIA2	Read/Write	0x0
0x1C	MC Protection Unit Area 3	MC_PUIA3	Read/Write	0x0
0x20	MC Protection Unit Area 4	MC_PUIA4	Read/Write	0x0
0x24	MC Protection Unit Area 5	MC_PUIA5	Read/Write	0x0
0x28	MC Protection Unit Area 6	MC_PUIA6	Read/Write	0x0
0x2C	MC Protection Unit Area 7	MC_PUIA7	Read/Write	0x0
0x30	MC Protection Unit Area 8	MC_PUIA8	Read/Write	0x0
0x34	MC Protection Unit Area 9	MC_PUIA9	Read/Write	0x0
0x38	MC Protection Unit Area 10	MC_PUIA10	Read/Write	0x0
0x3C	MC Protection Unit Area 11	MC_PUIA11	Read/Write	0x0
0x40	MC Protection Unit Area 12	MC_PUIA12	Read/Write	0x0
0x44	MC Protection Unit Area 13	MC_PUIA13	Read/Write	0x0
0x48	MC Protection Unit Area 14	MC_PUIA14	Read/Write	0x0
0x4C	MC Protection Unit Area 15	MC_PUIA15	Read/Write	0x0
0x50	MC Protection Unit Peripherals	MC_PUP	Read/Write	0x0
0x54	MC Protection Unit Enable Register	MC_PUER	Read/Write	0x0
0x60	EFC Configuration Registers		See EFC Part	

## 19.4.1 MC Remap Control Register

**Register Name:** MC\_RCR

**Access Type:** Write-only

**Absolute Address:** 0xFFFF FF00

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

- **RCB: Remap Command Bit**

0: No effect.

1: This Command Bit acts on a toggle basis: writing a 1 alternatively cancels and restores the remapping of the page zero memory devices.

### 19.4.2 MC Abort Status Register

**Register Name:** MC\_ASR  
**Access Type:** Read-only  
**Reset Value:** 0x0  
**Absolute Address:** 0xFFFF FF04

31	30	29	28	27	26	25	24
–	–	–	–	–	–	SVMST1	SVMST0
23	22	21	20	19	18	17	16
–	–	–	–	–	–	MST1	MST0
15	14	13	12	11	10	9	8
–	–	–	–	ABTTYP		ABTSZ	
7	6	5	4	3	2	1	0
–	–	–	–	–	MPU	MISADD	UNDADD

- **UNDADD: Undefined Address Abort Status**

0: The last abort was not due to the access of an undefined address in the address space.  
 1: The last abort was due to the access of an undefined address in the address space.

- **MISADD: Misaligned Address Abort Status**

0: The last aborted access was not due to an address misalignment.  
 1: The last aborted access was due to an address misalignment.

- **MPU: Memory Protection Unit Abort Status**

0: The last aborted access was not due to the Memory Protection Unit.  
 1: The last aborted access was due to the Memory Protection Unit.

- **ABTSZ: Abort Size Status**

ABTSZ		Abort Size
0	0	Byte
0	1	Half-word
1	0	Word
1	1	Reserved

- **ABTTYP: Abort Type Status**

ABTTYP		Abort Type
0	0	Data Read
0	1	Data Write
1	0	Code Fetch
1	1	Reserved

- **MST0: PDC Abort Source**

0: The last aborted access was not due to the PDC.  
 1: The last aborted access was due to the PDC.



- **MST1: ARM7TDMI Abort Source**

0: The last aborted access was not due to the ARM7TDMI.

1: The last aborted access was due to the ARM7TDMI.

- **SVMST0: Saved PDC Abort Source**

0: No abort due to the PDC occurred.

1: At least one abort due to the PDC occurred.

- **SVMST1: Saved ARM7TDMI Abort Source**

0: No abort due to the ARM7TDMI occurred.

1: At least one abort due to the ARM7TDMI occurred.

### 19.4.3 MC Abort Address Status Register

**Register Name:** MC\_AASR

**Access Type:** Read-only

**Reset Value:** 0x0

**Absolute Address:** 0xFFFF FF08

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

- **ABTADD: Abort Address**

This field contains the address of the last aborted access.

#### 19.4.4 MC Protection Unit Area 0 to 15 Registers

**Register Name:** MC\_PUIA0 - MC\_PUIA15

**Access Type:** Read/Write

**Reset Value:** 0x0

**Absolute Address:** 0xFFFFFFF10 - 0xFFFFFFF4C

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

• **PROT: Protection:**

PROT		Processor Mode	
		Privilege	User
0	0	No access	No access
0	1	Read/Write	No access
1	0	Read/Write	Read-only
1	1	Read/Write	Read/Write

• **SIZE: Internal Area Size**

SIZE				Area Size	LSB of BA
0	0	0	0	1 KB	10
0	0	0	1	2 KB	11
0	0	1	0	4 KB	12
0	0	1	1	8 KB	13
0	1	0	0	16 KB	14
0	1	0	1	32 KB	15
0	1	1	0	64 KB	16
0	1	1	1	128 KB	17
1	0	0	0	256 KB	18
1	0	0	1	512 KB	19
1	0	1	0	1 MB	20
1	0	1	1	2 MB	21
1	1	0	1	4 MB	22

• **BA: Internal Area Base Address**

These bits define the Base Address of the area. Note that only the most significant bits of BA are significant. The number of significant bits are in respect with the size of the area.

## 19.4.5 MC Protection Unit Peripheral

**Register Name:** MC\_PUP  
**Access Type:** Read/Write  
**Reset Value:** 0x00000000  
**Absolute Address:** 0xFFFFF50

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

- **PROT: Protection**

PROT		Processor Mode	
		Privilege	User
0	0	Read/Write	No access
0	1	Read/Write	No access
1	0	Read/Write	Read-only
1	1	Read/Write	Read/Write



### 19.4.6 MC Protection Unit Enable Register

**Register Name:** MC\_PUER  
**Access Type:** Read/Write  
**Reset Value:** 0x00000000  
**Absolute Address:** 0xFFFFF54

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

- **PUEB: Protection Unit Enable Bit**

0: The Memory Controller Protection Unit is disabled.

1: The Memory Controller Protection Unit is enabled.

## 20. Embedded Flash Controller (EFC)

### 20.1 Overview

The Embedded Flash Controller is added to the Memory Controller and ensures the interface of the Flash block with the 32-bit internal bus. It increases performance in Thumb Mode for Code Fetch with its system of 32-bit buffers. It also manages the programming, erasing, locking and unlocking sequences using a full set of commands.

### 20.2 Functional Description

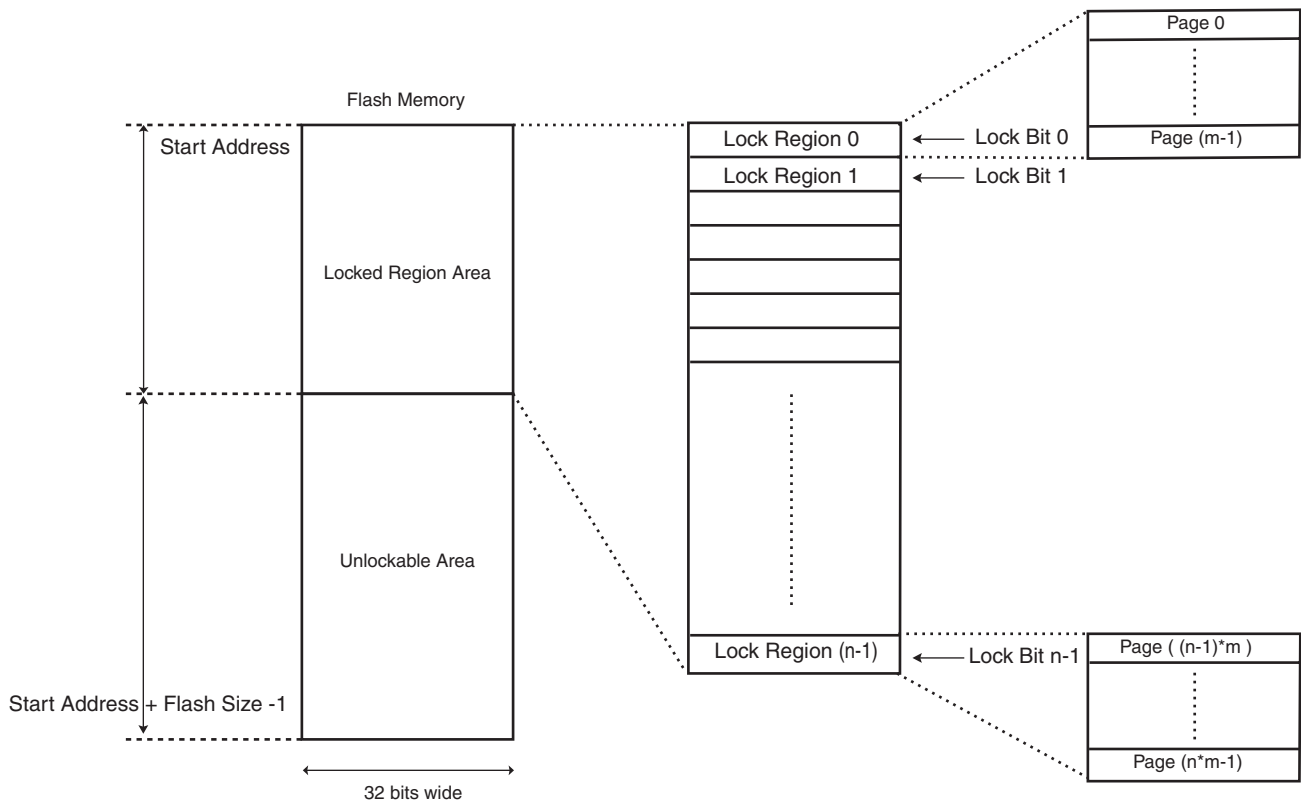
#### 20.2.1 Embedded Flash Organization

The Embedded Flash interfaces directly to the 32-bit internal bus. It is composed of several interfaces:

- One memory plane organized in several pages of the same size.
- Two 32-bit read buffers used for code read optimization (see ["Read Operations" on page 102](#)).
- 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 (see ["Write Operations" on page 104](#)).
- 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.

The Embedded Flash size, the page size and the lock region organization are described in the product definition section.

**Figure 20-1.** Embedded Flash Memory Mapping



### 20.2.2 Read Operations

An optimized controller manages embedded Flash reads. A system of 2 x 32-bit buffers is added in order to start access at following address during the second read, thus increasing performance when the processor is running in Thumb mode (16-bit instruction set). See [Figure 20-2](#), [Figure 20-3](#) and [Figure 20-4](#).

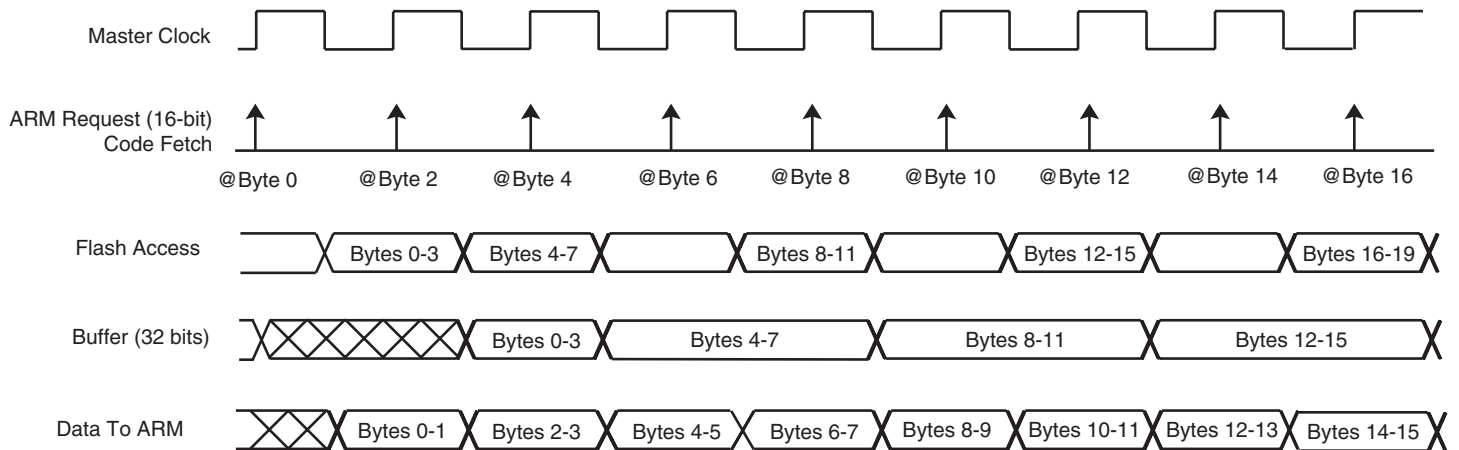
This optimization concerns only Code Fetch and not Data.

The read operations can be performed with or without wait state. Up to 3 wait states can be programmed in the field FWS (Flash Wait State) in the Flash Mode Register MC\_FMR (see ["MC Flash Mode Register" on page 110](#)). Defining FWS to be 0 enables the single-cycle access of the embedded Flash.

The Flash memory is accessible through 8-, 16- and 32-bit reads.

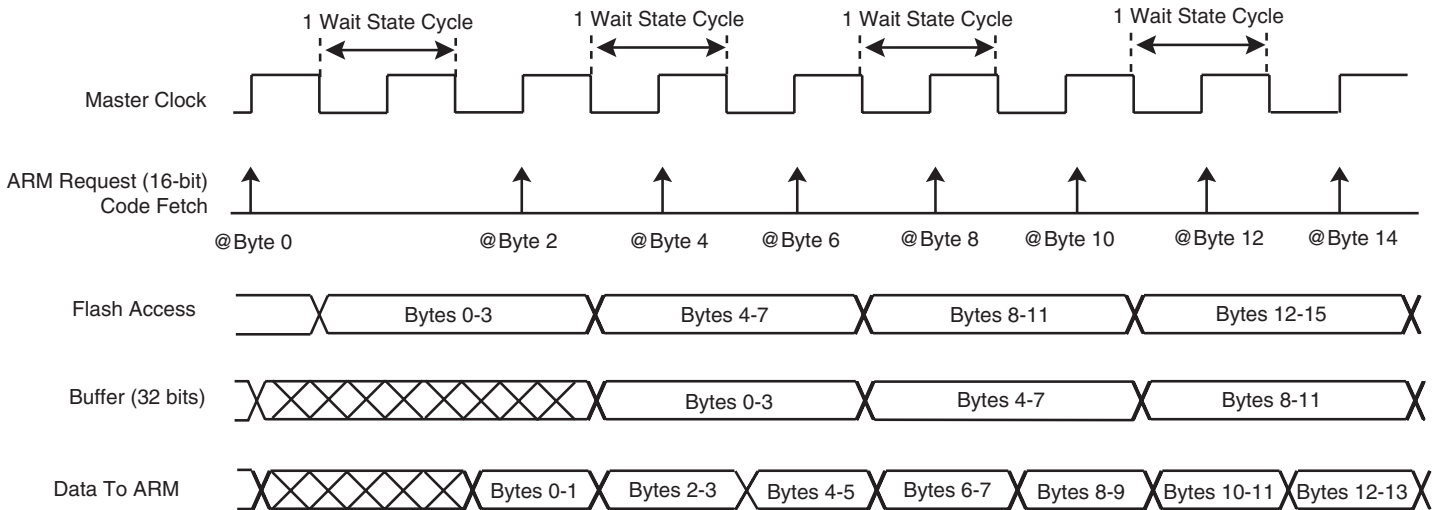
As the Flash block size is smaller than the address space reserved for the internal memory area, the embedded Flash wraps around the address space and appears to be repeated within it.

**Figure 20-2.** Code Read Optimization in Thumb Mode for FWS = 0



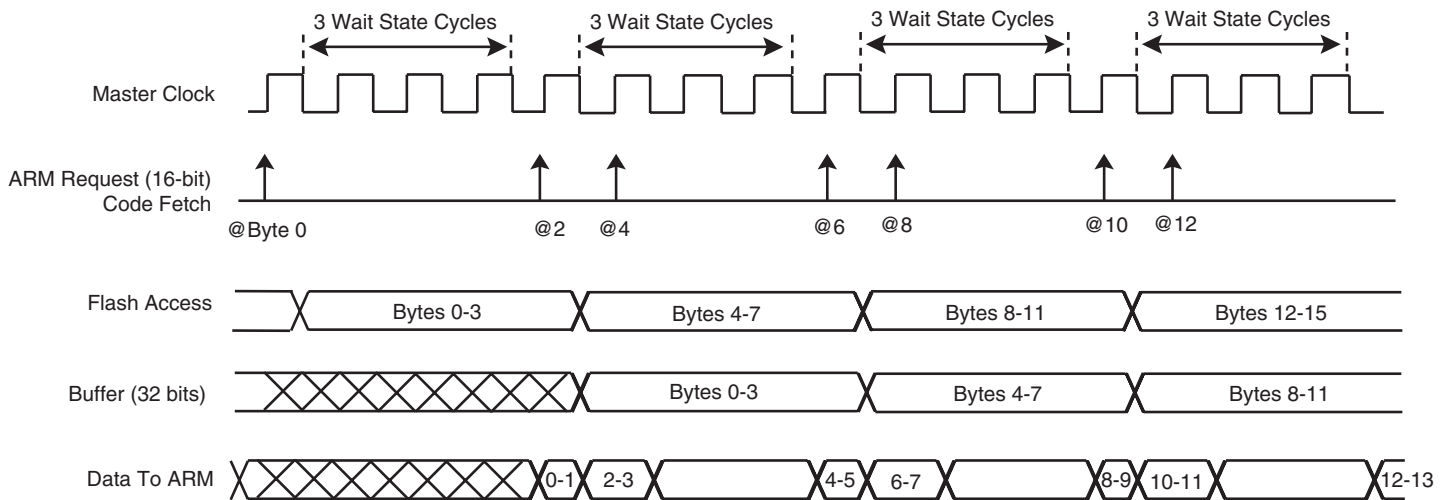
Note: When FWS is equal to 0, all accesses are performed in a single-cycle access.

**Figure 20-3.** Code Read Optimization in Thumb Mode for FWS = 1



Note: When FWS is equal to 1, in case of sequential reads, all the accesses are performed in a single-cycle access (except for the first one).

**Figure 20-4.** Code Read Optimization in Thumb Mode for FWS = 3



**Note:** When FWS is equal to 2 or 3, in case of sequential reads, the first access takes FWS cycles, the second access one cycle, the third access FWS cycles, the fourth access one cycle, etc.

### 20.2.3 Write Operations

The internal memory area reserved for the Embedded Flash can also be written through a write-only latch buffer. Write operations take into account only the 8 lowest address bits and thus wrap around within the internal memory area address space and appear to be repeated 1024 times within it.

Write operations might be prevented by programming the Memory Protection Unit of the product.

Writing of 8-bit and 16-bit data is not allowed and may lead to unpredictable data corruption.

Write operations are performed in the number of wait states equal to the number of wait states for read operations + 1, except for FWS = 3 (see ["MC Flash Mode Register" on page 110](#)).

### 20.2.4 Flash Commands

The Embedded Flash Controller offers a command set to manage programming the memory flash, locking and unlocking lock regions, consecutive programming and locking, and full Flash erasing.

**Table 20-1.** Set of Commands

Command	Value	Mnemonic
Write page	0x01	WP
Set Lock Bit	0x02	SLB
Write Page and Lock	0x03	WPL
Clear Lock Bit	0x04	CLB
Erase all	0x08	EA

In order to perform one of these commands, the Flash Command Register (MC\_FCR) has to be written with the correct command using to the field FCMD (see ["MC Flash Command Register" on page 112](#)).



All the commands are protected by the same keyword, which has to be written in the eight highest bits of the MC\_FCR register.

Writing MC\_FCR with data that does not contain the correct key and/or with an invalid command has no effect on the memory plane; however, the PROGE flag is set in the MC\_FSR register. This flag is automatically cleared by a read access to the MC\_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; however, the LOCKE flag is set in the MC\_FSR register. This flag is automatically cleared by a read access to the MC\_FSR register.

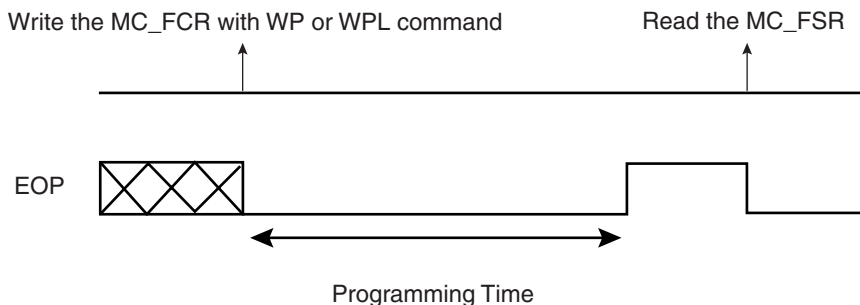
In order to guarantee valid operations on the Flash memory, the field Flash Microsecond Cycle Number (FMCN) in the Flash Mode Register MC\_FMR must be correctly programmed (see ["MC Flash Mode Register" on page 110](#)).

## 20.2.4.1 Programming

The programming is done by writing data into the latch buffer and then triggering a programming command that corresponds to the **Write Page Command (WP)** in the Flash Command Register MC\_FCR. The sequence is as follows:

- Write the full page, at any page address, within the internal memory area address space using only 32-bit access.
- If not already done, set the bit EOP (End of Programming) in the Flash Mode Register, depending on whether an interrupt is required or not at the end of programming.
- Write in the field PAGEN of the Flash Command Register (MC\_FCR) the Page Number to be programmed.
- Clear the bit NEBP (No Erase Before Programming) in MC\_FMR, if an erase before programming is required.
- Start the programming by writing the Flash Command Register with the Write Page Command.
- The page defined by PAGEN is first erased if the bit NEBP is set to 0 and then programmed with the data written in the buffer.
- When the programming completes, the bit EOP in the Flash Programming Status Register raises. If an interrupt has been enabled by setting the bit EOP in MC\_FMR, the interrupt line of the Memory Controller is activated.

**Figure 20-5.** State of the EOP Bit in MC\_FSR



When the software reads the Flash Status Register (MC\_FSR), the EOP bit is automatically cleared and the interrupt line is deactivated.

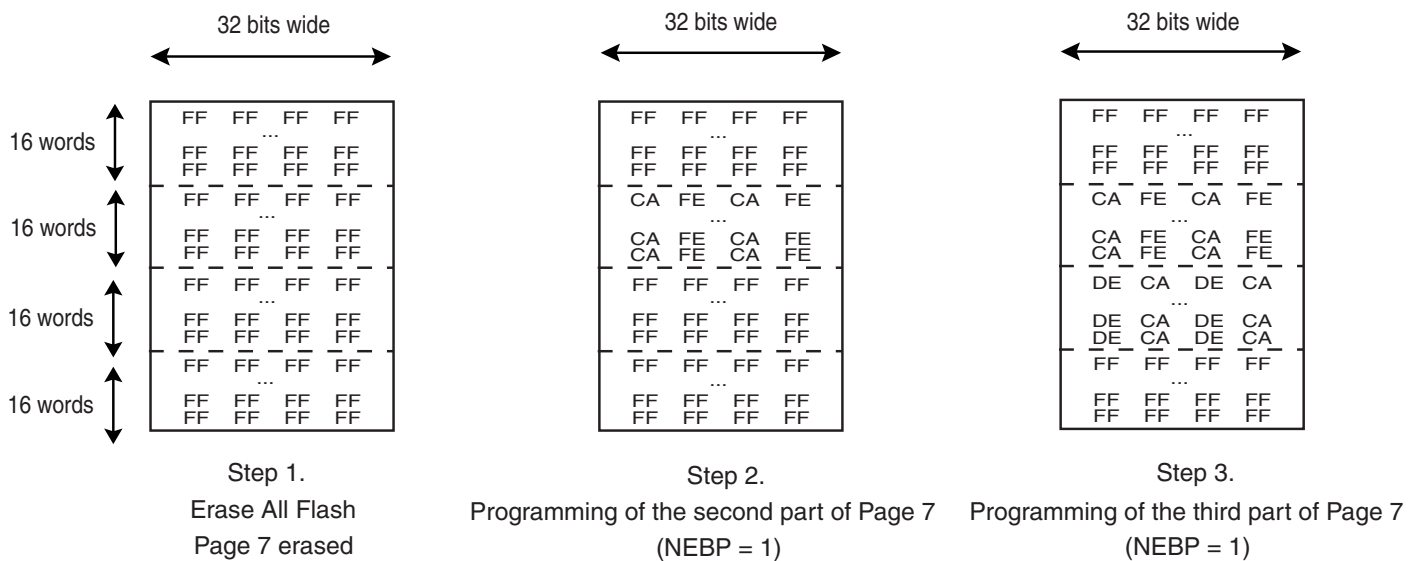
Two errors can be detected in the MC\_FSR register after a programming sequence:

- Programming Error: A bad keyword and/or an invalid command have been written in the MC\_FCR register.
- Lock Error: The page to be programmed belongs to a locked region. A command must be previously run to unlock the corresponding region.

The Flash technology requires that an erase must be done before programming. The entire memory plane can be erased at the same time, or a page can be automatically erased by clearing the NEBP bit in the MC\_FMR register before writing the command in the MC\_FCR register.

By setting the NEBP bit in the MC\_FMR register, a page can be programmed in several steps if it has been erased before (see Figure 20-6).

**Figure 20-6.** Example of Partial Page Programming:



#### 20.2.4.2 Lock and Unlock Operations

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.

Each lock region has its own lock bit that is readable in the highest bits of the Flash Status Register (MC\_FSR).

After production, the device may have some embedded Flash lock regions locked. These locked regions are reserved for a default application. Refer to the product definition section for the default embedded Flash mapping. Locked lock regions can be unlocked to be erased and then programmed with another application or other data.

The lock and unlock commands are performed by defining the PAGEN field and by writing the appropriate command (**Set Lock Bit Command (SLB)** or **Clear Lock Bit Command (CLB)**) in the Flash Command Register (MC\_FCR). PAGEN defines one page number of the lock region to be locked or unlocked. Writing in all the other bits of PAGEN has no effect.

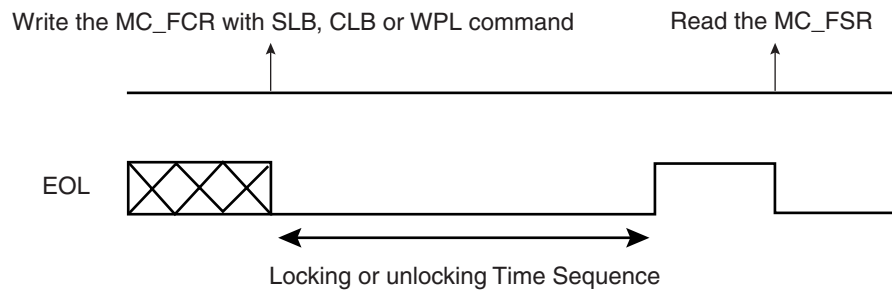
The Clear Lock Bit command programs the lock bit to 1; the corresponding bit LOCKSx in MC\_FSR reads 0. The Set Lock Bit command programs the lock bit to 0; the corresponding bit LOCKSx in MC\_FSR reads 1.

When the Set Lock Bit or Clear Lock Bit command is triggered, the programming or erasing operation of the lock bit is performed. When it completes, the bit EOL is set.

No access to the Flash is permitted when a Set Lock Bit or Clear Lock Bit command is performed.

A programming error, where a bad keyword and/or an invalid command have been written in the MC\_FCR register, may be detected in the MC\_FSR register after a programming sequence.

**Figure 20-7.** State of the EOL Bit in MC\_FSR



### 20.2.4.3 Lock Protection

When a programming command is performed with PAGEN defining a locked lock region, the bit LOCKE in MC\_FSR rises. If the bit LOCKE has been written at 1 in MC\_FMR, the interrupt line rises. Reading MC\_FSR automatically clears the bit LOCKE in MC\_FSR and thus deactivates the interrupt line.

### 20.2.4.4 Write Page and Lock

The user can perform consecutively the programming of the page and the lock of the lock region (**Write Page and Lock Command (WPL)** in the FCMD field of the Flash Command Register MC\_FCR), both defined by PAGEN.

Only one or both end of programming or end of lock interrupts may be enabled to trigger an interrupt when the operations completes.

### 20.2.4.5 Erase All Flash

The entire memory can be erased if the **Erase All Command (EA)** in the Flash Command Register MC\_FCR is written.

Erase All operation is allowed only if there are no lock bits set. Thus, if at least one lock region is locked, the bit LOCKE in MC\_FSR rises and the command is cancelled. If the bit LOCKE has been written at 1 in MC\_FMR, the interrupt line rises (see ["Lock Protection" on page 107](#)).

If not already done, set the bit EOP (End of Programming) in the Flash Mode Register, depending on whether an interrupt is required or not at the end of the erase.

When the Flash erase is complete, the bit EOP in the Flash Programming Status Register rises. If an interrupt has been enabled by setting the bit EOP in MC\_FMR, the interrupt line of the Memory Controller is activated.



When the software reads the Flash Status Register (MC\_FSR), the EOP bit is automatically cleared and the interrupt line is deactivated.

Two errors can be detected in the MC\_FSR register after a programming sequence:

- Programming Error: A bad keyword and/or an invalid command have been written in the MC\_FCR register.
- Lock Error: At least one lock region to be erased is protected. The erase command has been refused and no page has been erased. A Clear Lock Bit command must be executed previously to unlock the corresponding lock regions.

### 20.3 Embedded Flash Controller (EFC) User Interface

The User Interface of the Embedded Flash Controller is integrated within the Memory Controller with Base Address: 0xFFFF FF00.

**Table 20-2.** Embedded Flash Controller (EFC) Register Mapping

Offset	Register	Name	Access	Reset State
0x60	MC Flash Mode Register	MC_FMR	Read/Write	0x0
0x64	MC Flash Command Register	MC_FCR	Write-only	–
0x68	MC Flash Status Register	MC_FSR	Read-only	–
0x6C	Reserved	–	–	–

### 20.3.1 MC Flash Mode Register

**Register Name:** MC\_FMR  
**Access Type:** Read/Write  
**Offset:** 0x60

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

- **EOP: End of Programming Interrupt Enable**

0: End of Programming (page programming or erase all flash) does not generate an interrupt.

1: End of Programming (page programming or erase all flash) generates an interrupt.

- **EOL: End of Lock/Unlock Interrupt Enable**

0: End of Lock or End of Unlock does not generate an interrupt.

1: End of Lock or End of Unlock generates an interrupt.

- **LOCKE: Lock Error Interrupt Enable**

0: Lock Error does not generate an interrupt.

1: Lock Error generates an interrupt.

- **PROGE: Programming Error Interrupt Enable**

0: Programming Error does not generate an interrupt.

1: Programming Error generates an interrupt.

- **NEBP: No Erase Before Programming**

0: A page erase is performed before programming.

1: No erase is performed before programming.

- **FWS: Flash Wait State**

This field defines the number of wait states for read and write operations:

FWS	Read Operations	Write Operations
0	1 cycle	2 cycles
1	2 cycles	3 cycles
2	3 cycles	4 cycles
3	4 cycles	4 cycles

- **FMCN: Flash Microsecond Cycle Number**

Before writing Lock bits, this field must be set to the number of Master Clock cycles in one hundred nanoseconds.

When writing the rest of the Flash, this field defines the number of Master Clock cycles in 1.5 microseconds. This number must be rounded up.

**Warning:** The value 0 is only allowed for a Master Clock period superior to 30 microseconds.

**Warning:** In order to guarantee valid operations on the Flash memory, the field Flash Microsecond Cycle Number (FMCN) **must be** correctly programmed.

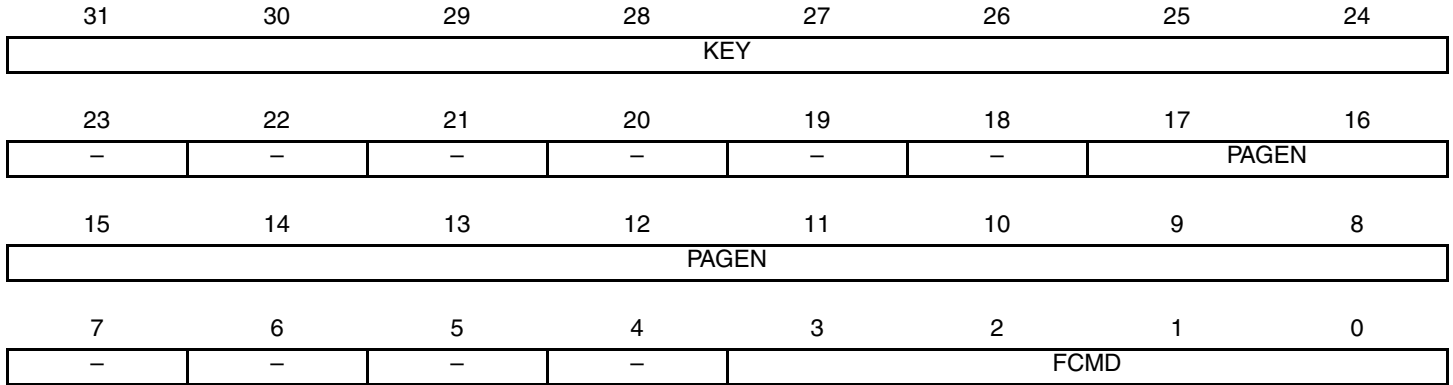


### 20.3.2 MC Flash Command Register

**Register Name:** MC\_FCR

**Access Type:** Write only

**Offset:** 0x64



• **FCMD: Flash Command**

This field defines the Flash commands:

FCMD	Operations
0000	No command. Does not raise the Programming Error Status flag in the Flash Status Register MC_FSR.
0001	Write Page Command (WP): Starts the programming of the page specified in the PAGEN field.
0010	Set Lock Bit Command (SLB): Starts a set lock bit sequence of the lock region specified in the PAGEN field.
0011	Write Page and Lock Command (WPL): The lock sequence of the lock region associated with the page specified in the field PAGEN occurs automatically after completion of the programming sequence.
0100	Clear Lock Bit Command (CLB): Starts a clear lock bit sequence of the lock region specified in the PAGEN field.
1000	Erase All Command (EA): Starts the erase of the entire Flash. If at least one page is locked, the command is cancelled.
Others	Reserved. Raises the Programming Error Status flag in the Flash Status Register MC_FSR.



- **PAGEN: Page Number**

Command	PAGEN Description
Write Page Command	PAGEN defines the page number to be written.
Write Page and Lock Command	PAGEN defines the page number to be written and its associated lock region.
Erase All Command	This field is meaningless
Set/Clear Lock Bit Command	PAGEN defines one page number of the lock region to be locked or unlocked.

Note: Depending on the command, all the possible unused bits of PAGEN are meaningless.

- **KEY: 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 actually not performed and no action is started.

### 20.3.3 MC Flash Status Register

**Register Name:** MC\_FSR  
**Access Type:** Read only  
**Offset:** 0x68

31	30	29	28	27	26	25	24
LOCKS15	LOCKS14	LOCKS13	LOCKS12	LOCKS11	LOCKS10	LOCKS9	LOCKS8
23	22	21	20	19	18	17	16
LOCKS7	LOCKS6	LOCKS5	LOCKS4	LOCKS3	LOCKS2	LOCKS1	LOCKS0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	PROGE	LOCKE	EOL	EOP

- **EOP: End of Programming Status**

0: The programming sequence (page programming or erase all Flash) triggered by the last write in MC\_FCR is not yet completed, or FMC\_FSR has been read.

1: The programming sequence (page programming or erase all Flash) triggered by the last write in MC\_FCR is completed and MC\_FSR has not been read yet.

- **EOL: End of Lock Status**

0: The lock or unlock sequence triggered by the last write in MC\_FCR is not yet completed, or FMC\_FSR has been read.

1: The lock or unlock sequence triggered by the last write in MC\_FCR is completed and MC\_FSR has not been read yet.

- **LOCKE: Lock Error Status**

0: No programming of at least one locked lock region has happened since the last read of MC\_FSR.

1: Programming of at least one locked lock region has happened since the last read of MC\_FSR.

- **PROGE: Programming Error Status**

0: No invalid commands and no bad key-words were written in the Flash Command Register MC\_FCR.

1: An invalid command and/or a bad key-word was/were written in the Flash Command Register MC\_FCR.

- **LOCKSx: Lock Region x Lock Status**

0: The corresponding lock region is not locked.

1: The corresponding lock region is locked.

## 21. Peripheral DMA Controller (PDC)

### 21.1 Overview

The Peripheral DMA Controller (PDC) transfers data between on-chip serial peripherals such as the UART, USART, SSC, SPI, MCI and the on- and off-chip memories. Using the Peripheral DMA Controller avoids processor intervention and removes the processor interrupt-handling overhead. This significantly reduces the number of clock cycles required for a data transfer and, as a result, improves the performance of the microcontroller and makes it more power efficient.

The PDC channels are implemented in pairs, each pair being dedicated to a particular peripheral. One channel in the pair is dedicated to the receiving channel and one to the transmitting channel of each UART, USART, SSC and SPI.

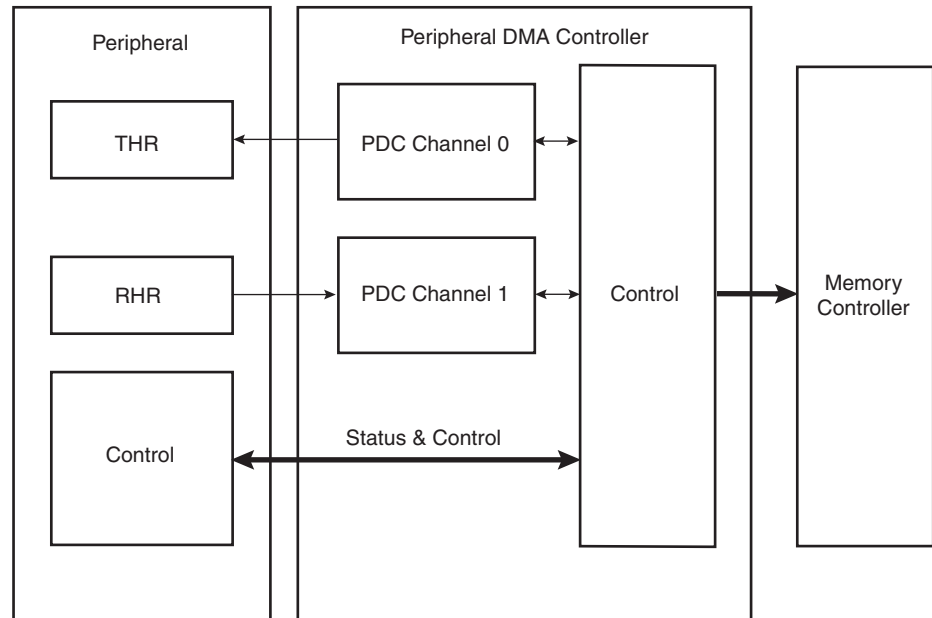
The user interface of a PDC channel is integrated in the memory space of each peripheral. It contains:

- two 32-bit memory pointer registers (send and receive)
- two 16-bit transfer count registers (send and receive)
- two 32-bit registers for next memory pointer (send and receive)
- two 16-bit registers for next transfer count (send and receive)

The peripheral triggers PDC transfers using transmit and receive signals. When the programmed data is transferred, an end of transfer interrupt is generated by the corresponding peripheral.

### 21.2 Block Diagram

Figure 21-1. Block Diagram



## 21.3 Functional Description

### 21.3.1 Configuration

The PDC channels user interface enables the user to configure and control the data transfers for each channel. The user interface of a PDC channel is integrated into the user interface of the peripheral (offset 0x100), which it is related to.

Per peripheral, it contains four 32-bit Pointer Registers (RPR, RNPR, TPR, and TNPR) and four 16-bit Counter Registers (RCR, RNCr, TCR, and TNCR).

The size of the buffer (number of transfers) is configured in an internal 16-bit transfer counter register, and it is possible, at any moment, to read the number of transfers left for each channel.

The memory base address is configured in a 32-bit memory pointer by defining the location of the first address to access in the memory. It is possible, at any moment, to read the location in memory of the next transfer and the number of remaining transfers. 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 peripheral status register. Transfers can be enabled and/or disabled by setting TXTEN/TXTDIS and RXTEN/RXTDIS in PDC Transfer Control Register. These control bits enable reading the pointer and counter registers safely without any risk of their changing between both reads.

The PDC sends status flags to the peripheral visible in its status-register (ENDRX, ENDTX, RXBUFF, and TXBUFE).

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.

### 21.3.2 Memory Pointers

Each peripheral is connected to the PDC by a receiver data channel and a transmitter data channel. Each channel has an internal 32-bit memory pointer. Each memory pointer points to a location anywhere in the memory space (on-chip memory or external bus interface memory).

Depending on the type of transfer (byte, half-word or word), the memory pointer is incremented by 1, 2 or 4, respectively for peripheral transfers.

If a memory pointer is reprogrammed while the PDC is in operation, the transfer address is changed, and the PDC performs transfers using the new address.

### 21.3.3 Transfer Counters

There is one internal 16-bit transfer counter for each channel used to count the size of the block already transferred by its associated channel. These counters are decremented after each data transfer. When the counter reaches zero, the transfer is complete and the PDC stops transferring data.

If the Next Counter Register is equal to zero, the PDC disables the trigger while activating the related peripheral end flag.

If the counter is reprogrammed while the PDC is operating, the number of transfers is updated and the PDC counts transfers from the new value.

Programming the Next Counter/Pointer registers chains the buffers. The counters are decremented after each data transfer as stated above, but when the transfer counter reaches zero, the values of the Next Counter/Pointer are loaded into the Counter/Pointer registers in order to re-enable the triggers.

For each channel, two status bits indicate the end of the current buffer (ENDRX, ENDTX) and the end of both current and next buffer (RXBUFF, TXBUFE). These bits are directly mapped to the peripheral status register and can trigger an interrupt request to the AIC.

The peripheral end flag is automatically cleared when one of the counter-registers (Counter or Next Counter Register) is written.

Note: When the Next Counter Register is loaded into the Counter Register, it is set to zero.

#### 21.3.4 Data Transfers

The peripheral triggers PDC transfers using transmit (TXRDY) and receive (RXRDY) signals.

When the peripheral receives an external character, it sends a Receive Ready signal to the PDC which then requests access to the system bus. When access is granted, the PDC starts a read of the peripheral Receive Holding Register (RHR) and then triggers a write in the memory.

After each transfer, the relevant PDC memory pointer is incremented and the number of transfers left is decremented. When the memory block size is reached, a signal is sent to the peripheral and the transfer stops.

The same procedure is followed, in reverse, for transmit transfers.

#### 21.3.5 Priority of PDC Transfer Requests

The Peripheral DMA Controller handles transfer requests from the channel according to priorities fixed for each product. These priorities are defined in the product datasheet.

If simultaneous requests of the same type (receiver or transmitter) occur on identical peripherals, the priority is determined by the numbering of the peripherals.

If transfer requests are not simultaneous, they are treated in the order they occurred. Requests from the receivers are handled first and then followed by transmitter requests.

## 21.4 Peripheral DMA Controller (PDC) User Interface

**Table 21-1.** Register Mapping

Offset	Register	Register Name	Read/Write	Reset
0x100	Receive Pointer Register	PERIPH <sup>(1)</sup> _RPR	Read/Write	0x0
0x104	Receive Counter Register	PERIPH_RCR	Read/Write	0x0
0x108	Transmit Pointer Register	PERIPH_TPR	Read/Write	0x0
0x10C	Transmit Counter Register	PERIPH_TCR	Read/Write	0x0
0x110	Receive Next Pointer Register	PERIPH_RNPR	Read/Write	0x0
0x114	Receive Next Counter Register	PERIPH_RNCR	Read/Write	0x0
0x118	Transmit Next Pointer Register	PERIPH_TNPR	Read/Write	0x0
0x11C	Transmit Next Counter Register	PERIPH_TNCR	Read/Write	0x0
0x120	PDC Transfer Control Register	PERIPH_PTCR	Write-only	-
0x124	PDC Transfer Status Register	PERIPH_PTSR	Read-only	0x0

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 peripheral desired (DBGU, USART, SSC, SPI, MCI etc).

## 21.4.1 PDC Receive Pointer Register

**Register Name:** PERIPH\_RPR

**Access Type:** 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 Address**

Address of the next receive transfer.

## 21.4.2 PDC Receive Counter Register

**Register Name:** PERIPH\_RCR

**Access Type:** Read/Write

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

- **RXCTR: Receive Counter Value**

Number of receive transfers to be performed.

### 21.4.3 PDC Transmit Pointer Register

**Register Name:** PERIPH\_TPR

**Access Type:** 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 Pointer Address**

Address of the transmit buffer.

### 21.4.4 PDC Transmit Counter Register

**Register Name:** PERIPH\_TCR

**Access Type:** Read/Write

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

- **TXCTR: Transmit Counter Value**

TXCTR is the size of the transmit transfer to be performed. At zero, the peripheral data transfer is stopped.



## 21.4.5 PDC Receive Next Pointer Register

**Register Name:** PERIPH\_RNPR

**Access Type:** 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 Address**

RXNPTR is the address of the next buffer to fill with received data when the current buffer is full.

## 21.4.6 PDC Receive Next Counter Register

**Register Name:** PERIPH\_RNCR

**Access Type:** Read/Write

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

- **RXNCR: Receive Next Counter Value**

RXNCR is the size of the next buffer to receive.



### 21.4.7 PDC Transmit Next Pointer Register

Register Name: PERIPH\_TNPR

Access Type: 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 Address**

TXNPTR is the address of the next buffer to transmit when the current buffer is empty.

### 21.4.8 PDC Transmit Next Counter Register

Register Name: PERIPH\_TNCR

Access Type: Read/Write

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

- **TXNCR: Transmit Next Counter Value**

TXNCR is the size of the next buffer to transmit.

## 21.4.9 PDC Transfer Control Register

**Register Name:** PERIPH\_PTCR

**Access Type:** Write-only

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

- **RXTEN: Receiver Transfer Enable**

0 = No effect.

1 = Enables the receiver PDC transfer requests if RXTDIS is not set.

- **RXTDIS: Receiver Transfer Disable**

0 = No effect.

1 = Disables the receiver PDC transfer requests.

- **TXTEN: Transmitter Transfer Enable**

0 = No effect.

1 = Enables the transmitter PDC transfer requests.

- **TXTDIS: Transmitter Transfer Disable**

0 = No effect.

1 = Disables the transmitter PDC transfer requests

### 21.4.10 PDC Transfer Status Register

**Register Name:** PERIPH\_PTSR

**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
–	–	–	–	–	–	–	TXTEN
7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	RXTEN

- **RXTEN: Receiver Transfer Enable**

0 = Receiver PDC transfer requests are disabled.

1 = Receiver PDC transfer requests are enabled.

- **TXTEN: Transmitter Transfer Enable**

0 = Transmitter PDC transfer requests are disabled.

1 = Transmitter PDC transfer requests are enabled.

## 22. Advanced Interrupt Controller (AIC)

### 22.1 Overview

The Advanced Interrupt Controller (AIC) is an 8-level priority, individually maskable, vectored interrupt controller, providing handling of up to thirty-two interrupt sources. It is designed to substantially reduce the software and real-time overhead in handling internal and external interrupts.

The AIC drives the nFIQ (fast interrupt request) and the nIRQ (standard interrupt request) inputs of an ARM processor. Inputs of the AIC are either internal peripheral interrupts or external interrupts coming from the product's pins.

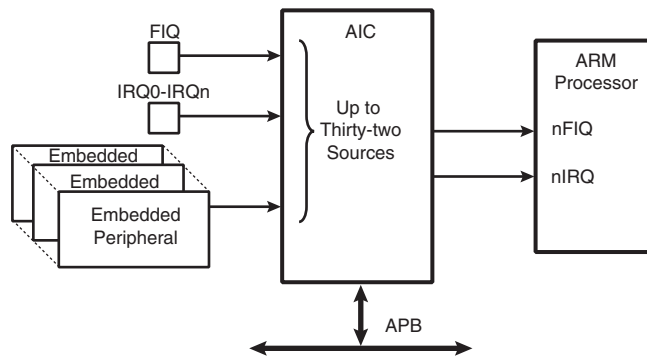
The 8-level Priority Controller allows the user to define the priority for each interrupt source, thus permitting higher priority interrupts to be serviced even if a lower priority interrupt is being treated.

Internal interrupt sources can be programmed to be level sensitive or edge triggered. External interrupt sources can be programmed to be positive-edge or negative-edge triggered or high-level or low-level sensitive.

The fast forcing feature redirects any internal or external interrupt source to provide a fast interrupt rather than a normal interrupt.

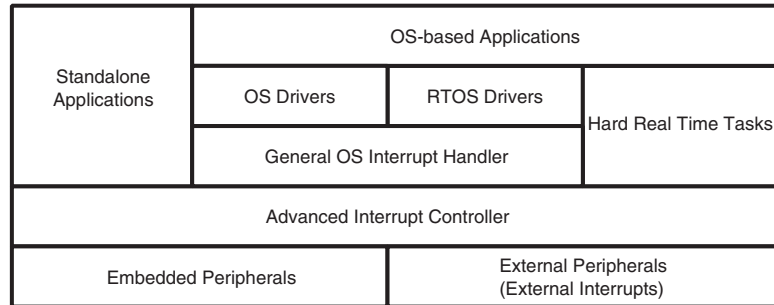
### 22.2 Block Diagram

Figure 22-1. Block Diagram



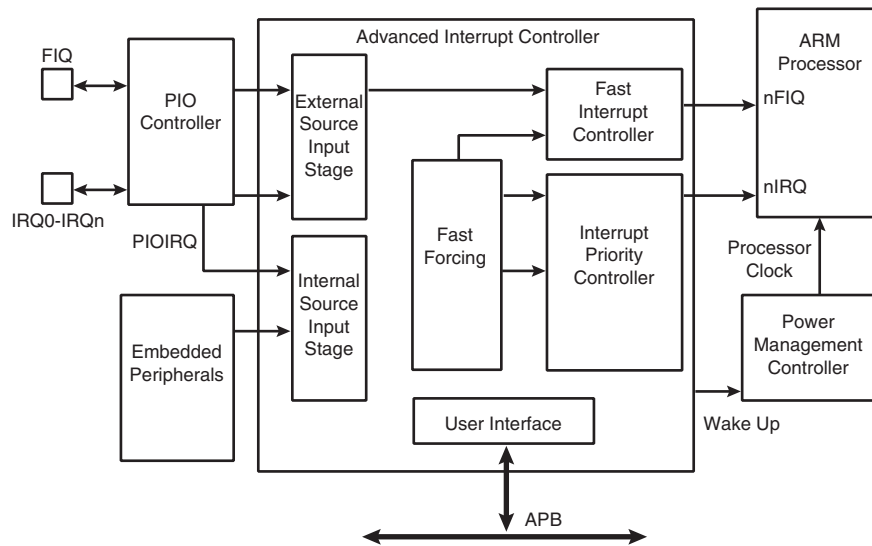
## 22.3 Application Block Diagram

Figure 22-2. Description of the Application Block



## 22.4 AIC Detailed Block Diagram

Figure 22-3. AIC Detailed Block Diagram



## 22.5 I/O Line Description

Table 22-1. I/O Line Description

Pin Name	Pin Description	Type
FIQ	Fast Interrupt	Input
IRQ0 - IRQn	Interrupt 0 - Interrupt n	Input

## 22.6 Product Dependencies

### 22.6.1 I/O Lines

The interrupt signals FIQ and IRQ0 to IRQn are normally multiplexed through the PIO controllers. Depending on the features of the PIO controller used in the product, the pins must be programmed in accordance with their assigned interrupt function. This is not applicable when the PIO controller used in the product is transparent on the input path.

### 22.6.2 Power Management

The Advanced Interrupt Controller is continuously clocked. The Power Management Controller has no effect on the Advanced Interrupt Controller behavior.

The assertion of the Advanced Interrupt Controller outputs, either nIRQ or nFIQ, wakes up the ARM processor while it is in Idle Mode. The General Interrupt Mask feature enables the AIC to wake up the processor without asserting the interrupt line of the processor, thus providing synchronization of the processor on an event.

### 22.6.3 Interrupt Sources

The Interrupt Source 0 is always located at FIQ. If the product does not feature an FIQ pin, the Interrupt Source 0 cannot be used.

The Interrupt Source 1 is always located at System Interrupt. This is the result of the OR-wiring of the system peripheral interrupt lines, such as the System Timer, the Real Time Clock, the Power Management Controller and the Memory Controller. When a system interrupt occurs, the service routine must first distinguish the cause of the interrupt. This is performed by reading successively the status registers of the above mentioned system peripherals.

The interrupt sources 2 to 31 can either be connected to the interrupt outputs of an embedded user peripheral or to external interrupt lines. The external interrupt lines can be connected directly, or through the PIO Controller.

The PIO Controllers are considered as user peripherals in the scope of interrupt handling. Accordingly, the PIO Controller interrupt lines are connected to the Interrupt Sources 2 to 31.

The peripheral identification defined at the product level corresponds to the interrupt source number (as well as the bit number controlling the clock of the peripheral). Consequently, to simplify the description of the functional operations and the user interface, the interrupt sources are named FIQ, SYS, and PID2 to PID31.

## 22.7 Functional Description

### 22.7.1 Interrupt Source Control

#### 22.7.1.1 *Interrupt Source Mode*

The Advanced Interrupt Controller independently programs each interrupt source. The SRC-TYPE field of the corresponding AIC\_SMR (Source Mode Register) selects the interrupt condition of each source.

The internal interrupt sources wired on the interrupt outputs of the embedded peripherals can be programmed either in level-sensitive mode or in edge-triggered mode. The active level of the internal interrupts is not important for the user.

The external interrupt sources can be programmed either in high level-sensitive or low level-sensitive modes, or in positive edge-triggered or negative edge-triggered modes.

#### 22.7.1.2 *Interrupt Source Enabling*

Each interrupt source, including the FIQ in source 0, can be enabled or disabled by using the command registers; AIC\_IECR (Interrupt Enable Command Register) and AIC\_IDCR (Interrupt Disable Command Register). This set of registers conducts enabling or disabling in one instruction. The interrupt mask can be read in the AIC\_IMR register. A disabled interrupt does not affect servicing of other interrupts.

#### 22.7.1.3 *Interrupt Clearing and Setting*

All interrupt sources programmed to be edge-triggered (including the FIQ in source 0) can be individually set or cleared by writing respectively the AIC\_ISCR and AIC\_ICCR registers. Clearing or setting interrupt sources programmed in level-sensitive mode has no effect.

The clear operation is perfunctory, as the software must perform an action to reinitialize the “memorization” circuitry activated when the source is programmed in edge-triggered mode. However, the set operation is available for auto-test or software debug purposes. It can also be used to execute an AIC-implementation of a software interrupt.

The AIC features an automatic clear of the current interrupt when the AIC\_IVR (Interrupt Vector Register) is read. Only the interrupt source being detected by the AIC as the current interrupt is affected by this operation. (See “Priority Controller” on page 132.) The automatic clear reduces the operations required by the interrupt service routine entry code to reading the AIC\_IVR. Note that the automatic interrupt clear is disabled if the interrupt source has the Fast Forcing feature enabled as it is considered uniquely as a FIQ source. (For further details, See “Fast Forcing” on page 136.)

The automatic clear of the interrupt source 0 is performed when AIC\_FVR is read.

#### 22.7.1.4 *Interrupt Status*

For each interrupt, the AIC operation originates in AIC\_IPR (Interrupt Pending Register) and its mask in AIC\_IMR (Interrupt Mask Register). AIC\_IPR enables the actual activity of the sources, whether masked or not.

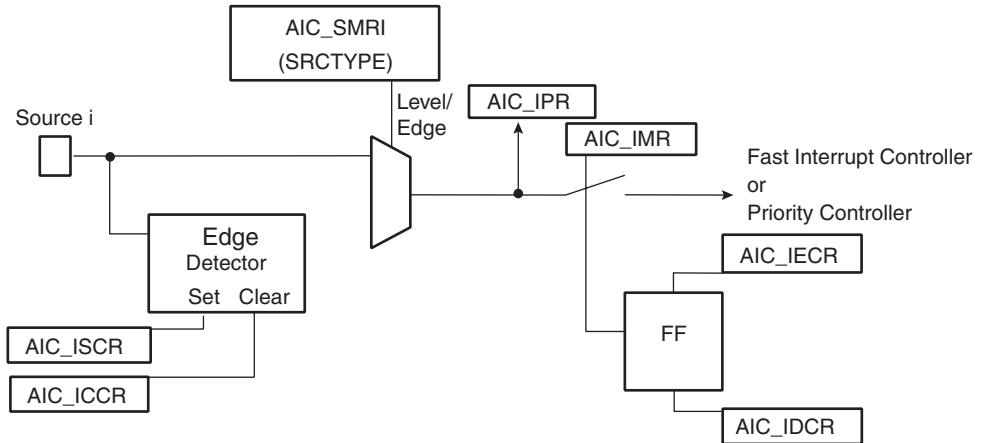
The AIC\_ISR register reads the number of the current interrupt (see “Priority Controller” on page 132) and the register AIC\_CISR gives an image of the signals nIRQ and nFIQ driven on the processor.

Each status referred to above can be used to optimize the interrupt handling of the systems.



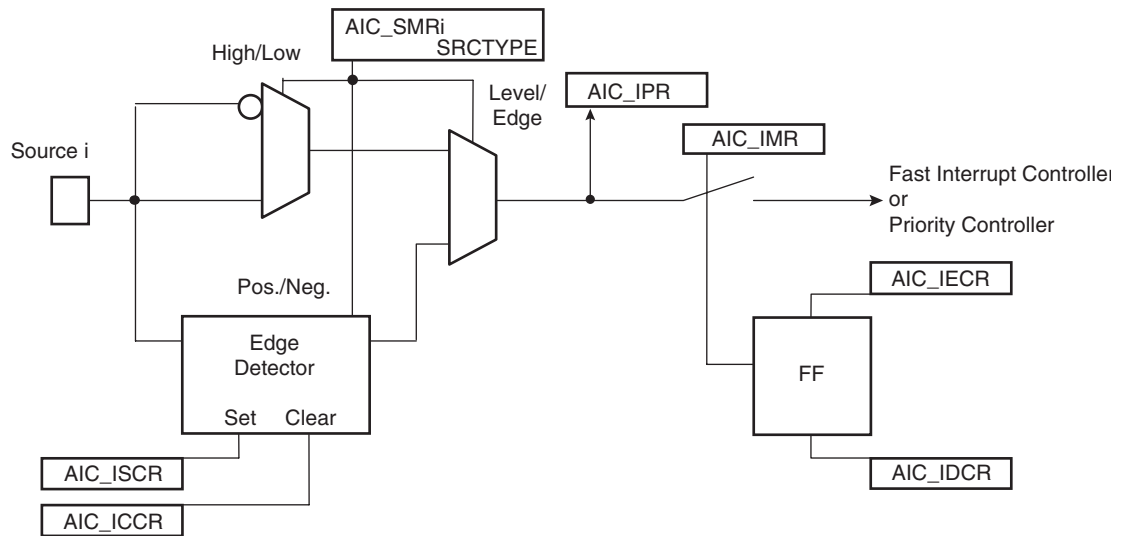
## 22.7.1.5 Internal Interrupt Source Input Stage

**Figure 22-4.** Internal Interrupt Source Input Stage



## 22.7.1.6 External Interrupt Source Input Stage

**Figure 22-5.** External Interrupt Source Input Stage



## 22.7.2 Interrupt Latencies

Global interrupt latencies depend on several parameters, including:

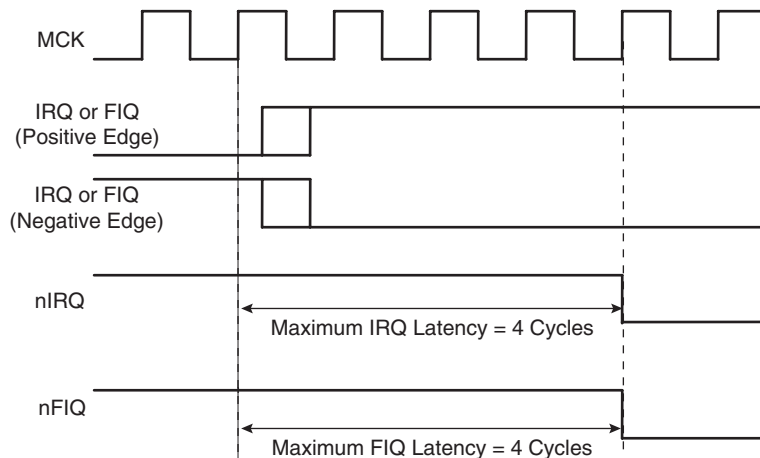
- The time the software masks the interrupts.
- Occurrence, either at the processor level or at the AIC level.
- The execution time of the instruction in progress when the interrupt occurs.
- The treatment of higher priority interrupts and the resynchronization of the hardware signals.

This section addresses only the hardware resynchronizations. It gives details of the latency times between the event on an external interrupt leading in a valid interrupt (edge or level) or the assertion of an internal interrupt source and the assertion of the nIRQ or nFIQ line on the processor. The resynchronization time depends on the programming of the interrupt source and on its type (internal or external). For the standard interrupt, resynchronization times are given assuming there is no higher priority in progress.

The PIO Controller multiplexing has no effect on the interrupt latencies of the external interrupt sources.

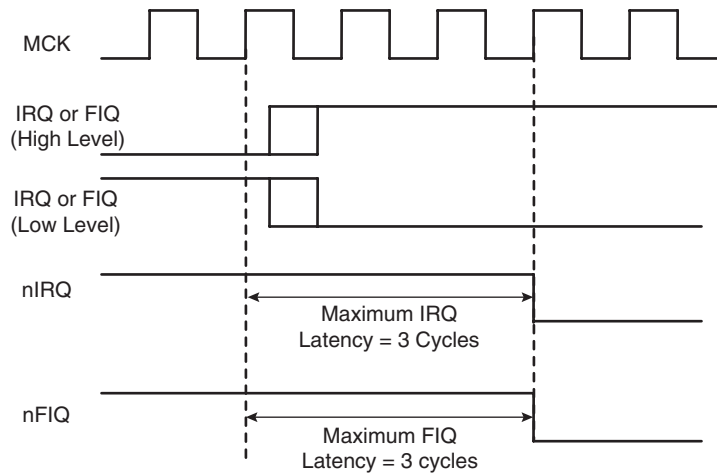
### 22.7.2.1 External Interrupt Edge Triggered Source

**Figure 22-6.** External Interrupt Edge Triggered Source



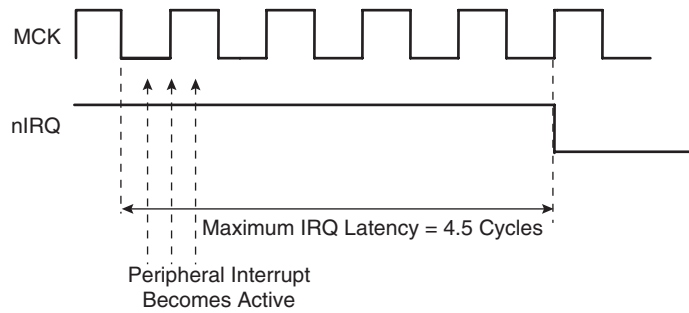
## 22.7.2.2 External Interrupt Level Sensitive Source

**Figure 22-7.** External Interrupt Level Sensitive Source



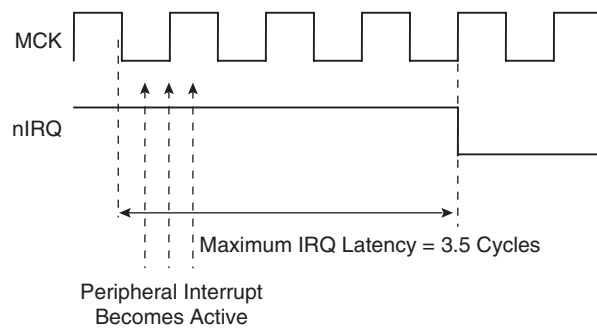
## 22.7.2.3 Internal Interrupt Edge Triggered Source

**Figure 22-8.** Internal Interrupt Edge Triggered Source



## 22.7.2.4 Internal Interrupt Level Sensitive Source

**Figure 22-9.** Internal Interrupt Level Sensitive Source



## 22.7.3 Normal Interrupt

### 22.7.3.1 Priority Controller

An 8-level priority controller drives the nIRQ line of the processor, depending on the interrupt conditions occurring on the interrupt sources 1 to 31 (except for those programmed in Fast Forcing).

Each interrupt source has a programmable priority level of 7 to 0, which is user-definable by writing the PRIOR field of the corresponding AIC\_SMR (Source Mode Register). Level 7 is the highest priority and level 0 the lowest.

As soon as an interrupt condition occurs, as defined by the SRCTYPE field of the AIC\_SMR (Source Mode Register), the nIRQ line is asserted. As a new interrupt condition might have happened on other interrupt sources since the nIRQ has been asserted, the priority controller determines the current interrupt at the time the AIC\_IVR (Interrupt Vector Register) is read. **The read of AIC\_IVR is the entry point of the interrupt handling** which allows the AIC to consider that the interrupt has been taken into account by the software.

The current priority level is defined as the priority level of the current interrupt.

If several interrupt sources of equal priority are pending and enabled when the AIC\_IVR is read, the interrupt with the lowest interrupt source number is serviced first.

The nIRQ line can be asserted only if an interrupt condition occurs on an interrupt source with a higher priority. If an interrupt condition happens (or is pending) during the interrupt treatment in progress, it is delayed until the software indicates to the AIC the end of the current service by writing the AIC\_EOICR (End of Interrupt Command Register). **The write of AIC\_EOICR is the exit point of the interrupt handling.**

### 22.7.3.2 Interrupt Nesting

The priority controller utilizes interrupt nesting in order for the high priority interrupt to be handled during the service of lower priority interrupts. This requires the interrupt service routines of the lower interrupts to re-enable the interrupt at the processor level.

When an interrupt of a higher priority happens during an already occurring interrupt service routine, the nIRQ line is re-asserted. If the interrupt is enabled at the core level, the current execution is interrupted and the new interrupt service routine should read the AIC\_IVR. At this time, the current interrupt number and its priority level are pushed into an embedded hardware stack, so that they are saved and restored when the higher priority interrupt servicing is finished and the AIC\_EOICR is written.

The AIC is equipped with an 8-level wide hardware stack in order to support up to eight interrupt nestings pursuant to having eight priority levels.

### 22.7.3.3 Interrupt Vectoring

The interrupt handler addresses corresponding to each interrupt source can be stored in the registers AIC\_SVR1 to AIC\_SVR31 (Source Vector Register 1 to 31). When the processor reads AIC\_IVR (Interrupt Vector Register), the value written into AIC\_SVR corresponding to the current interrupt is returned.

This feature offers a way to branch in one single instruction to the handler corresponding to the current interrupt, as AIC\_IVR is mapped at the absolute address 0xFFFF F100 and thus accessible from the ARM interrupt vector at address 0x0000 0018 through the following instruction:

```
LDR PC, [PC, # -&F20]
```

When the processor executes this instruction, it loads the read value in AIC\_IVR in its program counter, thus branching the execution on the correct interrupt handler.

This feature is often not used when the application is based on an operating system (either real time or not). Operating systems often have a single entry point for all the interrupts and the first task performed is to discern the source of the interrupt.

However, it is strongly recommended to port the operating system on AT91 products by supporting the interrupt vectoring. This can be performed by defining all the AIC\_SVR of the interrupt source to be handled by the operating system at the address of its interrupt handler. When doing so, the interrupt vectoring permits a critical interrupt to transfer the execution on a specific very fast handler and not onto the operating system's general interrupt handler. This facilitates the support of hard real-time tasks (input/outputs of voice/audio buffers and software peripheral handling) to be handled efficiently and independently of the application running under an operating system.

#### 22.7.3.4 Interrupt Handlers

This section gives an overview of the fast interrupt handling sequence when using the AIC. It is assumed that the programmer understands the architecture of the ARM processor, and especially the processor interrupt modes and the associated status bits.

It is assumed that:

1. The Advanced Interrupt Controller has been programmed, AIC\_SVR registers are loaded with corresponding interrupt service routine addresses and interrupts are enabled.
2. The instruction at the ARM interrupt exception vector address is required to work with the vectoring

```
LDR PC, [PC, # -&F20]
```

When nIRQ is asserted, if the bit "I" of CPSR is 0, the sequence is as follows:

1. The CPSR is stored in SPSR\_irq, the current value of the Program Counter is loaded in the Interrupt link register (R14\_irq) and the Program Counter (R15) is loaded with 0x18. In the following cycle during fetch at address 0x1C, the ARM core adjusts R14\_irq, decrementing it by four.
2. The ARM core enters Interrupt mode, if it has not already done so.
3. When the instruction loaded at address 0x18 is executed, the program counter is loaded with the value read in AIC\_IVR. Reading the AIC\_IVR has the following effects:
  - Sets the current interrupt to be the pending and enabled interrupt with the highest priority. The current level is the priority level of the current interrupt.
  - De-asserts the nIRQ line on the processor. Even if vectoring is not used, AIC\_IVR must be read in order to de-assert nIRQ.
  - Automatically clears the interrupt, if it has been programmed to be edge-triggered.
  - Pushes the current level and the current interrupt number on to the stack.
  - Returns the value written in the AIC\_SVR corresponding to the current interrupt.
4. The previous step has the effect of branching to the corresponding interrupt service routine. This should start by saving the link register (R14\_irq) and SPSR\_IRQ. The link register must be decremented by four when it is saved if it is to be restored

directly into the program counter at the end of the interrupt. For example, the instruction `SUB PC, LR, #4` may be used.

5. Further interrupts can then be unmasked by clearing the “I” bit in CPSR, allowing re-assertion of the nIRQ to be taken into account by the core. This can happen if an interrupt with a higher priority than the current interrupt occurs.
6. The interrupt handler can then proceed as required, saving the registers that will be used and restoring them at the end. During this phase, an interrupt of higher priority than the current level will restart the sequence from step 1.

Note: If the interrupt is programmed to be level sensitive, the source of the interrupt must be cleared during this phase.

7. The “I” bit in CPSR must be set in order to mask interrupts before exiting to ensure that the interrupt is completed in an orderly manner.
8. The End of Interrupt Command Register (AIC\_EOICR) must be written in order to indicate to the AIC that the current interrupt is finished. This causes the current level to be popped from the stack, restoring the previous current level if one exists on the stack. If another interrupt is pending, with lower or equal priority than the old current level but with higher priority than the new current level, the nIRQ line is re-asserted, but the interrupt sequence does not immediately start because the “I” bit is set in the core. SPSR\_irq is restored. Finally, the saved value of the link register is restored directly into the PC. This has the effect of returning from the interrupt to whatever was being executed before, and of loading the CPSR with the stored SPSR, masking or unmasking the interrupts depending on the state saved in SPSR\_irq.

Note: The “I” bit in SPSR is significant. If it is set, it indicates that the ARM core was on the verge of masking an interrupt when the mask instruction was interrupted. Hence, when SPSR is restored, the mask instruction is completed (interrupt is masked).

## 22.7.4 Fast Interrupt

### 22.7.4.1 Fast Interrupt Source

The interrupt source 0 is the only source which can raise a fast interrupt request to the processor except if fast forcing is used. The interrupt source 0 is generally connected to a FIQ pin of the product, either directly or through a PIO Controller.

### 22.7.4.2 Fast Interrupt Control

The fast interrupt logic of the AIC has no priority controller. The mode of interrupt source 0 is programmed with the AIC\_SMR0 and the field PRIOR of this register is not used even if it reads what has been written. The field SRCTYPE of AIC\_SMR0 enables programming the fast interrupt source to be positive-edge triggered or negative-edge triggered or high-level sensitive or low-level sensitive

Writing 0x1 in the AIC\_IECR (Interrupt Enable Command Register) and AIC\_IDCR (Interrupt Disable Command Register) respectively enables and disables the fast interrupt. The bit 0 of AIC\_IMR (Interrupt Mask Register) indicates whether the fast interrupt is enabled or disabled.

### 22.7.4.3 Fast Interrupt Vectoring

The fast interrupt handler address can be stored in AIC\_SVR0 (Source Vector Register 0). The value written into this register is returned when the processor reads AIC\_FVR (Fast Vector Register). This offers a way to branch in one single instruction to the interrupt handler, as AIC\_FVR is mapped at the absolute address 0xFFFF F104 and thus accessible from the ARM fast interrupt vector at address 0x0000 001C through the following instruction:

```
LDR PC, [PC, # -&F20]
```

When the processor executes this instruction it loads the value read in AIC\_FVR in its program counter, thus branching the execution on the fast interrupt handler. It also automatically performs the clear of the fast interrupt source if it is programmed in edge-triggered mode.

### 22.7.4.4 Fast Interrupt Handlers

This section gives an overview of the fast interrupt handling sequence when using the AIC. It is assumed that the programmer understands the architecture of the ARM processor, and especially the processor interrupt modes and associated status bits.

Assuming that:

1. The Advanced Interrupt Controller has been programmed, AIC\_SVR0 is loaded with the fast interrupt service routine address, and the interrupt source 0 is enabled.
2. The Instruction at address 0x1C (FIQ exception vector address) is required to vector the fast interrupt:

```
LDR PC, [PC, # -&F20]
```

3. The user does not need nested fast interrupts.

When nFIQ is asserted, if the bit "F" of CPSR is 0, the sequence is:

1. The CPSR is stored in SPSR\_fiq, the current value of the program counter is loaded in the FIQ link register (R14\_FIQ) and the program counter (R15) is loaded with 0x1C. In the following cycle, during fetch at address 0x20, the ARM core adjusts R14\_fiq, decrementing it by four.
2. The ARM core enters FIQ mode.
3. When the instruction loaded at address 0x1C is executed, the program counter is loaded with the value read in AIC\_FVR. Reading the AIC\_FVR has effect of automat-

- ically clearing the fast interrupt, if it has been programmed to be edge triggered. In this case only, it de-asserts the nFIQ line on the processor.
4. The previous step enables branching to the corresponding interrupt service routine. It is not necessary to save the link register R14\_fiq and SPSR\_fiq if nested fast interrupts are not needed.
  5. The Interrupt Handler can then proceed as required. It is not necessary to save registers R8 to R13 because FIQ mode has its own dedicated registers and the user R8 to R13 are banked. The other registers, R0 to R7, must be saved before being used, and restored at the end (before the next step). Note that if the fast interrupt is programmed to be level sensitive, the source of the interrupt must be cleared during this phase in order to de-assert the interrupt source 0.
  6. Finally, the Link Register R14\_fiq is restored into the PC after decrementing it by four (with instruction `SUB PC, LR, #4` for example). This has the effect of returning from the interrupt to whatever was being executed before, loading the CPSR with the SPSR and masking or unmasking the fast interrupt depending on the state saved in the SPSR.

Note: The “F” bit in SPSR is significant. If it is set, it indicates that the ARM core was just about to mask FIQ interrupts when the mask instruction was interrupted. Hence when the SPSR is restored, the interrupted instruction is completed (FIQ is masked).

Another way to handle the fast interrupt is to map the interrupt service routine at the address of the ARM vector 0x1C. This method does not use the vectoring, so that reading AIC\_FVR must be performed at the very beginning of the handler operation. However, this method saves the execution of a branch instruction.

#### 22.7.4.5 Fast Forcing

The Fast Forcing feature of the advanced interrupt controller provides redirection of any normal Interrupt source on the fast interrupt controller.

Fast Forcing is enabled or disabled by writing to the Fast Forcing Enable Register (AIC\_FFER) and the Fast Forcing Disable Register (AIC\_FFDR). Writing to these registers results in an update of the Fast Forcing Status Register (AIC\_FFSR) that controls the feature for each internal or external interrupt source.

When Fast Forcing is disabled, the interrupt sources are handled as described in the previous pages.

When Fast Forcing is enabled, the edge/level programming and, in certain cases, edge detection of the interrupt source is still active but the source cannot trigger a normal interrupt to the processor and is not seen by the priority handler.

If the interrupt source is programmed in level-sensitive mode and an active level is sampled, Fast Forcing results in the assertion of the nFIQ line to the core.

If the interrupt source is programmed in edge-triggered mode and an active edge is detected, Fast Forcing results in the assertion of the nFIQ line to the core.

The Fast Forcing feature does not affect the Source 0 pending bit in the Interrupt Pending Register (AIC\_IPR).

The FIQ Vector Register (AIC\_FVR) reads the contents of the Source Vector Register 0 (AIC\_SVR0), whatever the source of the fast interrupt may be. The read of the FVR does not clear the Source 0 when the fast forcing feature is used and the interrupt source should be cleared by writing to the Interrupt Clear Command Register (AIC\_ICCR).

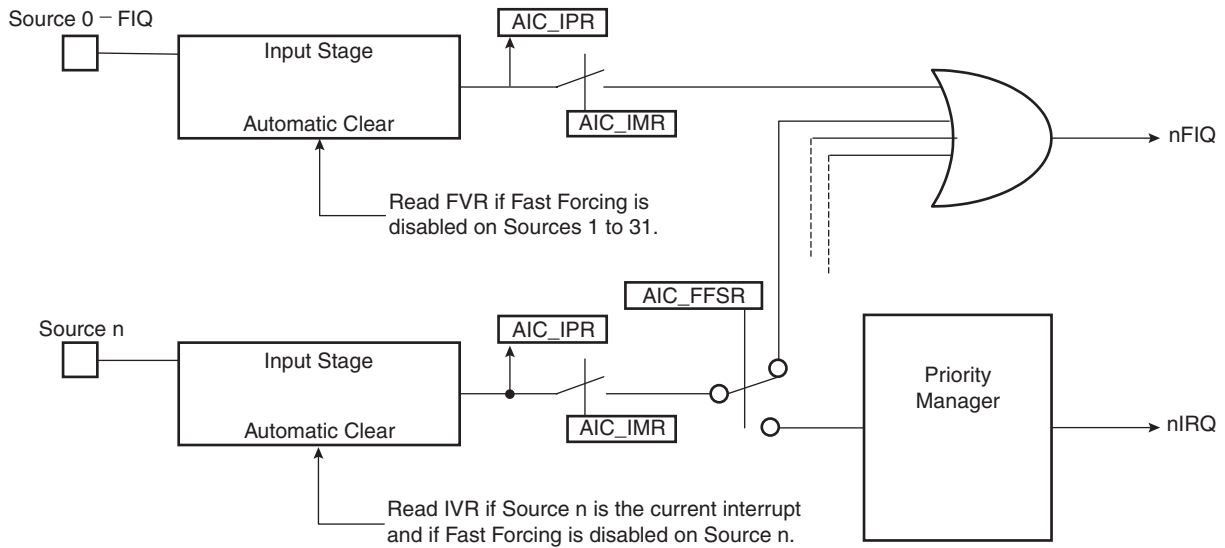


All enabled and pending interrupt sources that have the fast forcing feature enabled and that are programmed in edge-triggered mode must be cleared by writing to the Interrupt Clear Command Register. In doing so, they are cleared independently and thus lost interrupts are prevented.

The read of AIC\_IVR does not clear the source that has the fast forcing feature enabled.

The source 0, reserved to the fast interrupt, continues operating normally and becomes one of the Fast Interrupt sources.

**Figure 22-10. Fast Forcing**



## 22.7.5 Protect Mode

The Protect Mode permits reading the Interrupt Vector Register without performing the associated automatic operations. This is necessary when working with a debug system. When a debugger, working either with a Debug Monitor or the ARM processor's ICE, stops the applications and updates the opened windows, it might read the AIC User Interface and thus the IVR. This has undesirable consequences:

- If an enabled interrupt with a higher priority than the current one is pending, it is stacked.
- If there is no enabled pending interrupt, the spurious vector is returned.

In either case, an End of Interrupt command is necessary to acknowledge and to restore the context of the AIC. This operation is generally not performed by the debug system as the debug system would become strongly intrusive and cause the application to enter an undesired state.

This is avoided by using the Protect Mode. Writing DBGM in AIC\_DCR (Debug Control Register) at 0x1 enables the Protect Mode.

When the Protect Mode is enabled, the AIC performs interrupt stacking only when a write access is performed on the AIC\_IVR. Therefore, the Interrupt Service Routines must write (arbitrary data) to the AIC\_IVR just after reading it. The new context of the AIC, including the value of the Interrupt Status Register (AIC\_ISR), is updated with the current interrupt only when AIC\_IVR is written.

An AIC\_IVR read on its own (e.g., by a debugger), modifies neither the AIC context nor the AIC\_ISR. Extra AIC\_IVR reads perform the same operations. However, it is recommended to not stop the processor between the read and the write of AIC\_IVR of the interrupt service routine to make sure the debugger does not modify the AIC context.

To summarize, in normal operating mode, the read of AIC\_IVR performs the following operations within the AIC:

1. Calculates active interrupt (higher than current or spurious).
2. Determines and returns the vector of the active interrupt.
3. Memorizes the interrupt.
4. Pushes the current priority level onto the internal stack.
5. Acknowledges the interrupt.

However, while the Protect Mode is activated, only operations 1 to 3 are performed when AIC\_IVR is read. Operations 4 and 5 are only performed by the AIC when AIC\_IVR is written.

Software that has been written and debugged using the Protect Mode runs correctly in Normal Mode without modification. However, in Normal Mode the AIC\_IVR write has no effect and can be removed to optimize the code.

## 22.7.6 Spurious Interrupt

The Advanced Interrupt Controller features protection against spurious interrupts. A spurious interrupt is defined as being the assertion of an interrupt source long enough for the AIC to assert the nIRQ, but no longer present when AIC\_IVR is read. This is most prone to occur when:

- An external interrupt source is programmed in level-sensitive mode and an active level occurs for only a short time.

- An internal interrupt source is programmed in level sensitive and the output signal of the corresponding embedded peripheral is activated for a short time. (As in the case for the Watchdog.)
- An interrupt occurs just a few cycles before the software begins to mask it, thus resulting in a pulse on the interrupt source.

The AIC detects a spurious interrupt at the time the AIC\_IVR is read while no enabled interrupt source is pending. When this happens, the AIC returns the value stored by the programmer in AIC\_SPU (Spurious Vector Register). The programmer must store the address of a spurious interrupt handler in AIC\_SPU as part of the application, to enable an as fast as possible return to the normal execution flow. This handler writes in AIC\_EOICR and performs a return from interrupt.

## 22.7.7 General Interrupt Mask

The AIC features a General Interrupt Mask bit to prevent interrupts from reaching the processor. Both the nIRQ and the nFIQ lines are driven to their inactive state if the bit GMSK in AIC\_DCR (Debug Control Register) is set. However, this mask does not prevent waking up the processor if it has entered Idle Mode. This function facilitates synchronizing the processor on a next event and, as soon as the event occurs, performs subsequent operations without having to handle an interrupt. It is strongly recommended to use this mask with caution.

## 22.8 Advanced Interrupt Controller (AIC) User Interface

### 22.8.1 Base Address

The AIC is mapped at the address **0xFFFF F000**. It has a total 4-Kbyte addressing space. This permits the vectoring feature, as the PC-relative load/store instructions of the ARM processor support only a  $\pm$  4-Kbyte offset.

### 22.8.2 Register Mapping

**Table 22-2.** Register Mapping

Offset	Register	Name	Access	Reset Value
0000	Source Mode Register 0	AIC_SMR0	Read/Write	0x0
0x04	Source Mode Register 1	AIC_SMR1	Read/Write	0x0
---	---	---	---	---
0x7C	Source Mode Register 31	AIC_SMR31	Read/Write	0x0
0x80	Source Vector Register 0	AIC_SVR0	Read/Write	0x0
0x84	Source Vector Register 1	AIC_SVR1	Read/Write	0x0
---	---	---	---	---
0xFC	Source Vector Register 31	AIC_SVR31	Read/Write	0x0
0x100	Interrupt Vector Register	AIC_IVR	Read-only	0x0
0x104	FIQ Interrupt Vector Register	AIC_FVR	Read-only	0x0
0x108	Interrupt Status Register	AIC_ISR	Read-only	0x0
0x10C	Interrupt Pending Register <sup>(2)</sup>	AIC_IPR	Read-only	0x0 <sup>(1)</sup>
0x110	Interrupt Mask Register <sup>(2)</sup>	AIC_IMR	Read-only	0x0
0x114	Core Interrupt Status Register	AIC_CISR	Read-only	0x0
0x118	Reserved	---	---	---
0x11C	Reserved	---	---	---
0x120	Interrupt Enable Command Register <sup>(2)</sup>	AIC_IECR	Write-only	---
0x124	Interrupt Disable Command Register <sup>(2)</sup>	AIC_IDCR	Write-only	---
0x128	Interrupt Clear Command Register <sup>(2)</sup>	AIC_ICCR	Write-only	---
0x12C	Interrupt Set Command Register <sup>(2)</sup>	AIC_ISCR	Write-only	---
0x130	End of Interrupt Command Register	AIC_EOICR	Write-only	---
0x134	Spurious Interrupt Vector Register	AIC_SPU	Read/Write	0x0
0x138	Debug Control Register	AIC_DCR	Read/Write	0x0
0x13C	Reserved	---	---	---
0x140	Fast Forcing Enable Register <sup>(2)</sup>	AIC_FFER	Write-only	---
0x144	Fast Forcing Disable Register <sup>(2)</sup>	AIC_FFDR	Write-only	---
0x148	Fast Forcing Status Register <sup>(2)</sup>	AIC_FFSR	Read-only	0x0

- Notes:
1. The reset value of this register depends on the level of the external interrupt source. All other sources are cleared at reset, thus not pending.
  2. PID2...PID31 bit fields refer to the identifiers as defined in the Peripheral Identifiers Section of the product datasheet.

## 22.8.3 AIC Source Mode Register

**Register Name:** AIC\_SMR0..AIC\_SMR31

**Access Type:** Read/Write

**Reset Value:** 0x0

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

- **PRIOR: Priority Level**

Programs the priority level for all sources except FIQ source (source 0).

The priority level can be between 0 (lowest) and 7 (highest).

The priority level is not used for the FIQ in the related SMR register AIC\_SMRx.

- **SRCTYPE: Interrupt Source Type**

The active level or edge is not programmable for the internal interrupt sources.

SRCTYPE		Internal Interrupt Sources	External Interrupt Sources
0	0	High level Sensitive	Low level Sensitive
0	1	Positive edge triggered	Negative edge triggered
1	0	High level Sensitive	High level Sensitive
1	1	Positive edge triggered	Positive edge triggered

### 22.8.4 AIC Source Vector Register

**Register Name:** AIC\_SVR0..AIC\_SVR31

**Access Type:** Read/Write

**Reset Value:** 0x0

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

- **VECTOR: Source Vector**

The user may store in these registers the addresses of the corresponding handler for each interrupt source.

### 22.8.5 AIC Interrupt Vector Register

**Register Name:** AIC\_IVR

**Access Type:** Read-only

**Reset Value:** 0x0

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

- **IRQV: Interrupt Vector Register**

The Interrupt Vector Register contains the vector programmed by the user in the Source Vector Register corresponding to the current interrupt.

The Source Vector Register is indexed using the current interrupt number when the Interrupt Vector Register is read.

When there is no current interrupt, the Interrupt Vector Register reads the value stored in AIC\_SPU.

## 22.8.6 AIC FIQ Vector Register

**Register Name:** AIC\_FVR

**Access Type:** Read-only

**Reset Value:** 0 x0

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

- **FIQV: FIQ Vector Register**

The FIQ Vector Register contains the vector programmed by the user in the Source Vector Register 0. When there is no fast interrupt, the FIQ Vector Register reads the value stored in AIC\_SPU.

## 22.8.7 AIC Interrupt Status Register

**Register Name:** AIC\_ISR

**Access Type:** Read-only

**Reset Value:** 0x0

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

- **IRQID: Current Interrupt Identifier**

The Interrupt Status Register returns the current interrupt source number.

### 22.8.8 AIC Interrupt Pending Register

**Register Name:** AIC\_IPR

**Access Type:** Read-only

**Reset Value:** 0x0

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	SYS	FIQ

• **FIQ, SYS, PID2-PID31: Interrupt Pending**

0 = Corresponding interrupt is not pending.

1 = Corresponding interrupt is pending.

### 22.8.9 AIC Interrupt Mask Register

**Register Name:** AIC\_IMR

**Access Type:** Read-only

**Reset Value:** 0x0

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	SYS	FIQ

• **FIQ, SYS, PID2-PID31: Interrupt Mask**

0 = Corresponding interrupt is disabled.

1 = Corresponding interrupt is enabled.



## 22.8.10 AIC Core Interrupt Status Register

**Register Name:** AIC\_CISR

**Access Type:** Read-only

**Reset Value:** 0x0

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

- **NFIQ: NFIQ Status**

0 = nFIQ line is deactivated.

1 = nFIQ line is active.

- **NIRQ: NIRQ Status**

0 = nIRQ line is deactivated.

1 = nIRQ line is active.

## 22.8.11 AIC Interrupt Enable Command Register

**Register Name:** AIC\_IECR

**Access Type:** 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	SYS	FIQ

- **FIQ, SYS, PID2-PID3: Interrupt Enable**

0 = No effect.

1 = Enables corresponding interrupt.

### 22.8.12 AIC Interrupt Disable Command Register

**Register Name:** AIC\_IDCR

**Access Type:** 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	SYS	FIQ

• **FIQ, SYS, PID2-PID31: Interrupt Disable**

0 = No effect.

1 = Disables corresponding interrupt.

### 22.8.13 AIC Interrupt Clear Command Register

**Register Name:** AIC\_ICCR

**Access Type:** 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	SYS	FIQ

• **FIQ, SYS, PID2-PID31: Interrupt Clear**

0 = No effect.

1 = Clears corresponding interrupt.

## 22.8.14 AIC Interrupt Set Command Register

**Register Name:** AIC\_ISCR

**Access Type:** 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	SYS	FIQ

- **FIQ, SYS, PID2-PID31: Interrupt Set**

0 = No effect.

1 = Sets corresponding interrupt.

## 22.8.15 AIC End of Interrupt Command Register

**Register Name:** AIC\_EOICR

**Access Type:** Write-only

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

The End of Interrupt Command Register is used by the interrupt routine to indicate that the interrupt treatment is complete. Any value can be written because it is only necessary to make a write to this register location to signal the end of interrupt treatment.

## 22.8.16 AIC Spurious Interrupt Vector Register

**Register Name:** AIC\_SPU  
**Access Type:** Read/Write  
**Reset Value:** 0x0

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

- **SIQV: Spurious Interrupt Vector Register**

The user may store the address of a spurious interrupt handler in this register. The written value is returned in AIC\_IVR in case of a spurious interrupt and in AIC\_FVR in case of a spurious fast interrupt.

## 22.8.17 AIC Debug Control Register

**Register Name:** AIC\_DEBUG  
**Access Type:** Read/Write  
**Reset Value:** 0x0

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

- **PROT: Protection Mode**

0 = The Protection Mode is disabled.

1 = The Protection Mode is enabled.

- **GMSK: General Mask**

0 = The nIRQ and nFIQ lines are normally controlled by the AIC.

1 = The nIRQ and nFIQ lines are tied to their inactive state.

## 22.8.18 AIC Fast Forcing Enable Register

Register Name: AIC\_FFER

Access Type: 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	SYS	–

- **SYS, PID2-PID31: Fast Forcing Enable**

0 = No effect.

1 = Enables the fast forcing feature on the corresponding interrupt.

## 22.8.19 AIC Fast Forcing Disable Register

Register Name: AIC\_FFDR

Access Type: 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	SYS	–

- **SYS, PID2-PID31: Fast Forcing Disable**

0 = No effect.

1 = Disables the Fast Forcing feature on the corresponding interrupt.



## 22.8.20 AIC Fast Forcing Status Register

Register Name: AIC\_FFSR

Access Type: 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	SYS	–

- **SYS, PID2-PID31: Fast Forcing Status**

0 = The Fast Forcing feature is disabled on the corresponding interrupt.

1 = The Fast Forcing feature is enabled on the corresponding interrupt.

## 23. Clock Generator

### 23.1 Description

The Clock Generator is made up of 1 PLL, a Main Oscillator, as well as an RC Oscillator .

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 Oscillator
- PLLCK is the output of the Divider and PLL block

The Clock Generator User Interface is embedded within the Power Management Controller one and is described in [Section 24.9](#). However, the Clock Generator registers are named CKGR\_.

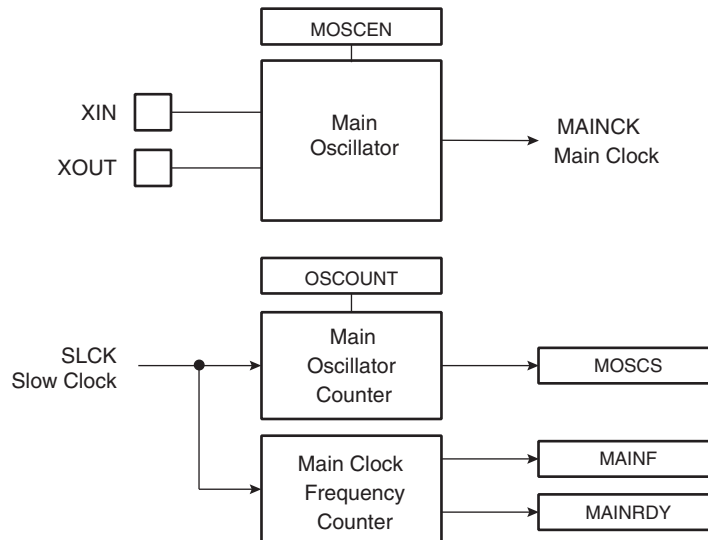
### 23.2 Slow Clock RC Oscillator

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.

### 23.3 Main Oscillator

[Figure 23-1](#) shows the Main Oscillator block diagram.

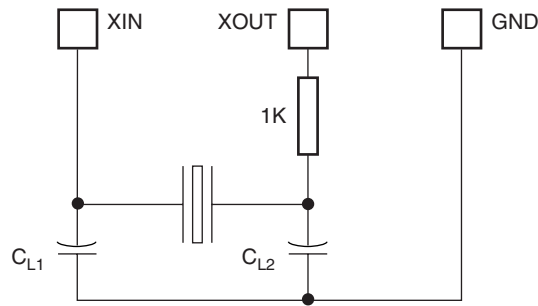
**Figure 23-1.** Main Oscillator Block Diagram



#### 23.3.1 Main Oscillator Connections

The Clock Generator integrates a Main Oscillator that is designed for a 3 to 20 MHz fundamental crystal. The typical crystal connection is illustrated in [Figure 23-2](#). The 1 kΩ resistor is only required for crystals with frequencies lower than 8 MHz. For further details on the electrical characteristics of the Main Oscillator, see the section “DC Characteristics” of the product datasheet.

**Figure 23-2.** Typical Crystal Connection



### 23.3.2 Main Oscillator Startup Time

The startup time of the Main Oscillator is given in the DC Characteristics section of the product datasheet. The startup time depends on the crystal frequency and decreases when the frequency rises.

### 23.3.3 Main Oscillator Control

To minimize the power required to start up the system, the main oscillator is disabled after reset and slow clock is selected.

The software enables or disables the main oscillator so as to reduce power consumption by clearing the MOSCEN bit in the Main Oscillator Register (CKGR\_MOR).

When disabling the main oscillator by clearing the MOSCEN bit in CKGR\_MOR, the MOSCS 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 main oscillator.

When the MOSCEN bit and the OSCOUNT are written in CKGR\_MOR to enable the main oscillator, the MOSCS bit in PMC\_SR (Status Register) is cleared and the counter starts counting down on the slow clock divided by 8 from the OSCOUNT value. Since the OSCOUNT value is coded with 8 bits, the maximum startup time is about 62 ms.

When the counter reaches 0, the MOSCS bit is set, indicating that the main clock is valid. Setting the MOSCS bit in PMC\_IMR can trigger an interrupt to the processor.

### 23.3.4 Main Clock Frequency Counter

The Main Oscillator features a Main Clock frequency counter that provides the quartz frequency connected to the Main Oscillator. Generally, this value is known by the system designer; however, it is useful for the boot program to configure the device with the correct clock speed, independently of the application.

The Main Clock frequency counter starts incrementing at the Main Clock speed after the next rising edge of the Slow Clock as soon as the Main Oscillator is stable, i.e., as soon as the MOSCS bit is set. Then, at the 16th falling edge of Slow Clock, the MAINRDY bit in CKGR\_MCFR (Main Clock Frequency Register) 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 crystal connected on the Main Oscillator can be determined.



## 23.3.5 Main Oscillator Bypass

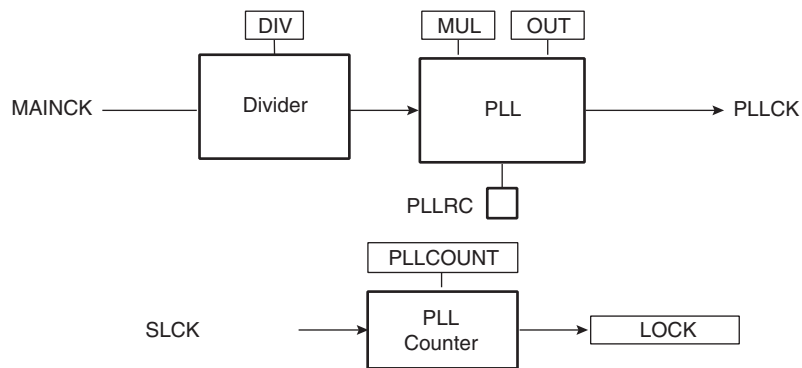
The user can input a clock on the device instead of connecting a crystal. In this case, the user has to provide the external clock signal on the XIN pin. The input characteristics of the XIN pin under these conditions are given in the product electrical characteristics section. The programmer has to be sure to set the OSCBYPASS bit to 1 and the MOSCEN bit to 0 in the Main OSC register (CKGR\_MOR) for the external clock to operate properly.

## 23.4 Divider and PLL Block

The PLL embeds an input divider to increase the accuracy of the resulting clock signals. However, the user must respect the PLL minimum input frequency when programming the divider.

Figure 23-3 shows the block diagram of the divider and PLL block.

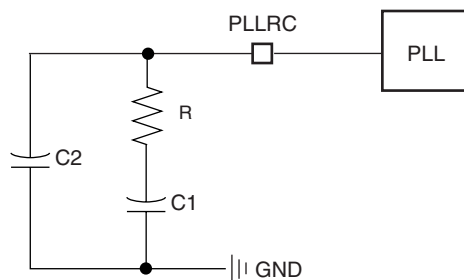
**Figure 23-3.** Divider and PLL Block Diagram



### 23.4.1 PLL Filter

The PLL requires connection to an external second-order filter through the PLLRC pin. Figure 23-4 shows a schematic of these filters.

**Figure 23-4.** PLL Capacitors and Resistors



Values of R, C1 and C2 to be connected to the PLLRC pin must be calculated as a function of the PLL input frequency, the PLL output frequency and the phase margin. A trade-off has to be found between output signal overshoot and startup time.

### 23.4.2 Divider and Phase Lock Loop Programming

The divider can be set between 1 and 255 in steps of 1. When a divider field (DIV) is set to 0, the output of the corresponding divider and the PLL output is a continuous signal at level 0. On reset, each DIV field is set to 0, thus the corresponding PLL input clock is set to 0.

The PLL allows multiplication of the divider's outputs. The PLL clock signal has a frequency that depends on the respective source signal frequency and on the parameters DIV and MUL. The factor applied to the source signal frequency is  $(MUL + 1)/DIV$ . When MUL is written to 0, the corresponding 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 values written in the PLLCOUNT field in CKGR\_PLLR are loaded in the PLL counter. The PLL counter then decrements at the speed of the Slow Clock until it reaches 0. At this time, the LOCK bit is set in PMC\_SR and can trigger an interrupt to the processor. The user has to load the number of Slow Clock cycles required to cover the PLL transient time into the PLLCOUNT field. The transient time depends on the PLL filter. The initial state of the PLL and its target frequency can be calculated using a specific tool provided by Atmel.

## 24. Power Management Controller (PMC)

### 24.1 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 ARM Processor.

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 AIC and the Memory Controller.
- Processor Clock (PCK), switched off when entering processor in idle mode.
- Peripheral Clocks, typically MCK, provided to the embedded peripherals (USART, SSC, SPI, TWI, TC, MCI, etc.) and independently controllable. In order to reduce the number of clock names in a product, the Peripheral Clocks are named MCK in the product datasheet.
- UDP Clock (UDPCK), required by USB Device Port operations.
- Programmable Clock Outputs can be selected from the clocks provided by the clock generator and driven on the PCKx pins.

### 24.2 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 and the memory controller.

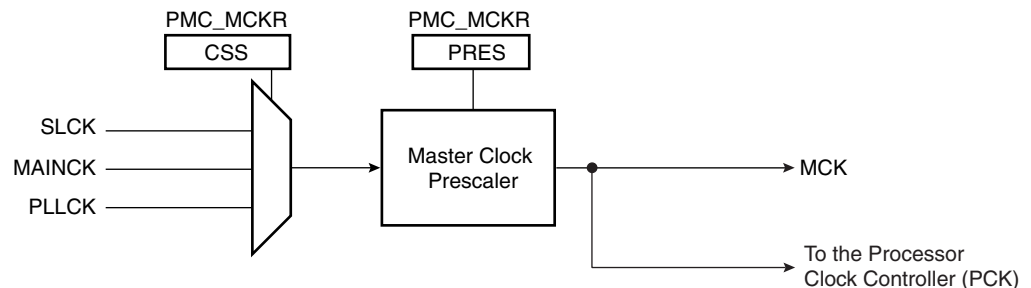
The Master Clock is selected from one of the clocks provided by the Clock Generator. Selecting the Slow Clock provides a Slow Clock signal to the whole device. Selecting the Main Clock saves power consumption of the PLL.

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. 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 24-1.** Master Clock Controller



### 24.3 Processor Clock Controller

The PMC features a Processor Clock Controller (PCK) that implements the Processor Idle Mode. The Processor Clock can be enabled and disabled by writing the System Clock Enable (PMC\_SCER) and System Clock Disable Registers (PMC\_SCDR). The status of this clock (at least for debug purpose) can be read in the System Clock Status Register (PMC\_SCSR).

The Processor Clock PCK is enabled after a reset and is automatically re-enabled by any enabled interrupt. The Processor Idle 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 the Processor Clock is disabled, the current instruction is finished before the clock is stopped, but this does not prevent data transfers from other masters of the system bus.

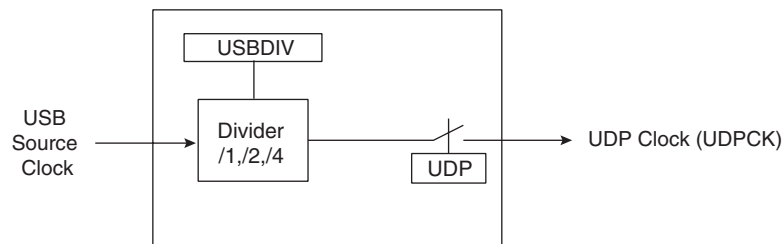
### 24.4 USB Clock Controller

The USB Source Clock is the PLL output. If using the USB, the user must program the PLL to generate a 48 MHz, a 96 MHz or a 192 MHz signal with an accuracy of  $\pm 0.25\%$  depending on the USBDIV bit in CKGR\_PLLR.

When the PLL output is stable, i.e., the LOCK bit is set:

- The USB device clock can be enabled by setting the UDP bit in PMC\_SCER. To save power on this peripheral when it is not used, the user can set the UDP bit in PMC\_SCDR. The UDP bit in PMC\_SCSR gives the activity of this clock. The USB device port require both the 48 MHz signal and the Master Clock. The Master Clock may be controlled via the Peripheral Clock Controller.

**Figure 24-2.** USB Clock Controller



### 24.5 Peripheral Clock Controller

The Power Management Controller controls the clocks of each embedded peripheral by the way of the Peripheral Clock Controller. The user can individually enable and disable the Master Clock on the peripherals by writing into the 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. Generally, the bit number corresponds to the interrupt source number assigned to the peripheral.

## 24.6 Programmable Clock Output Controller

The PMC controls 4 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, the PLL output and the main clock 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.

## 24.7 Programming Sequence

### 1. Enabling the Main Oscillator:

The main oscillator is enabled by setting the MOSCEN field in the CKGR\_MOR register. In some cases it may be advantageous to define a start-up time. This can be achieved by writing a value in the OSCOUNT field in the CKGR\_MOR register.

Once this register has been correctly configured, the user must wait for MOSCS field in the PMC\_SR register 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 MOSCS has been enabled in the PMC\_IER register.

Code Example:

```
write_register(CKGR_MOR, 0x00000701)
```

Start Up Time =  $8 * \text{OSCOUNT} / \text{SLCK} = 56$  Slow Clock Cycles.

So, the main oscillator will be enabled (MOSCS 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 oscillator frequency. This measure can be accomplished via the CKGR\_MCFR register.

Once the MAINRDY field is set in CKGR\_MCFR register, the user may read the MAINF field in CKGR\_MCFR register. 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 CKGR\_PLLR register.

The DIV field is used to control divider itself. A value between 0 and 255 can be programmed. Divider output is divider input divided by DIV parameter. By default DIV parameter is set to 0 which means that divider is turned off.

The OUT field is used to select the PLL B output frequency range.

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 register after CKGR\_PLLR register 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 register. 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 register. 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.

The USBDIV field is used to control the additional divider by 1, 2 or 4, which generates the USB clock(s).

Code Example:

```
write_register(CKGR_PLLR, 0x00040805)
```

If PLL and divider are enabled, the PLL input clock is the main clock. PLL output clock is PLL input clock multiplied by 5. Once CKGR\_PLLR has been written, LOCK bit will be set after eight slow clock cycles.

#### 4. Selection of Master Clock and Processor Clock

The Master Clock and the Processor Clock are configurable via the PMC\_MCKR register.

The CSS field is used to select the Master Clock divider source. By default, the selected clock source is slow clock.

The PRES field is used to control the Master Clock prescaler. The user can choose between different values (1, 2, 4, 8, 16, 32, 64). Master Clock output is prescaler input divided by PRES parameter. By default, PRES parameter is set to 1 which means that master clock is equal to slow clock.

Once the PMC\_MCKR register has been written, the user must wait for the MCKRDY bit to be set in the PMC\_SR register. 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 register.

The PMC\_MCKR register must not be programmed in a single write operation. The preferred programming sequence for the PMC\_MCKR register is as follows:

- If a new value for CSS field corresponds to PLL Clock,
  - Program the PRES field in the PMC\_MCKR register.
  - Wait for the MCKRDY bit to be set in the PMC\_SR register.
  - Program the CSS field in the PMC\_MCKR register.
  - Wait for the MCKRDY bit to be set in the PMC\_SR register.
- If a new value for CSS field corresponds to Main Clock or Slow Clock,
  - Program the CSS field in the PMC\_MCKR register.
  - Wait for the MCKRDY bit to be set in the PMC\_SR register.
  - Program the PRES field in the PMC\_MCKR register.
  - Wait for the MCKRDY bit to be set in the PMC\_SR register.

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 PLLx 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 Main Clock. For further information, see [Section 24.8.2. "Clock Switching Waveforms" on page 161](#).

#### 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 16.

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 registers. Depending on the system used, 4 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. Four clock options are available: main clock, slow clock, PLLCK. 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 1 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 register. 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 register. 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.

#### Code Example:

```
write_register(PMC_PCK0, 0x00000015)
```

Programmable clock 0 is main clock divided by 32.

#### 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\_PCER and PMC\_PCDR.

Depending on the system used, 26 peripheral clocks can be enabled or disabled. The PMC\_PCSR provides a clear view as to which peripheral clock is enabled.

Note: Each enabled peripheral clock corresponds to Master Clock.

#### Code Examples:

```
write_register(PMC_PCER, 0x00000110)
```

Peripheral clocks 4 and 8 are enabled.

```
write_register(PMC_PCDR, 0x00000010)
```

Peripheral clock 4 is disabled.



## 24.8 Clock Switching Details

### 24.8.1 Master Clock Switching Timings

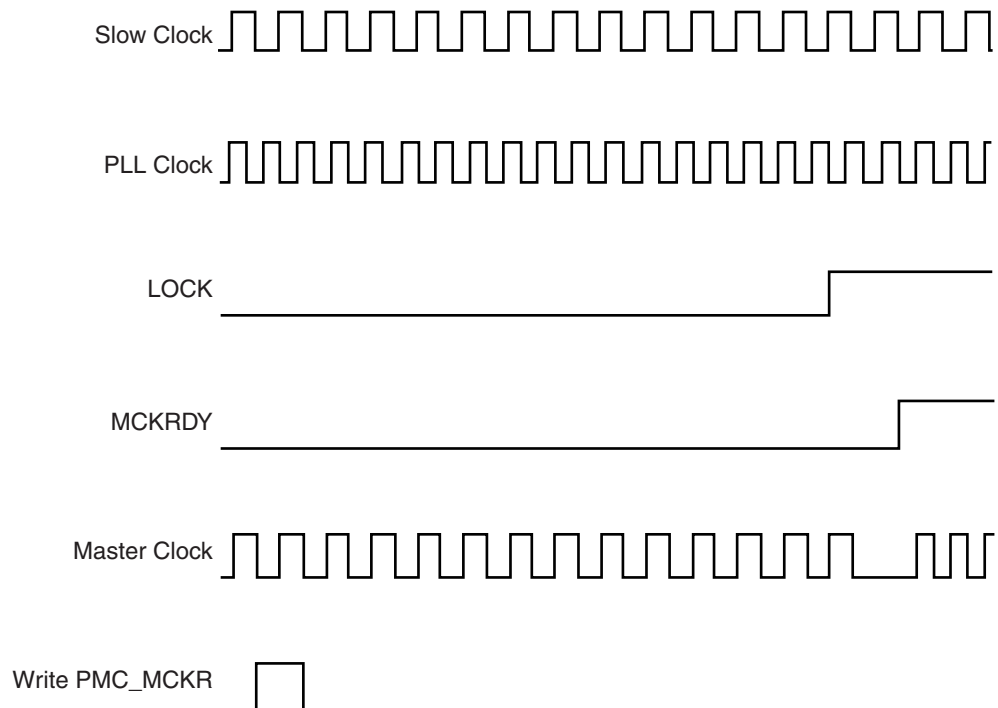
Table 24-1 gives the worst case timings required for the Master Clock to switch from one selected clock to another one. This is in the event that the prescaler is de-activated. When the prescaler is activated, an additional time of 64 clock cycles of the new selected clock has to be added.

**Table 24-1.** Clock Switching Timings (Worst Case)

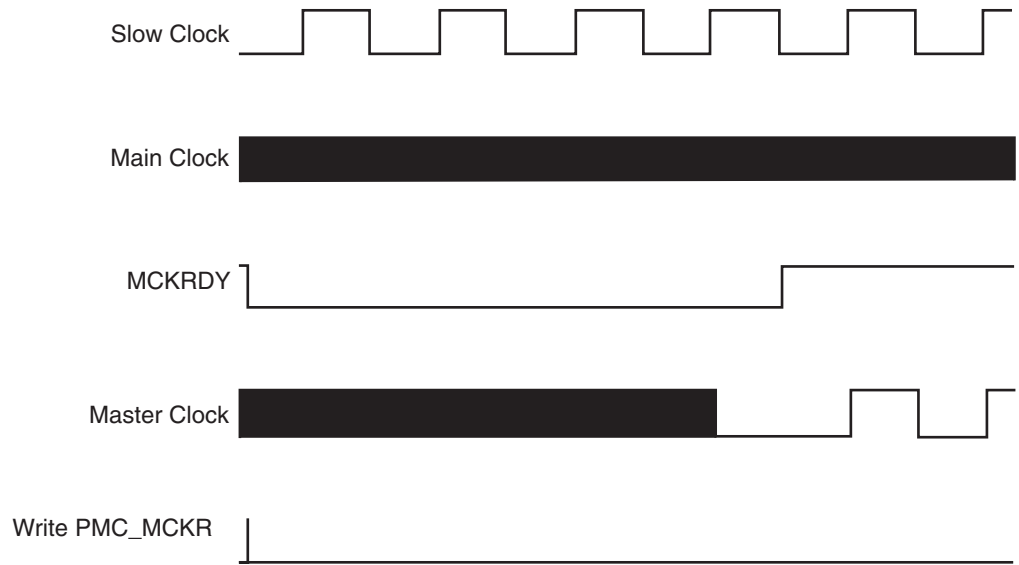
From To	Main Clock	SLCK	PLL Clock
Main Clock	–	4 x SLCK + 2.5 x Main Clock	3 x PLL Clock + 4 x SLCK + 1 x Main Clock
SLCK	0.5 x Main Clock + 4.5 x SLCK	–	3 x PLL Clock + 5 x SLCK
PLL Clock	0.5 x Main Clock + 4 x SLCK + PLLCOUNT x SLCK + 2.5 x PLLx Clock	2.5 x PLL Clock + 5 x SLCK + PLLCOUNT x SLCK	2.5 x PLL Clock + 4 x SLCK + PLLCOUNT x SLCK

### 24.8.2 Clock Switching Waveforms

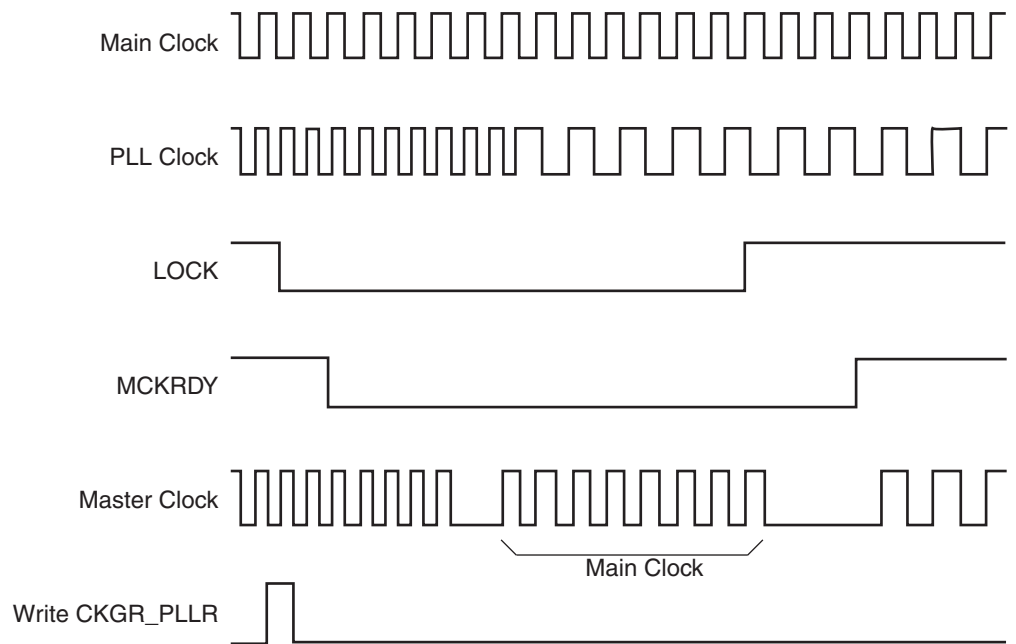
**Figure 24-3.** Switch Master Clock from Slow Clock to PLL Clock



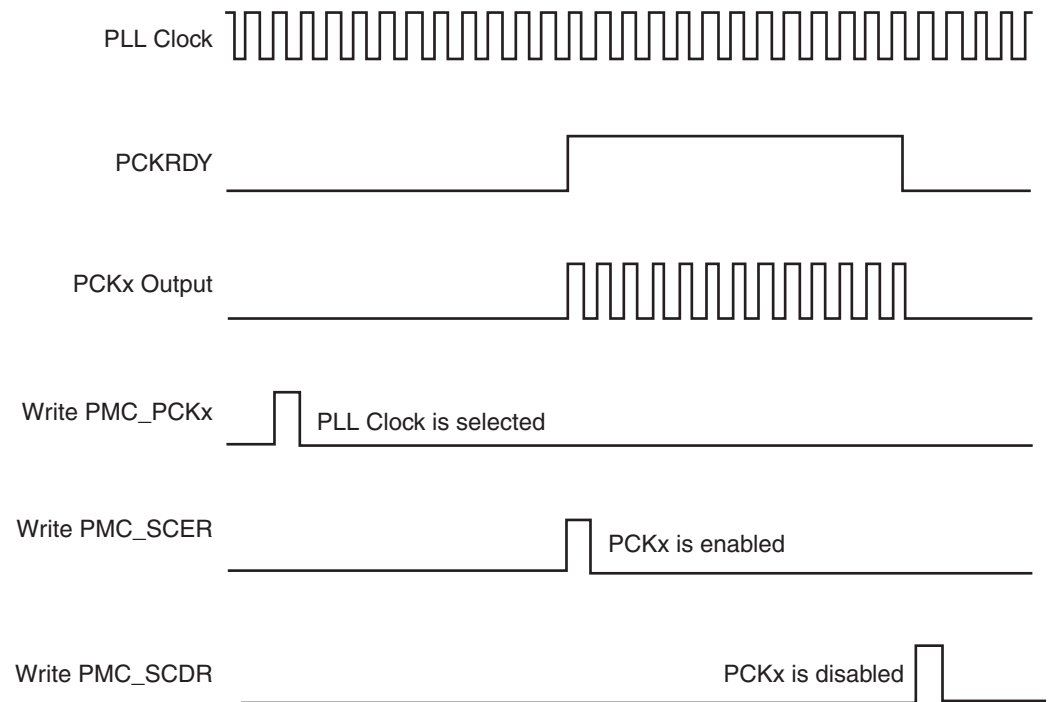
**Figure 24-4.** Switch Master Clock from Main Clock to Slow Clock



**Figure 24-5.** Change PLL Programming



**Figure 24-6.** Programmable Clock Output Programming



## 24.9 Power Management Controller (PMC) User Interface

**Table 24-2.** Register Mapping

Offset	Register	Name	Access	Reset Value
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	0x01
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	0x0
0x0020	Main Oscillator Register	CKGR_MOR	Read/Write	0x0
0x0024	Main Clock Frequency Register	CKGR_MCFR	Read-only	0x0
0x0028	Reserved	–	–	–
0x002C	PLL Register	CKGR_PLLR	Read/Write	0x3F00
0x0030	Master Clock Register	PMC_MCKR	Read/Write	0x0
0x0038	Reserved	–	–	–
0x003C	Reserved	–	–	–
0x0040	Programmable Clock 0 Register	PMC_PCK0	Read/Write	0x0
0x0044	Programmable Clock 1 Register	PMC_PCK1	Read/Write	0x0
...	...	...	...	...
0x0060	Interrupt Enable Register	PMC_IER	Write-only	--
0x0064	Interrupt Disable Register	PMC_IDR	Write-only	--
0x0068	Status Register	PMC_SR	Read-only	0x08
0x006C	Interrupt Mask Register	PMC_IMR	Read-only	0x0
0x0070 - 0x007C	Reserved	–	–	–

## 24.9.1 PMC System Clock Enable Register

**Register Name:** PMC\_SCER

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	PCK3	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
UDP	–	–	–	–	–	–	PCK

- **PCK: Processor Clock Enable**

0 = No effect.

1 = Enables the Processor clock.

- **UDP: USB Device Port Clock Enable**

0 = No effect.

1 = Enables the 48 MHz clock of the USB Device Port.

- **PCKx: Programmable Clock x Output Enable**

0 = No effect.

1 = Enables the corresponding Programmable Clock output.

## 24.9.2 PMC System Clock Disable Register

**Register Name:** PMC\_SCDR

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	PCK3	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
UDP	–	–	–	–	–	–	PCK

- **PCK: Processor Clock Disable**

0 = No effect.

1 = Disables the Processor clock. This is used to enter the processor in Idle Mode.

- **UDP: USB Device Port Clock Disable**

0 = No effect.

1 = Disables the 48 MHz clock of the USB Device Port.

- **PCKx: Programmable Clock x Output Disable**

0 = No effect.

1 = Disables the corresponding Programmable Clock output.

## 24.9.3 PMC System Clock Status Register

**Register Name:** PMC\_SCSR

**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
–	–	–	–	PCK3	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
UDP	–	–	–	–	–	–	PCK

- **PCK: Processor Clock Status**

0 = The Processor clock is disabled.

1 = The Processor clock is enabled.

- **UDP: USB Device Port Clock Status**

0 = The 48 MHz clock (UDPCK) of the USB Device Port is disabled.

1 = The 48 MHz clock (UDPCK) of the USB Device Port is enabled.

- **PCKx: Programmable Clock x Output Status**

0 = The corresponding Programmable Clock output is disabled.

1 = The corresponding Programmable Clock output is enabled.



#### 24.9.4 PMC Peripheral Clock Enable Register

**Register Name:** PMC\_PCER

**Access Type:** 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	-	-

- **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.



## 24.9.5 PMC Peripheral Clock Disable Register

**Register Name:** PMC\_PCDR

**Access Type:** 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	-	-

- 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.

## 24.9.6 PMC Peripheral Clock Status Register

**Register Name:** PMC\_PCSR

**Access Type:** 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.

## 24.9.7 PMC Clock Generator Main Oscillator Register

**Register Name:** CKGR\_MOR

**Access Type:** Read/Write

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

- **MOSCEN: Main Oscillator Enable**

A crystal must be connected between XIN and XOUT.

0 = The Main Oscillator is disabled.

1 = The Main Oscillator is enabled. OSCBYPASS must be set to 0.

When MOSCEN is set, the MOSCS flag is set once the Main Oscillator startup time is achieved.

- **OSCBYPASS: Oscillator Bypass**

0 = No effect.

1 = The Main Oscillator is bypassed. MOSCEN must be set to 0. An external clock must be connected on XIN.

When OSCBYPASS is set, the MOSCS flag in PMC\_SR is automatically set.

Clearing MOSCEN and OSCBYPASS bits allows resetting the MOSCS flag.

- **OSCOUNT: Main Oscillator Start-up Time**

Specifies the number of Slow Clock cycles multiplied by 8 for the Main Oscillator start-up time.



### 24.9.8 PMC Clock Generator Main Clock Frequency Register

Register Name: CKGR\_MCFR

Access Type: Read-only

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

- **MAINF: Main Clock Frequency**

Gives the number of Main Clock cycles within 16 Slow Clock periods.

- **MAINRDY: 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.

## 24.9.9 PMC Clock Generator PLL Register

**Register Name:** CKGR\_PLLR

**Access Type:** Read/Write

31	30	29	28	27	26	25	24
-		USBDIV		-		MUL	
23	22	21	20	19	18	17	16
MUL							
15	14	13	12	11	10	9	8
OUT		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.

- **DIV: Divider**

DIV	Divider Selected
0	Divider output is 0
1	Divider is bypassed
2 - 255	Divider output is the selected clock divided by DIV.

- **PLLCOUNT: PLL Counter**

Specifies the number of slow clock cycles before the LOCK bit is set in PMC\_SR after CKGR\_PLLR is written.

- **OUT: PLL Clock Frequency Range**

To optimize clock performance, this field must be programmed as specified in “PLL Characteristics” in the Electrical Characteristics section of the product datasheet.

- **MUL: PLL Multiplier**

0 = The PLL is deactivated.

1 up to 2047 = The PLL Clock frequency is the PLL input frequency multiplied by MUL+ 1.

- **USBDIV: Divider for USB Clock**

USBDIV		Divider for USB Clock(s)
0	0	Divider output is PLL clock output.
0	1	Divider output is PLL clock output divided by 2.
1	0	Divider output is PLL clock output divided by 4.
1	1	Reserved.



### 24.9.10 PMC Master Clock Register

Register Name: PMC\_MCKR

Access Type: Read/Write

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

• **CSS: Master Clock Selection**

CSS		Clock Source Selection
0	0	Slow Clock is selected
0	1	Main Clock is selected
1	0	Reserved
1	1	PLL Clock is selected.

• **PRES: Processor Clock Prescaler**

PRES			Processor Clock
0	0	0	Selected clock
0	0	1	Selected clock divided by 2
0	1	0	Selected clock divided by 4
0	1	1	Selected clock divided by 8
1	0	0	Selected clock divided by 16
1	0	1	Selected clock divided by 32
1	1	0	Selected clock divided by 64
1	1	1	Reserved

## 24.9.11 PMC Programmable Clock Register

**Register Name:** PMC\_PCKx

**Access Type:** Read/Write

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

- **CSS: Master Clock Selection**

CSS		Clock Source Selection
0	0	Slow Clock is selected
0	1	Main Clock is selected
1	0	Reserved
1	1	PLL Clock is selected

- **PRES: Programmable Clock Prescaler**

PRES			Programmable Clock
0	0	0	Selected clock
0	0	1	Selected clock divided by 2
0	1	0	Selected clock divided by 4
0	1	1	Selected clock divided by 8
1	0	0	Selected clock divided by 16
1	0	1	Selected clock divided by 32
1	1	0	Selected clock divided by 64
1	1	1	Reserved

### 24.9.12 PMC Interrupt Enable Register

**Register Name:** PMC\_IER

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	PCKRDY2	PCKRDY1	PCKRDY0
7	6	5	4	3	2	1	0
–	–	–	–	MCKRDY	LOCK	–	MOSCS

- **MOSCS: Main Oscillator Status Interrupt Enable**
- **LOCK: PLL Lock Interrupt Enable**
- **MCKRDY: Master Clock Ready Interrupt Enable**
- **PCKRDYx: Programmable Clock Ready x Interrupt Enable**

0 = No effect.

1 = Enables the corresponding interrupt.



## 24.9.13 PMC Interrupt Disable Register

**Register Name:** PMC\_IDR

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	PCKRDY2	PCKRDY1	PCKRDY0
7	6	5	4	3	2	1	0
–	–	–	–	MCKRDY	LOCK	–	MOSCS

- **MOSCS: Main Oscillator Status Interrupt Disable**
- **LOCK: PLL Lock Interrupt Disable**
- **MCKRDY: Master Clock Ready Interrupt Disable**
- **PCKRDYx: Programmable Clock Ready x Interrupt Disable**

0 = No effect.

1 = Disables the corresponding interrupt.

### 24.9.14 PMC Status Register

**Register Name:** PMC\_SR

**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
–	–	–	–	–	PCKRDY2	PCKRDY1	PCKRDY0
7	6	5	4	3	2	1	0
–	–	–	–	MCKRDY	LOCK	–	MOSCS

- **MOSCS: MOSCS Flag Status**

0 = Main oscillator is not stabilized.

1 = Main 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.

- **PCKRDYx: Programmable Clock Ready Status**

0 = Programmable Clock x is not ready.

1 = Programmable Clock x is ready.

## 24.9.15 PMC Interrupt Mask Register

**Register Name:** PMC\_IMR

**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
–	–	–	–	–	PCKRDY2	PCKRDY1	PCKRDY0
7	6	5	4	3	2	1	0
–	–	–	–	MCKRDY	LOCK	–	MOSCS

- **MOSCS: Main Oscillator Status Interrupt Mask**
- **LOCK: PLL Lock Interrupt Mask**
- **MCKRDY: Master Clock Ready Interrupt Mask**
- **PCKRDYx: Programmable Clock Ready x Interrupt Mask**

0 = The corresponding interrupt is enabled.

1 = The corresponding interrupt is disabled.



## 25. Debug Unit (DBGU)

### 25.1 Overview

The Debug Unit provides a single entry point from the processor for access to all the debug capabilities of Atmel's ARM-based systems.

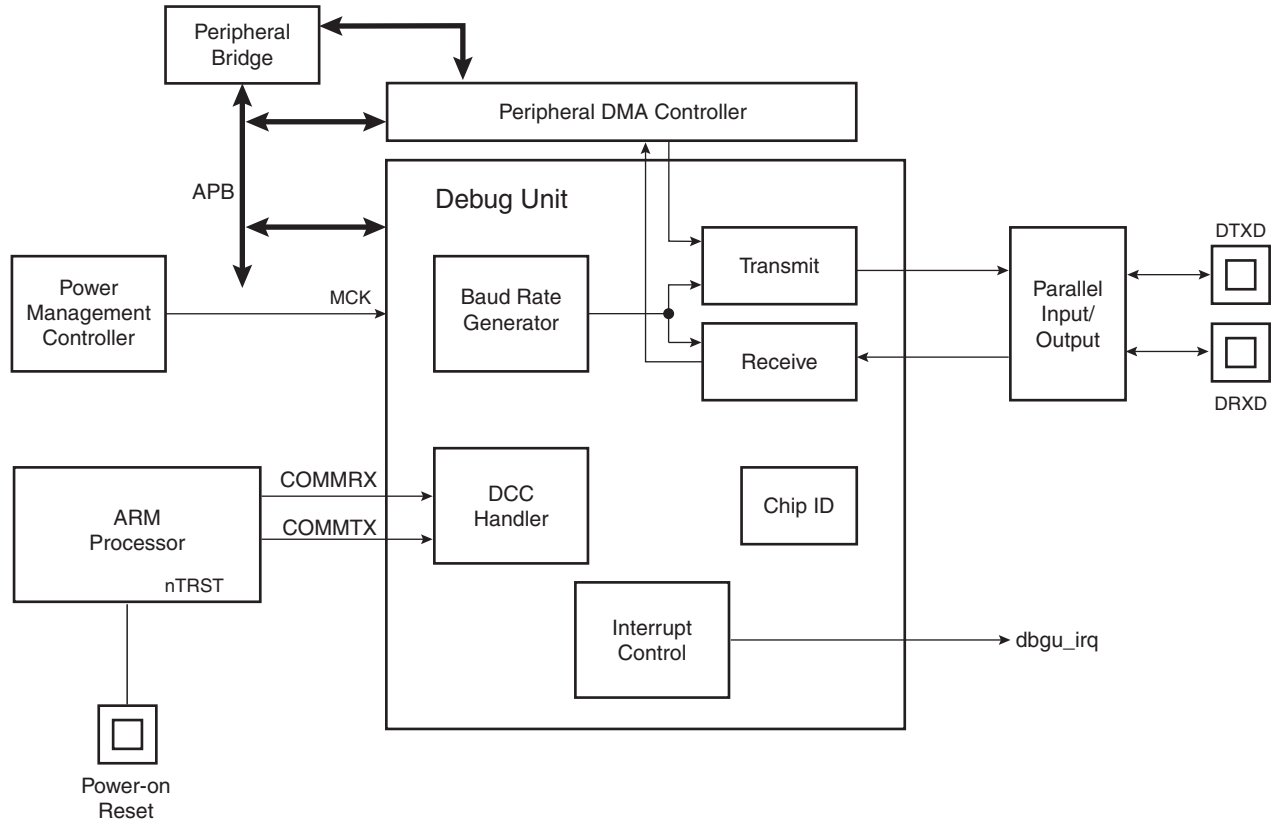
The Debug Unit features a two-pin UART that can be used for several debug and trace purposes and offers an ideal medium for in-situ programming solutions and debug monitor communications. Moreover, the association with two peripheral data controller channels permits packet handling for these tasks with processor time reduced to a minimum.

The Debug Unit also makes the Debug Communication Channel (DCC) signals provided by the In-circuit Emulator of the ARM processor visible to the software. These signals indicate the status of the DCC read and write registers and generate an interrupt to the ARM processor, making possible the handling of the DCC under interrupt control.

Chip Identifier registers permit recognition of the device and its revision. These registers inform as to the sizes and types of the on-chip memories, as well as the set of embedded peripherals.

## 25.2 Block Diagram

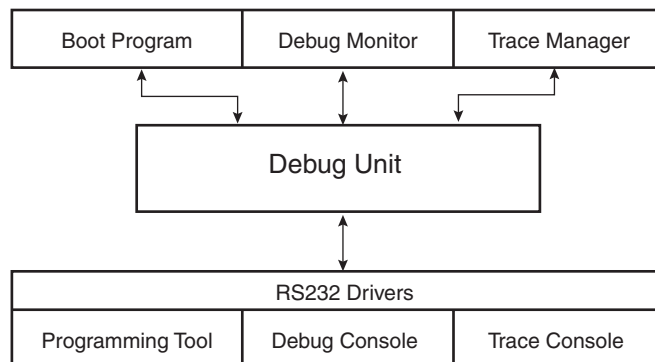
**Figure 25-1.** Debug Unit Functional Block Diagram



**Table 25-1.** Debug Unit Pin Description

Pin Name	Description	Type
DRXD	Debug Receive Data	Input
DTXD	Debug Transmit Data	Output

**Figure 25-2.** Debug Unit Application Example



## 25.3 Product Dependencies

### 25.3.1 I/O Lines

Depending on product integration, the Debug Unit pins may be multiplexed with PIO lines. In this case, the programmer must first configure the corresponding PIO Controller to enable I/O lines operations of the Debug Unit.

### 25.3.2 Power Management

Depending on product integration, the Debug Unit clock may be controllable through the Power Management Controller. In this case, the programmer must first configure the PMC to enable the Debug Unit clock. Usually, the peripheral identifier used for this purpose is 1.

### 25.3.3 Interrupt Source

Depending on product integration, the Debug Unit interrupt line is connected to one of the interrupt sources of the Advanced Interrupt Controller. Interrupt handling requires programming of the AIC before configuring the Debug Unit. Usually, the Debug Unit interrupt line connects to the interrupt source 1 of the AIC, which may be shared with the real-time clock, the system timer interrupt lines and other system peripheral interrupts, as shown in [Figure 25-1](#). This sharing requires the programmer to determine the source of the interrupt when the source 1 is triggered.

## 25.4 UART Operations

The Debug Unit operates as a UART, (asynchronous mode only) and supports only 8-bit character handling (with parity). It has no clock pin.

The Debug Unit's 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.

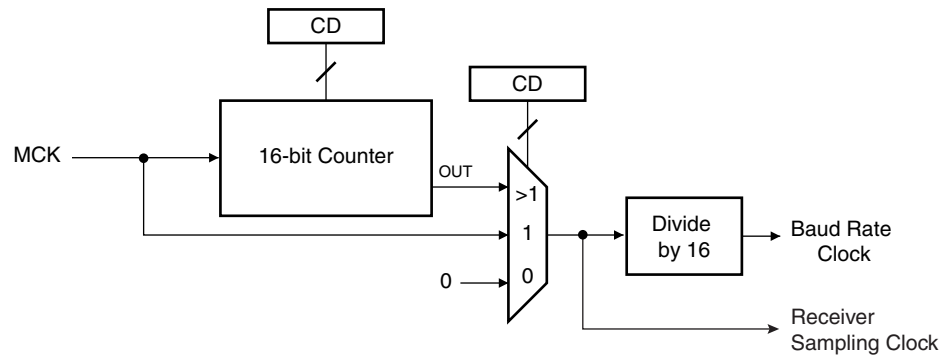
### 25.4.1 Baud Rate Generator

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 DBGU\_BRGR (Baud Rate Generator Register). If DBGU\_BRGR is set to 0, the baud rate clock is disabled and the Debug Unit's 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 25-3.** Baud Rate Generator



## 25.4.2 Receiver

### 25.4.2.1 Receiver Reset, Enable and Disable

After device reset, the Debug Unit receiver is disabled and must be enabled before being used. The receiver can be enabled by writing the control register `DBGU_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 `DBGU_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 `DBGU_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.

### 25.4.2.2 Start Detection and Data Sampling

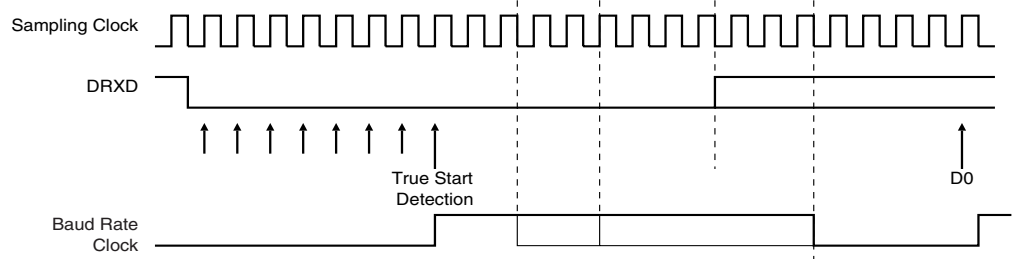
The Debug Unit only supports asynchronous operations, and this affects only its receiver. The Debug Unit receiver detects the start of a received character by sampling the `DRXD` signal until it detects a valid start bit. A low level (space) on `DRXD` is interpreted as a valid start bit if it is detected 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 `DRXD` 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.

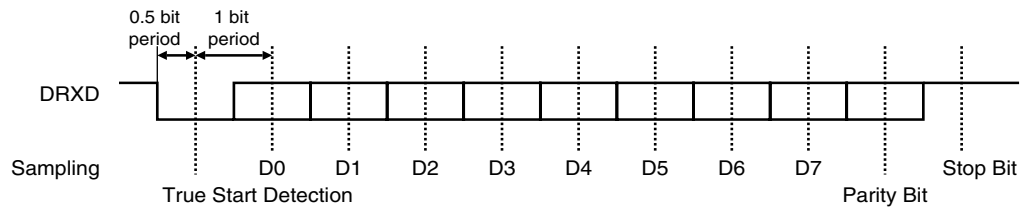


**Figure 25-4. Start Bit Detection**



**Figure 25-5. Character Reception**

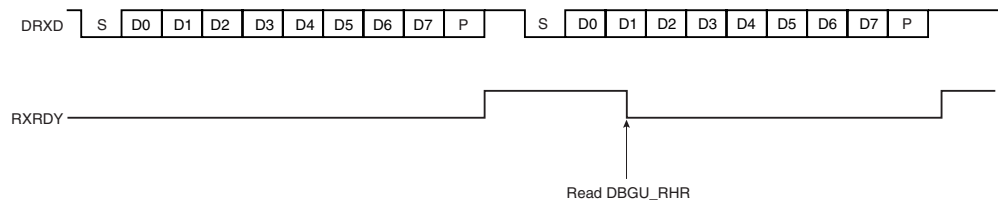
Example: 8-bit, parity enabled 1 stop



### 25.4.2.3 Receiver Ready

When a complete character is received, it is transferred to the DBGU\_RHR and the RXRDY status bit in DBGU\_SR (Status Register) is set. The bit RXRDY is automatically cleared when the receive holding register DBGU\_RHR is read.

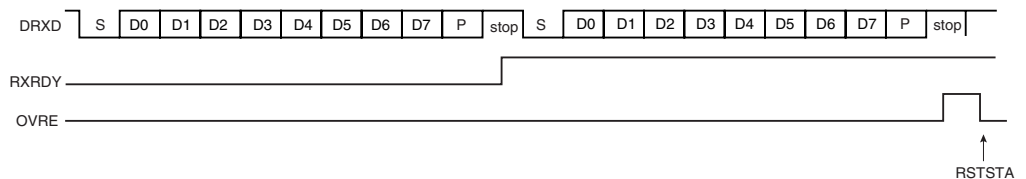
**Figure 25-6. Receiver Ready**



### 25.4.2.4 Receiver Overrun

If DBGU\_RHR has not been read by the software (or the Peripheral Data Controller) since the last transfer, the RXRDY bit is still set and a new character is received, the OVRE status bit in DBGU\_SR is set. OVRE is cleared when the software writes the control register DBGU\_CR with the bit RSTSTA (Reset Status) at 1.

**Figure 25-7. Receiver Overrun**

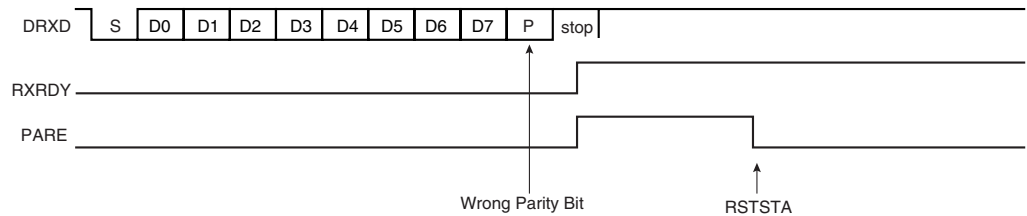


### 25.4.2.5 Parity Error

Each time a character is received, the receiver calculates the parity of the received data bits, in accordance with the field PAR in DBGU\_MR. It then compares the result with the received

parity bit. If different, the parity error bit PARE in DBGU\_SR is set at the same time the RXRDY is set. The parity bit is cleared when the control register DBGU\_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.

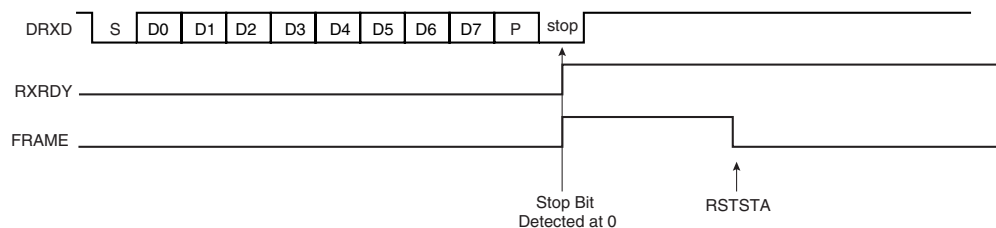
**Figure 25-8.** Parity Error



### 25.4.2.6 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 DBGU\_SR is set at the same time the RXRDY bit is set. The bit FRAME remains high until the control register DBGU\_CR is written with the bit RSTSTA at 1.

**Figure 25-9.** Receiver Framing Error



## 25.4.3 Transmitter

### 25.4.3.1 Transmitter Reset, Enable and Disable

After device reset, the Debug Unit transmitter is disabled and it must be enabled before being used. The transmitter is enabled by writing the control register DBGU\_CR with the bit TXEN at 1. From this command, the transmitter waits for a character to be written in the Transmit Holding Register DBGU\_THR before actually starting the transmission.

The programmer can disable the transmitter by writing DBGU\_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 DBGU\_CR with the bit RSTTX at 1. This immediately stops the transmitter, whether or not it is processing characters.

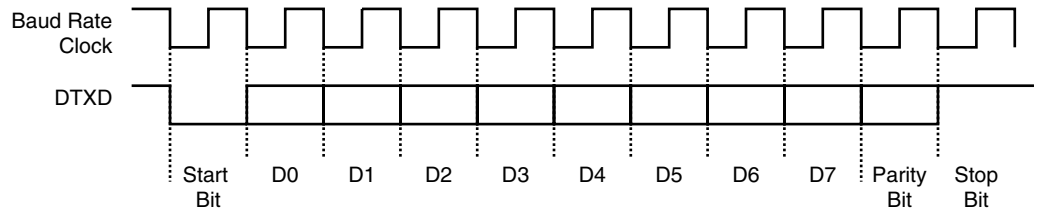
### 25.4.3.2 Transmit Format

The Debug Unit transmitter drives the pin DTXD 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 on the following

figure. The field PARE in the mode register DBGU\_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 25-10.** Character Transmission

Example: Parity enabled

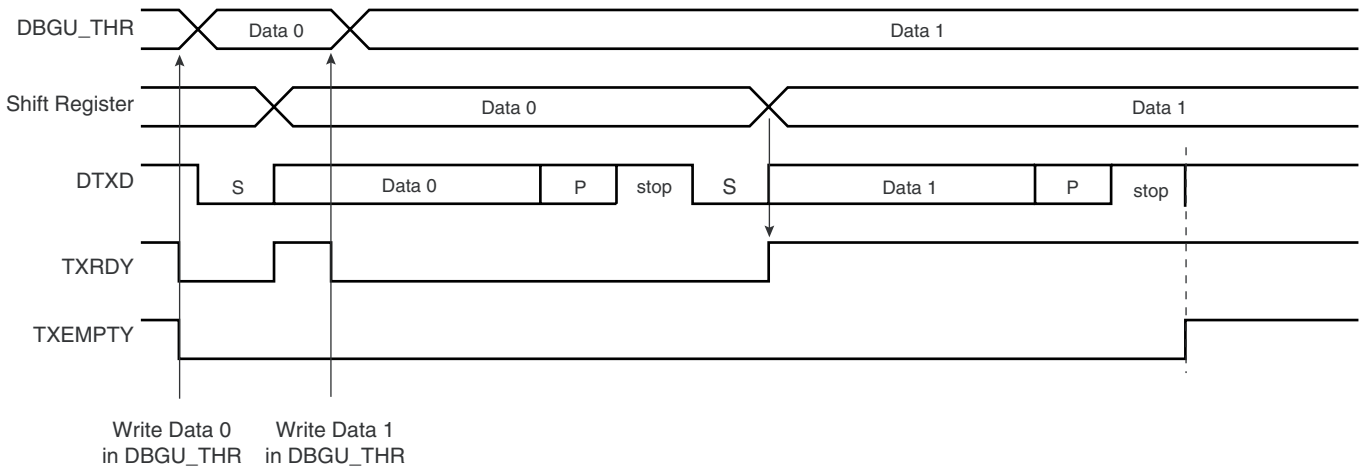


### 25.4.3.3 Transmitter Control

When the transmitter is enabled, the bit TXRDY (Transmitter Ready) is set in the status register DBGU\_SR. The transmission starts when the programmer writes in the Transmit Holding Register DBGU\_THR, and after the written character is transferred from DBGU\_THR to the Shift Register. The bit TXRDY remains high until a second character is written in DBGU\_THR. As soon as the first character is completed, the last character written in DBGU\_THR is transferred into the shift register and TXRDY rises again, showing that the holding register is empty.

When both the Shift Register and the DBGU\_THR are empty, i.e., all the characters written in DBGU\_THR have been processed, the bit TXEMPTY rises after the last stop bit has been completed.

**Figure 25-11.** Transmitter Control



### 25.4.4 Peripheral Data Controller

Both the receiver and the transmitter of the Debug Unit's UART are generally connected to a Peripheral Data Controller (PDC) channel.

The peripheral data controller channels are programmed via registers that are mapped within the Debug Unit user interface from the offset 0x100. The status bits are reported in the Debug Unit status register DBGU\_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 DBGU\_RHR. The TXRDY bit triggers the PDC channel data transfer of the transmitter. This results in a write of a data in DBGU\_THR.

## 25.4.5 Test Modes

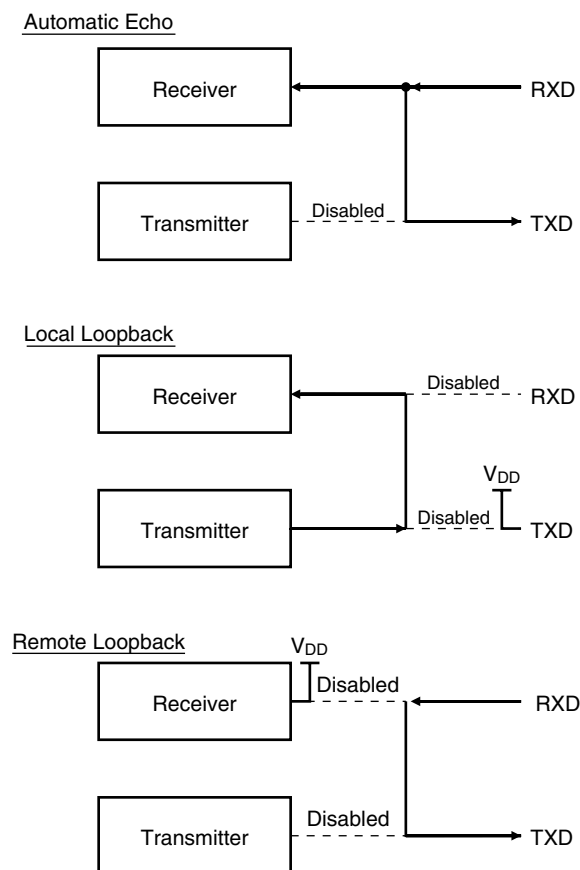
The Debug Unit supports three tests modes. These modes of operation are programmed by using the field CHMODE (Channel Mode) in the mode register DBGU\_MR.

The Automatic Echo mode allows bit-by-bit retransmission. When a bit is received on the DRXD line, it is sent to the DTXD line. The transmitter operates normally, but has no effect on the DTXD line.

The Local Loopback mode allows the transmitted characters to be received. DTXD and DRXD pins are not used and the output of the transmitter is internally connected to the input of the receiver. The DRXD pin level has no effect and the DTXD line is held high, as in idle state.

The Remote Loopback mode directly connects the DRXD pin to the DTXD line. The transmitter and the receiver are disabled and have no effect. This mode allows a bit-by-bit retransmission.

**Figure 25-12.** Test Modes



## 25.4.6 Debug Communication Channel Support

The Debug Unit handles the signals COMMRX and COMMTX that come from the Debug Communication Channel of the ARM Processor and are driven by the In-circuit Emulator.

The Debug Communication Channel contains two registers that are accessible through the ICE Breaker on the JTAG side and through the coprocessor 0 on the ARM Processor side.

As a reminder, the following instructions are used to read and write the Debug Communication Channel:

```
MRC    p14, 0, Rd, c1, c0, 0
```

Returns the debug communication data read register into Rd

```
MCR    p14, 0, Rd, c1, c0, 0
```

Writes the value in Rd to the debug communication data write register.

The bits COMMRX and COMMTX, which indicate, respectively, that the read register has been written by the debugger but not yet read by the processor, and that the write register has been written by the processor and not yet read by the debugger, are wired on the two highest bits of the status register DBGU\_SR. These bits can generate an interrupt. This feature permits handling under interrupt a debug link between a debug monitor running on the target system and a debugger.

## 25.4.7 Chip Identifier

The Debug Unit features two chip identifier registers, DBGU\_CIDR (Chip ID Register) and DBGU\_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 peripheral
- 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.

## 25.5 Debug Unit User Interface

**Table 25-2.** Debug Unit Memory Map

Offset	Register	Name	Access	Reset Value
0x0000	Control Register	DBGU_CR	Write-only	–
0x0004	Mode Register	DBGU_MR	Read/Write	0x0
0x0008	Interrupt Enable Register	DBGU_IER	Write-only	–
0x000C	Interrupt Disable Register	DBGU_IDR	Write-only	–
0x0010	Interrupt Mask Register	DBGU_IMR	Read-only	0x0
0x0014	Status Register	DBGU_SR	Read-only	–
0x0018	Receive Holding Register	DBGU_RHR	Read-only	0x0
0x001C	Transmit Holding Register	DBGU_THR	Write-only	–
0x0020	Baud Rate Generator Register	DBGU_BRGR	Read/Write	0x0
0x0024 - 0x003C	Reserved	–	–	–
0x0040	Chip ID Register	DBGU_CIDR	Read-only	–
0x0044	Chip ID Extension Register	DBGU_EXID	Read-only	–
0x0048	Reserved	–	–	–
0x004C - 0x00FC	Reserved	–	–	–
0x0100 - 0x0124	PDC Area	–	–	–

## 25.5.1 Debug Unit Control Register

**Name:** DBGU\_CR

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	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 the DBGU\_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 DBGU\_SR.

## 25.5.2 Debug Unit Mode Register

Name: DBGU\_MR

Access Type: Read/Write

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

- **PAR: Parity Type**

PAR			Parity Type
0	0	0	Even parity
0	0	1	Odd parity
0	1	0	Space: parity forced to 0
0	1	1	Mark: parity forced to 1
1	x	x	No parity

- **CHMODE: Channel Mode**

CHMODE		Mode Description
0	0	Normal Mode
0	1	Automatic Echo
1	0	Local Loopback
1	1	Remote Loopback



## 25.5.3 Debug Unit Interrupt Enable Register

**Name:** DBGU\_IER

**Access Type:** Write-only

31	30	29	28	27	26	25	24
COMMRX	COMMTX	–	–	–	–	–	–
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
- **COMMTX:** Enable COMMTX (from ARM) Interrupt
- **COMMRX:** Enable COMMRX (from ARM) Interrupt

0 = No effect.

1 = Enables the corresponding interrupt.

## 25.5.4 Debug Unit Interrupt Disable Register

**Name:** DBGU\_IDR

**Access Type:** Write-only

31	30	29	28	27	26	25	24
COMMRX	COMMTX	–	–	–	–	–	–
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
- **COMMTX:** Disable COMMTX (from ARM) Interrupt
- **COMMRX:** Disable COMMRX (from ARM) Interrupt

0 = No effect.

1 = Disables the corresponding interrupt.

## 25.5.5 Debug Unit Interrupt Mask Register

**Name:** DBGU\_IMR

**Access Type:** Read-only

31	30	29	28	27	26	25	24
COMMRX	COMMTX	–	–	–	–	–	–
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**
- **COMMTX: Mask COMMTX Interrupt**
- **COMMRX: Mask COMMRX Interrupt**

0 = The corresponding interrupt is disabled.

1 = The corresponding interrupt is enabled.

## 25.5.6 Debug Unit Status Register

**Name:** DBGU\_SR

**Access Type:** Read-only

31	30	29	28	27	26	25	24
COMMRX	COMMTX	–	–	–	–	–	–
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 DBGU\_RHR or the receiver is disabled.

1 = At least one complete character has been received, transferred to DBGU\_RHR and not yet read.

- **TXRDY: Transmitter Ready**

0 = A character has been written to DBGU\_THR and not yet transferred to the Shift Register, or the transmitter is disabled.

1 = There is no character written to DBGU\_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 DBGU\_THR, or characters being processed by the transmitter, or the transmitter is disabled.

1 = There are no characters in DBGU\_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.

- **COMMTX: Debug Communication Channel Write Status**

0 = COMMTX from the ARM processor is inactive.

1 = COMMTX from the ARM processor is active.

- **COMMRX: Debug Communication Channel Read Status**

0 = COMMRX from the ARM processor is inactive.

1 = COMMRX from the ARM processor is active.

## 25.5.7 Debug Unit Receiver Holding Register

**Name:** DBGU\_RHR

**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
RXCHR							

- **RXCHR: Received Character**

Last received character if RXRDY is set.

## 25.5.8 Debug Unit Transmit Holding Register

**Name:** DBGU\_THR

**Access Type:** Write-only

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

- **TXCHR: Character to be Transmitted**

Next character to be transmitted after the current character if TXRDY is not set.

## 25.5.9 Debug Unit Baud Rate Generator Register

**Name:** DBGU\_BRGR

**Access Type:** Read/Write

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

- **CD: Clock Divisor**

CD	Baud Rate Clock
0	Disabled
1	MCK
2 to 65535	$MCK / (CD \times 16)$



## 25.5.10 Debug Unit Chip ID Register

Name: DBGU\_CIDR

Access Type: 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
- **EPROC:** Embedded Processor

EPROC			Processor
0	0	1	ARM946ES™
0	1	0	ARM7TDMI®
1	0	0	ARM920T™
1	0	1	ARM926EJS™

- **NVPSIZ:** Nonvolatile Program Memory Size

NVPSIZ				Size
0	0	0	0	None
0	0	0	1	8K bytes
0	0	1	0	16K bytes
0	0	1	1	32K bytes
0	1	0	0	Reserved
0	1	0	1	64K bytes
0	1	1	0	Reserved
0	1	1	1	128K bytes
1	0	0	0	Reserved
1	0	0	1	256K bytes
1	0	1	0	512K bytes
1	0	1	1	Reserved
1	1	0	0	1024K bytes
1	1	0	1	Reserved
1	1	1	0	2048K bytes
1	1	1	1	Reserved

- **NVPSIZ2 Second Nonvolatile Program Memory Size**

NVPSIZ2				Size
0	0	0	0	None
0	0	0	1	8K bytes
0	0	1	0	16K bytes
0	0	1	1	32K bytes
0	1	0	0	Reserved
0	1	0	1	64K bytes
0	1	1	0	Reserved
0	1	1	1	128K bytes
1	0	0	0	Reserved
1	0	0	1	256K bytes
1	0	1	0	512K bytes
1	0	1	1	Reserved
1	1	0	0	1024K bytes
1	1	0	1	Reserved
1	1	1	0	2048K bytes
1	1	1	1	Reserved

- **SRAMSIZ: Internal SRAM Size**

SRAMSIZ				Size
0	0	0	0	Reserved
0	0	0	1	1K bytes
0	0	1	0	2K bytes
0	0	1	1	Reserved
0	1	0	0	112K bytes
0	1	0	1	4K bytes
0	1	1	0	80K bytes
0	1	1	1	160K bytes
1	0	0	0	8K bytes
1	0	0	1	16K bytes
1	0	1	0	32K bytes
1	0	1	1	64K bytes
1	1	0	0	128K bytes
1	1	0	1	256K bytes
1	1	1	0	96K bytes
1	1	1	1	512K bytes

- **ARCH: Architecture Identifier**

ARCH		Architecture
Hex	Bin	
0x19	0001 1001	AT91SAM9xx Series
0x29	0010 1001	AT91SAM9XExx Series
0x34	0011 0100	AT91x34 Series
0x39	0011 1001	CAP9 Series
0x40	0100 0000	AT91x40 Series
0x42	0100 0010	AT91x42 Series
0x55	0101 0101	AT91x55 Series
0x60	0101 0000	AT91SAM7Axx Series
0x63	0110 0011	AT91x63 Series
0x70	0111 0000	AT91SAM7Sxx Series
0x71	0111 0001	AT91SAM7XCxx Series
0x72	0111 0010	AT91SAM7SExx Series
0x73	0111 0011	AT91SAM7Lxx Series
0x75	0111 0101	AT91SAM7Xxx Series
0x92	1001 0010	AT91x92 Series
0xF0	1111 0001	AT75Cxx Series

- **NVPTYP: Nonvolatile Program Memory Type**

NVPTYP			Memory
0	0	0	ROM
0	0	1	ROMless or on-chip Flash
1	0	0	SRAM emulating ROM
0	1	0	Embedded Flash Memory
0	1	1	ROM and Embedded Flash Memory NVPSIZ is ROM size NVPSIZ2 is Flash size

- **EXT: Extension Flag**

0 = Chip ID has a single register definition without extension

1 = An extended Chip ID exists.



**25.5.11 Debug Unit Chip ID Extension Register**

**Name:** DBGU\_EXID

**Access Type:** 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 DBGU\_CIDR is 0.

## 26. Parallel Input Output Controller (PIO)

### 26.1 Overview

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.
- A glitch filter providing rejection of pulses lower than one-half of clock cycle.
- Multi-drive capability similar to an open drain I/O line.
- Control of the pull-up 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.

## 26.2 Block Diagram

Figure 26-1. Block Diagram

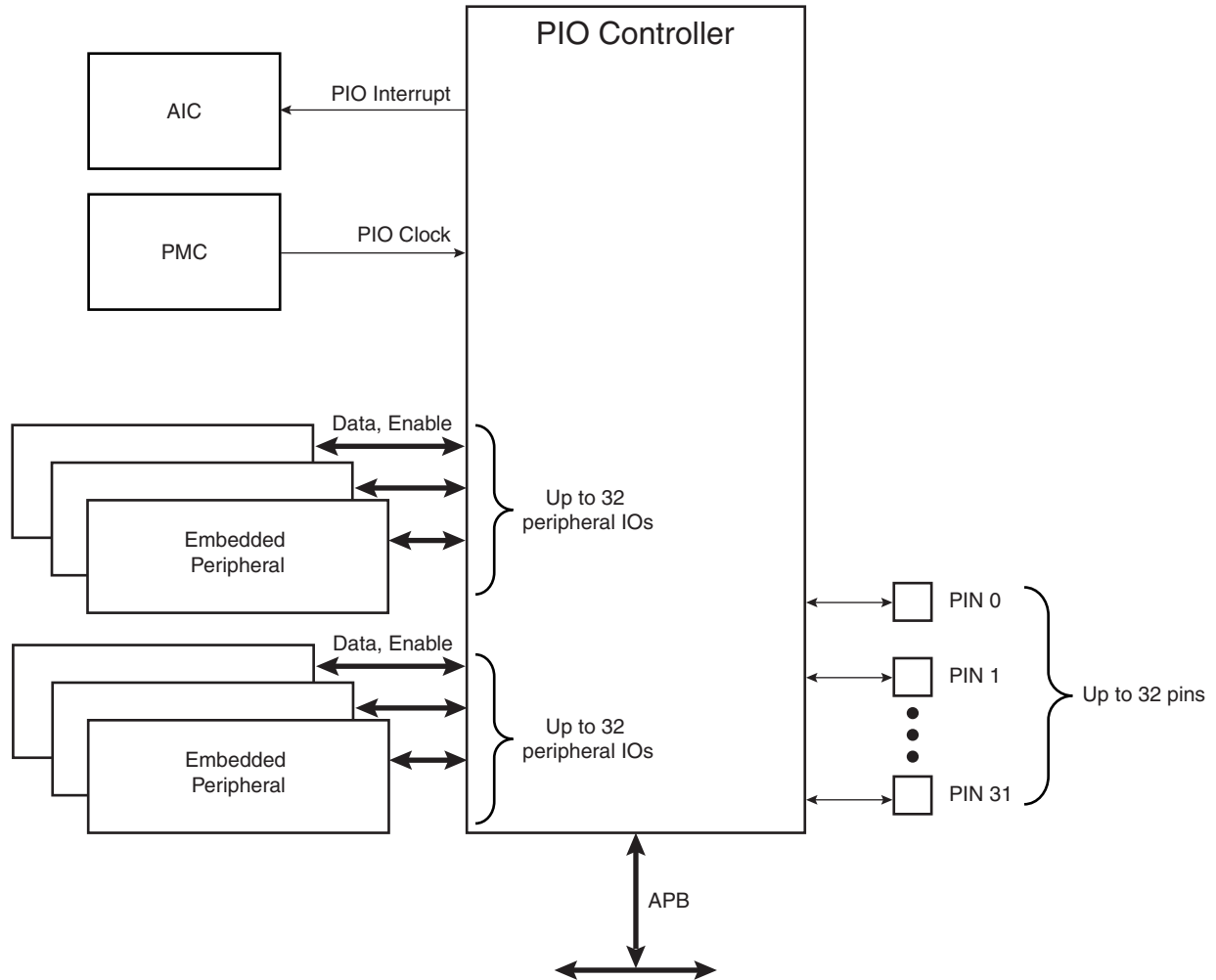
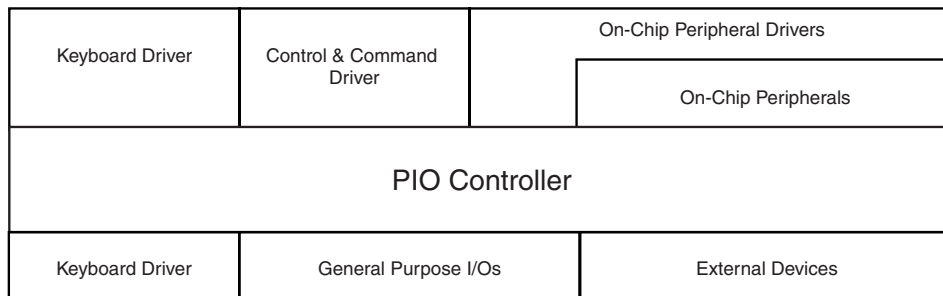


Figure 26-2. Application Block Diagram



## 26.3 Product Dependencies

### 26.3.1 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.

### 26.3.2 External Interrupt Lines

The interrupt signals FIQ and IRQ0 to IRQn are most generally multiplexed through the PIO Controllers. However, it is not necessary to assign the I/O line to the interrupt function as the PIO Controller has no effect on inputs and the interrupt lines (FIQ or IRQs) are used only as inputs.

### 26.3.3 Power Management

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

However, when the clock is disabled, not all of the features of the PIO Controller are available. Note that the Input Change Interrupt 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.

### 26.3.4 Interrupt Generation

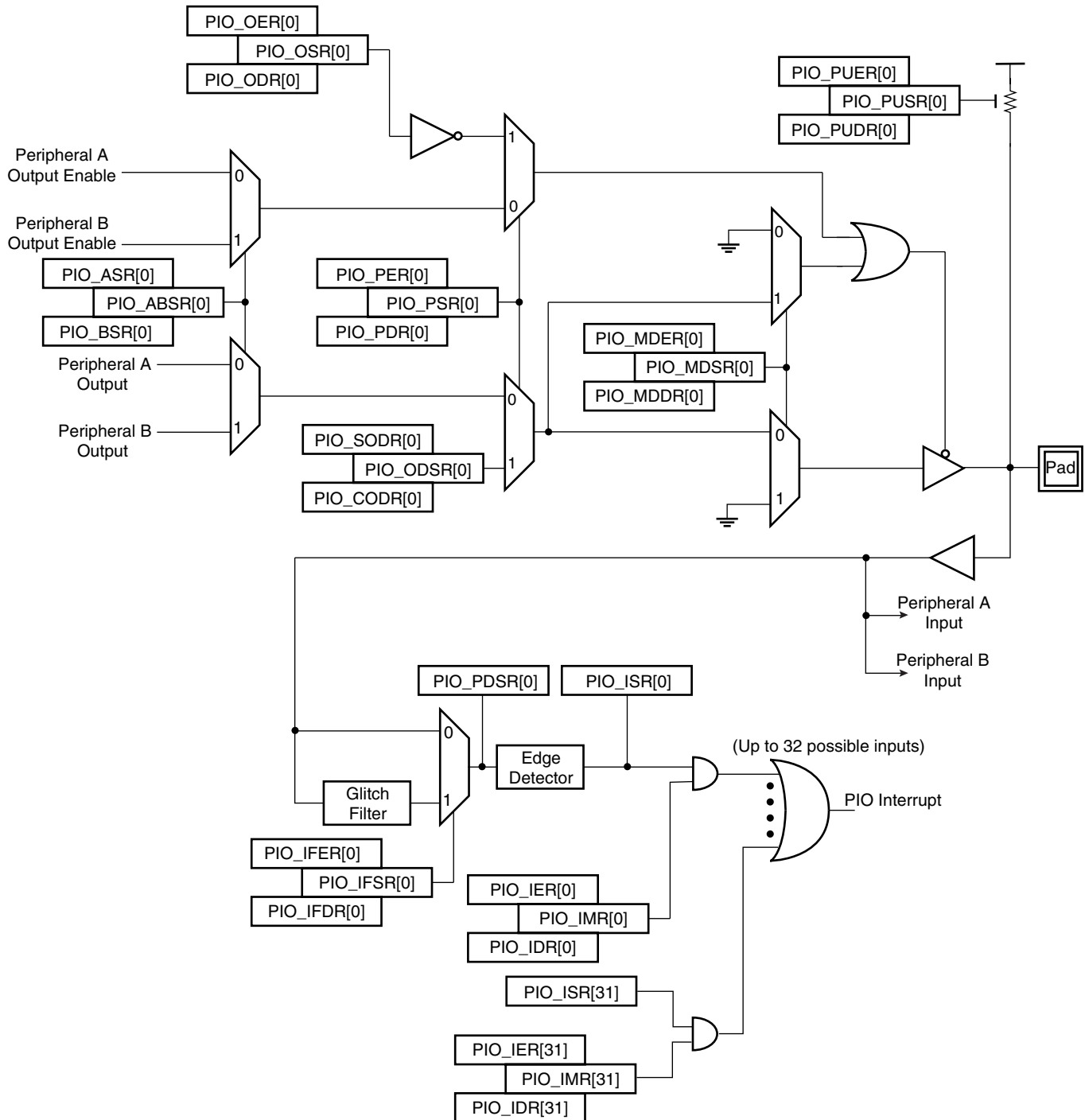
For interrupt handling, the PIO Controllers are considered as user peripherals. This means that the PIO Controller interrupt lines are connected among the interrupt sources 2 to 31. Refer to the PIO Controller peripheral identifier in the product description to identify the interrupt sources dedicated to the PIO Controllers.

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

## 26.4 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 26-3](#). In this description each signal shown represents but one of up to 32 possible indexes.

**Figure 26-3.** I/O Line Control Logic





## 26.4.1 Pull-up Resistor Control

Each I/O line is designed with an embedded pull-up 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 Register). 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.

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.

## 26.4.2 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\_ABSR (AB Select Status Register). 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.

## 26.4.3 Peripheral A or B Selection

The PIO Controller provides multiplexing of up to two peripheral functions on a single pin. The selection is performed by writing PIO\_ASR (A Select Register) and PIO\_BSR (Select B Register). PIO\_ABSR (AB Select Status Register) indicates which peripheral line is currently selected. For each pin, the corresponding bit at level 0 means peripheral A is selected whereas the corresponding bit at level 1 indicates that peripheral B is selected.

Note that multiplexing of peripheral lines A and B only affects the output line. The peripheral input lines are always connected to the pin input.

After reset, PIO\_ABSR is 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\_ASR and PIO\_BSR manages PIO\_ABSR regardless of the configuration of the pin. However, assignment of a pin to a peripheral function requires a write in the corresponding peripheral selection register (PIO\_ASR or PIO\_BSR) in addition to a write in PIO\_PDR.

## 26.4.4 Output Control

When the I/O 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, depending on the value in PIO\_ABSR, 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.

#### 26.4.5 Synchronous Data Output

Controlling all parallel busses using several PIOs requires two successive write operations in the PIO\_SODR and PIO\_CODR registers. This may lead to unexpected transient values. 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\_OSWSR (Output Write Status Register) are written. The mask bits in the 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.

#### 26.4.6 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.

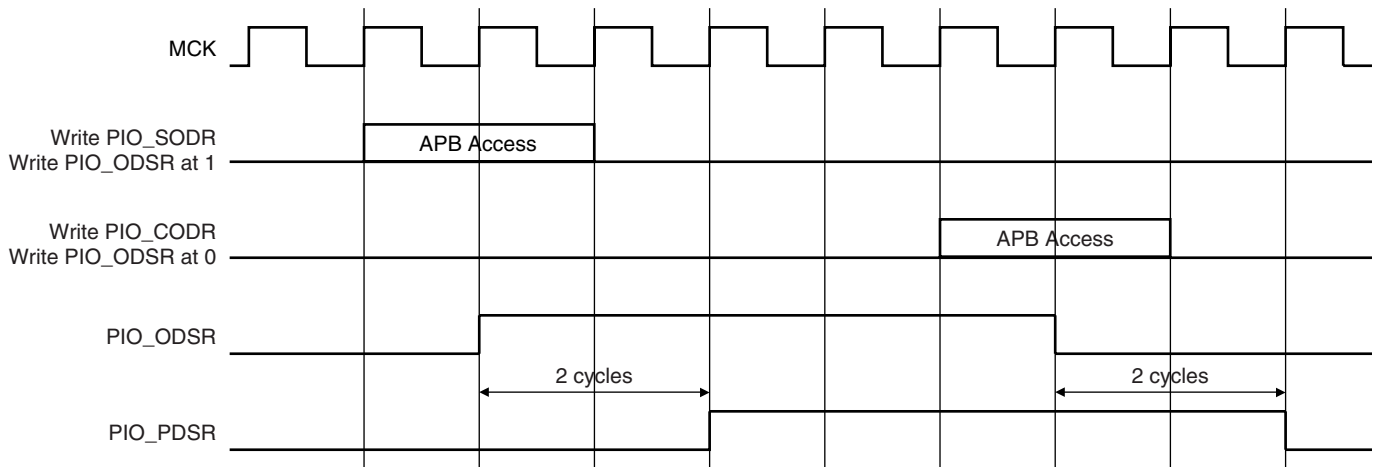
The Multi Drive feature is controlled by PIO\_MDER (Multi-driver Enable Register) and PIO\_MDDR (Multi-driver Disable Register). The Multi Drive can be selected whether the I/O line is controlled by the PIO controller or assigned to a peripheral function. PIO\_MDSR (Multi-driver Status Register) indicates 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.

#### 26.4.7 Output Line Timings

Figure 26-4 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 26-4 also shows when the feedback in PIO\_PDSR is available.

**Figure 26-4. Output Line Timings**



## 26.4.8 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.

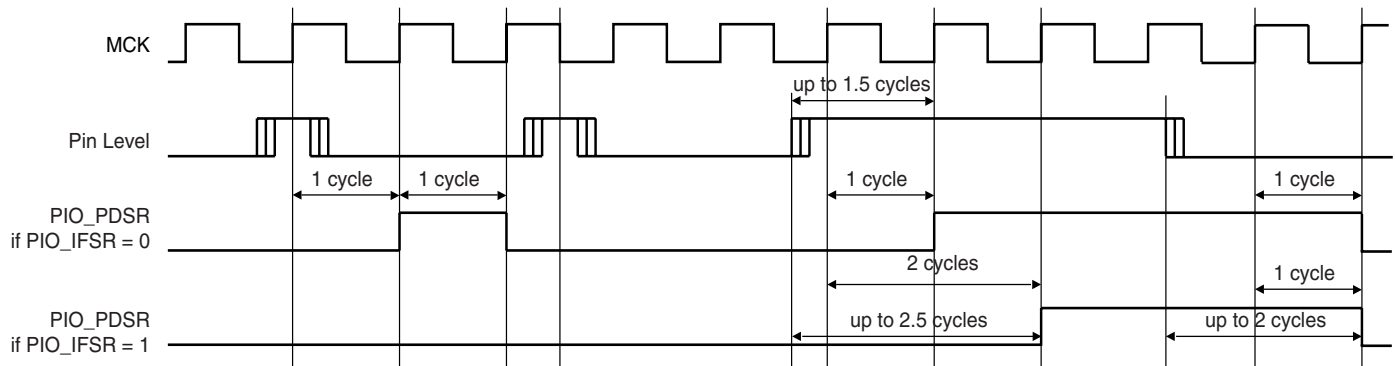
## 26.4.9 Input Glitch Filtering

Optional input glitch filters are independently programmable on each I/O line. When the glitch filter is enabled, a glitch with a duration of less than 1/2 Master Clock (MCK) cycle is automatically rejected, while a pulse with a duration of 1 Master Clock cycle or more is accepted. For pulse durations between 1/2 Master Clock cycle and 1 Master 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 Master Clock cycle, whereas for a glitch to be reliably filtered out, its duration must not exceed 1/2 Master Clock cycle. The filter introduces one Master Clock cycle latency if the pin level change occurs before a rising edge. However, this latency does not appear if the pin level change occurs before a falling edge. This is illustrated in [Figure 26-5](#).

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 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 filters require that the PIO Controller clock is enabled.

**Figure 26-5.** Input Glitch Filter Timing



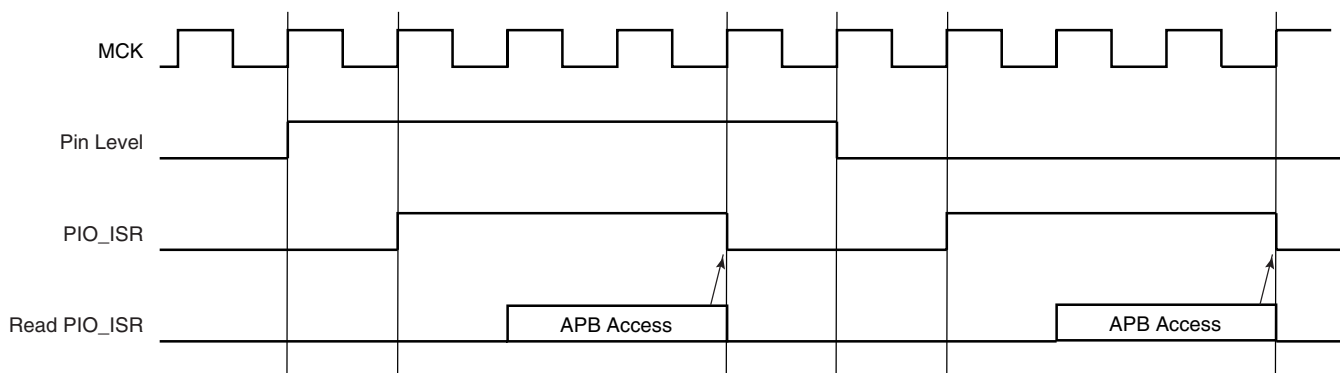
### 26.4.10 Input Change Interrupt

The PIO Controller can be programmed to generate an interrupt when it detects an input change on an I/O line. The Input Change 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.

When an input change 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 Advanced Interrupt Controller.

When the software reads PIO\_ISR, all the interrupts are automatically cleared. This signifies that all the interrupts that are pending when PIO\_ISR is read must be handled.

**Figure 26-6.** Input Change Interrupt Timings



## 26.5 I/O Lines Programming Example

The programming example as shown in [Table 26-1](#) below is used to define 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
- 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, no pull-up resistor
- I/O line 24 to 27 assigned to peripheral A with Input Change Interrupt and pull-up resistor

**Table 26-1.** Programming Example

Register	Value to be Written
PIO_PER	0x0000 FFFF
PIO_PDR	0x0FFF 0000
PIO_OER	0x0000 00FF
PIO_ODR	0x0FFF FF00
PIO_IFER	0x0000 0F00
PIO_IFDR	0x0FFF F0FF
PIO_SODR	0x0000 0000
PIO_CODR	0x0FFF FFFF
PIO_IER	0x0F00 0F00
PIO_IDR	0x00FF F0FF
PIO_MDER	0x0000 000F
PIO_MDDR	0x0FFF FFF0
PIO_PUDR	0x00F0 00F0
PIO_PUER	0x0F0F FF0F
PIO_ASR	0x0F0F 0000
PIO_BSR	0x00F0 0000
PIO_OWER	0x0000 000F
PIO_OWDR	0x0FFF FFF0

## 26.6 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 26-2.** Register Mapping

Offset	Register	Name	Access	Reset Value
0x0000	PIO Enable Register	PIO_PER	Write-only	–
0x0004	PIO Disable Register	PIO_PDR	Write-only	–
0x0008	PIO Status Register <sup>(1)</sup>	PIO_PSR	Read-only	0x0000 0000
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 <sup>(2)</sup>	PIO_ODSR	Read-only	0x0000 0000
0x003C	Pin Data Status Register <sup>(3)</sup>	PIO_PDSR	Read-only	
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 <sup>(4)</sup>	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			

**Table 26-2.** Register Mapping (Continued)

Offset	Register	Name	Access	Reset Value
0x0070	Peripheral A Select Register <sup>(5)</sup>	PIO_ASR	Write-only	–
0x0074	Peripheral B Select Register <sup>(5)</sup>	PIO_BSR	Write-only	–
0x0078	AB Status Register <sup>(5)</sup>	PIO_ABSR	Read-only	0x00000000
0x007C to 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			

- 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.
  4. PIO\_ISR is reset at 0x0. However, the first read of the register may read a different value as input changes may have occurred.
  5. Only this set of registers clears the status by writing 1 in the first register and sets the status by writing 1 in the second register.



### 26.6.1 PIO Controller PIO Enable Register

**Name:** PIO\_PER

**Access Type:** 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 Enable**

0 = No effect.

1 = Enables the PIO to control the corresponding pin (disables peripheral control of the pin).

### 26.6.2 PIO Controller PIO Disable Register

**Name:** PIO\_PDR

**Access Type:** 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 Disable**

0 = No effect.

1 = Disables the PIO from controlling the corresponding pin (enables peripheral control of the pin).



## 26.6.3 PIO Controller PIO Status Register

**Name:** PIO\_PSR

**Access Type:** 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).

## 26.6.4 PIO Controller Output Enable Register

**Name:** PIO\_OER

**Access Type:** 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: Output Enable**

0 = No effect.

1 = Enables the output on the I/O line.



### 26.6.5 PIO Controller Output Disable Register

Name: PIO\_ODR

Access Type: 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: Output Disable**

0 = No effect.

1 = Disables the output on the I/O line.

### 26.6.6 PIO Controller Output Status Register

Name: PIO\_OSR

Access Type: 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.

## 26.6.7 PIO Controller Input Filter Enable Register

**Name:** PIO\_IFER

**Access Type:** 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 Filter Enable**

0 = No effect.

1 = Enables the input glitch filter on the I/O line.

## 26.6.8 PIO Controller Input Filter Disable Register

**Name:** PIO\_IFDR

**Access Type:** 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 Filter Disable**

0 = No effect.

1 = Disables the input glitch filter on the I/O line.



### 26.6.9 PIO Controller Input Filter Status Register

Name: PIO\_IFSR

Access Type: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Input Filter Status**

0 = The input glitch filter is disabled on the I/O line.

1 = The input glitch filter is enabled on the I/O line.

### 26.6.10 PIO Controller Set Output Data Register

Name: PIO\_SODR

Access Type: 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.

## 26.6.11 PIO Controller Clear Output Data Register

**Name:** PIO\_CODR

**Access Type:** 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 = Clears the data to be driven on the I/O line.

## 26.6.12 PIO Controller Output Data Status Register

**Name:** PIO\_ODSR

**Access Type:** 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.



### 26.6.13 PIO Controller Pin Data Status Register

Name: PIO\_PDSR

Access Type: 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.

### 26.6.14 PIO Controller Interrupt Enable Register

Name: PIO\_IER

Access Type: 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.

## 26.6.15 PIO Controller Interrupt Disable Register

**Name:** PIO\_IDR

**Access Type:** 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.

## 26.6.16 PIO Controller Interrupt Mask Register

**Name:** PIO\_IMR

**Access Type:** 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.



### 26.6.17 PIO Controller Interrupt Status Register

Name: PIO\_ISR

Access Type: 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.

### 26.6.18 PIO Multi-driver Enable Register

Name: PIO\_MDER

Access Type: 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: Multi Drive Enable.**

0 = No effect.

1 = Enables Multi Drive on the I/O line.



## 26.6.19 PIO Multi-driver Disable Register

**Name:** PIO\_MDDR

**Access Type:** 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: Multi Drive Disable.**

0 = No effect.

1 = Disables Multi Drive on the I/O line.

## 26.6.20 PIO Multi-driver Status Register

**Name:** PIO\_MDSR

**Access Type:** 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.



### 26.6.21 PIO Pull Up Disable Register

Name: PIO\_PUDR

Access Type: 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: Pull Up Disable.**

0 = No effect.

1 = Disables the pull up resistor on the I/O line.

### 26.6.22 PIO Pull Up Enable Register

Name: PIO\_PUER

Access Type: 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: Pull Up Enable.**

0 = No effect.

1 = Enables the pull up resistor on the I/O line.

## 26.6.23 PIO Pull Up Status Register

**Name:** PIO\_PUSR

**Access Type:** 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.

## 26.6.24 PIO Peripheral A Select Register

**Name:** PIO\_ASR

**Access Type:** Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Peripheral A Select.**

0 = No effect.

1 = Assigns the I/O line to the Peripheral A function.



### 26.6.25 PIO Peripheral B Select Register

Name: PIO\_BSR

Access Type: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Peripheral B Select.**

0 = No effect.

1 = Assigns the I/O line to the peripheral B function.

### 26.6.26 PIO Peripheral A B Status Register

Name: PIO\_ABSR

Access Type: 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 A B Status.**

0 = The I/O line is assigned to the Peripheral A.

1 = The I/O line is assigned to the Peripheral B.

## 26.6.27 PIO Output Write Enable Register

**Name:** PIO\_OWER

**Access Type:** 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: Output Write Enable.**

0 = No effect.

1 = Enables writing PIO\_ODSR for the I/O line.

## 26.6.28 PIO Output Write Disable Register

**Name:** PIO\_OWDR

**Access Type:** 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: Output Write Disable.**

0 = No effect.

1 = Disables writing PIO\_ODSR for the I/O line.



### 26.6.29 PIO Output Write Status Register

Name: PIO\_OWSR

Access Type: 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. Serial Peripheral Interface (SPI)

### 27.1 Overview

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.

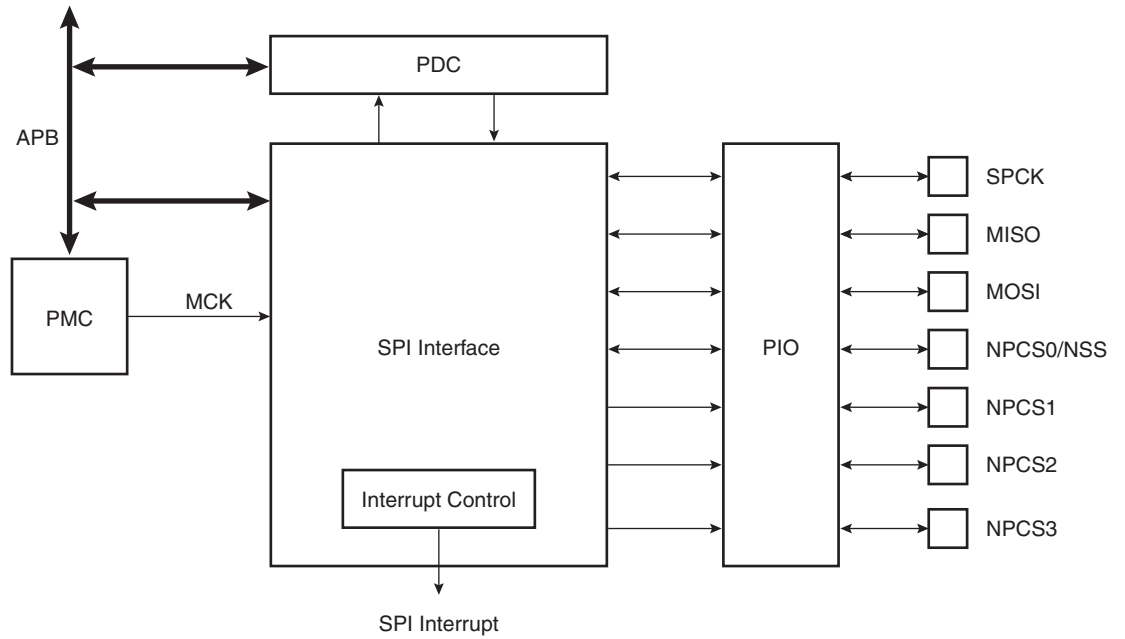
A slave device is selected when the master asserts its NSS signal. If multiple slave devices exist, the master generates a separate slave select signal for each slave (NPCS).

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.

## 27.2 Block Diagram

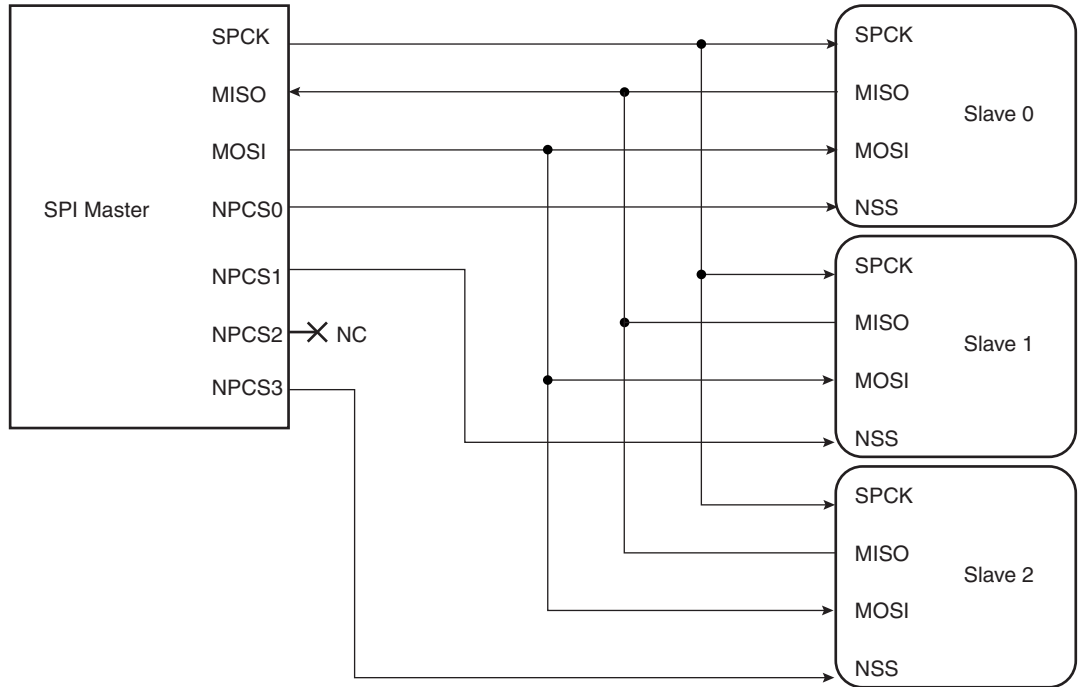
Figure 27-1. Block Diagram





27.3 Application Block Diagram

Figure 27-2. Application Block Diagram: Single Master/Multiple Slave Implementation



## 27.4 Signal Description

**Table 27-1.** 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

## 27.5 Product Dependencies

### 27.5.1 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 SPI pins to their peripheral functions.

### 27.5.2 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.

### 27.5.3 Interrupt

The SPI interface has an interrupt line connected to the Advanced Interrupt Controller (AIC). Handling the SPI interrupt requires programming the AIC before configuring the SPI.

## 27.6 Functional Description

### 27.6.1 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.

### 27.6.2 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.

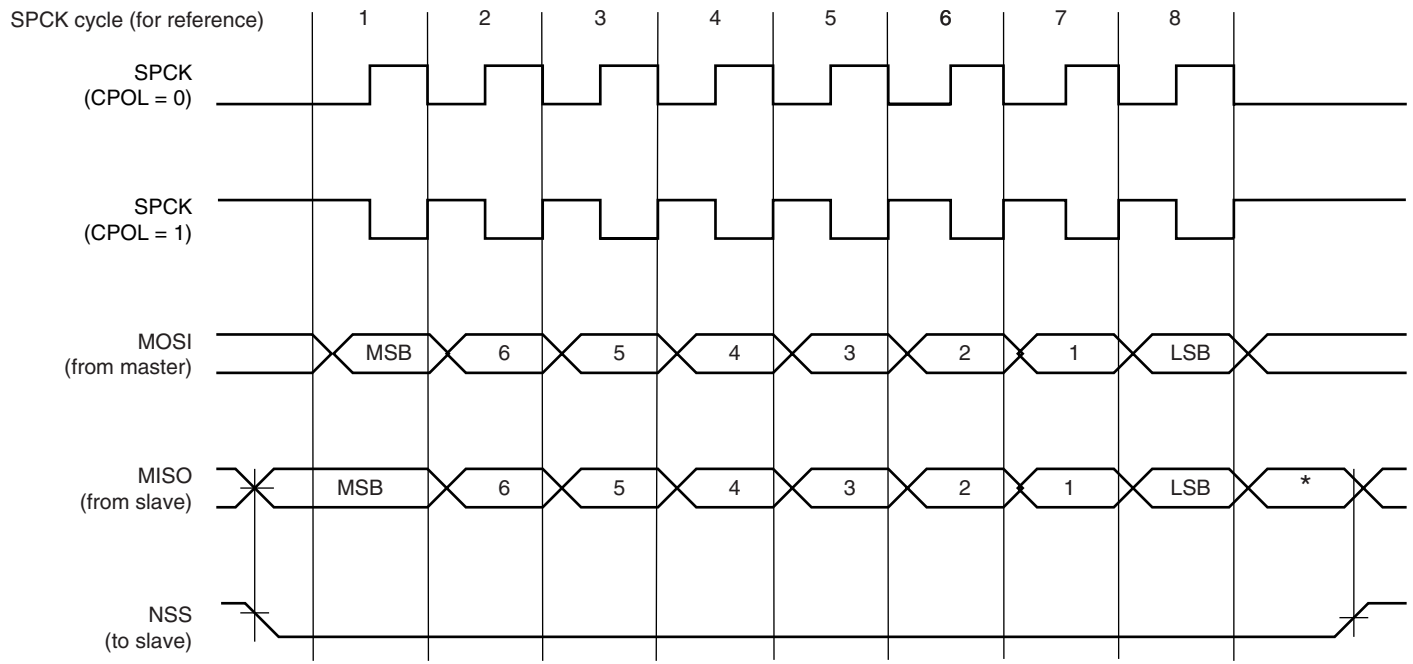
[Table 27-2](#) shows the four modes and corresponding parameter settings.

**Table 27-2.** SPI Bus Protocol Mode

SPI Mode	CPOL	NCPHA
0	0	1
1	0	0
2	1	1
3	1	0

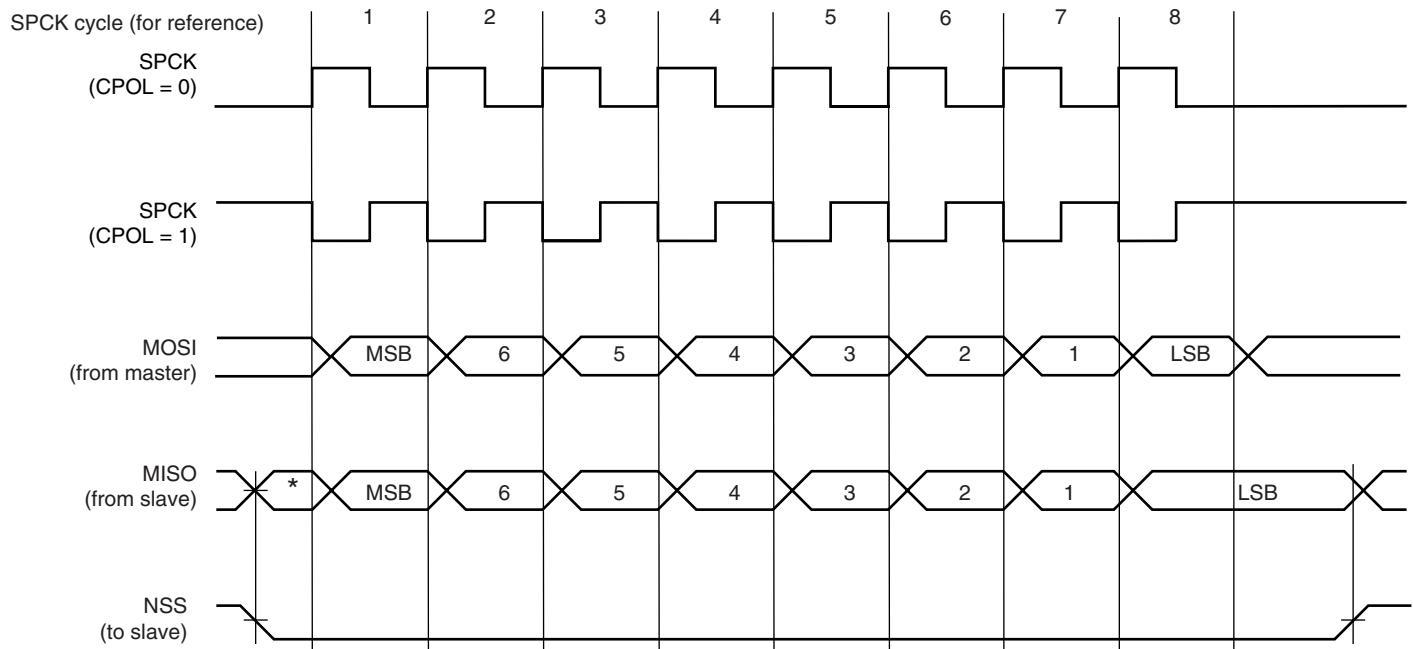
[Figure 27-3](#) and [Figure 27-4](#) show examples of data transfers.

**Figure 27-3.** SPI Transfer Format (NCPHA = 1, 8 bits per transfer)



\* Not defined, but normally MSB of previous character received.

**Figure 27-4.** SPI Transfer Format (NCPHA = 0, 8 bits per transfer)



\* Not defined but normally LSB of previous character transmitted.

## 27.6.3 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. 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.

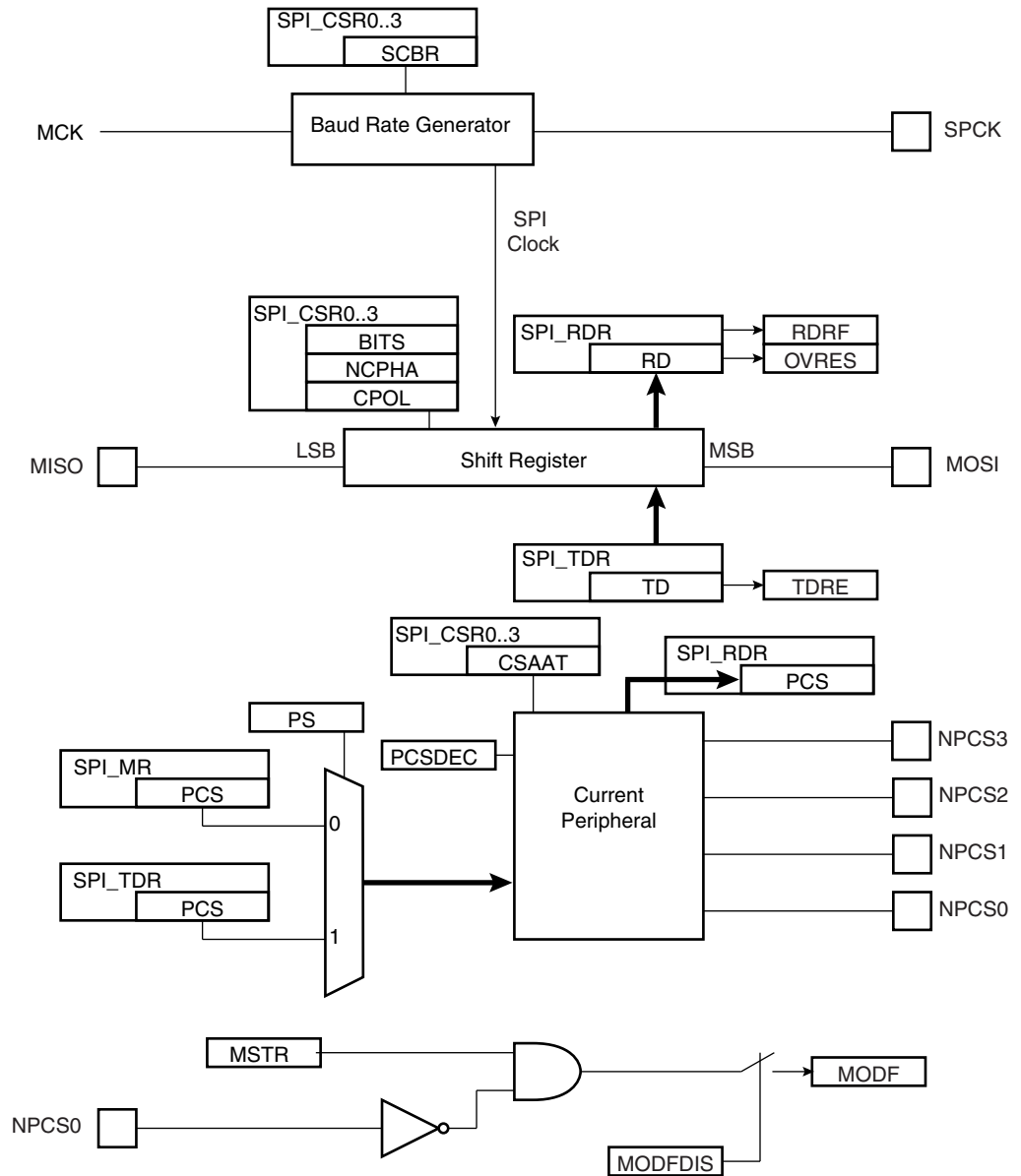
The transfer of received data from the Shift Register in SPI\_RDR is indicated by the RDRF bit (Receive Data Register Full) in the Status Register (SPI\_SR). When the received data is read, the RDRF bit is cleared.

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 27-5 on page 238](#) shows a block diagram of the SPI when operating in Master Mode. [Figure 27-6 on page 239](#) shows a flow chart describing how transfers are handled.

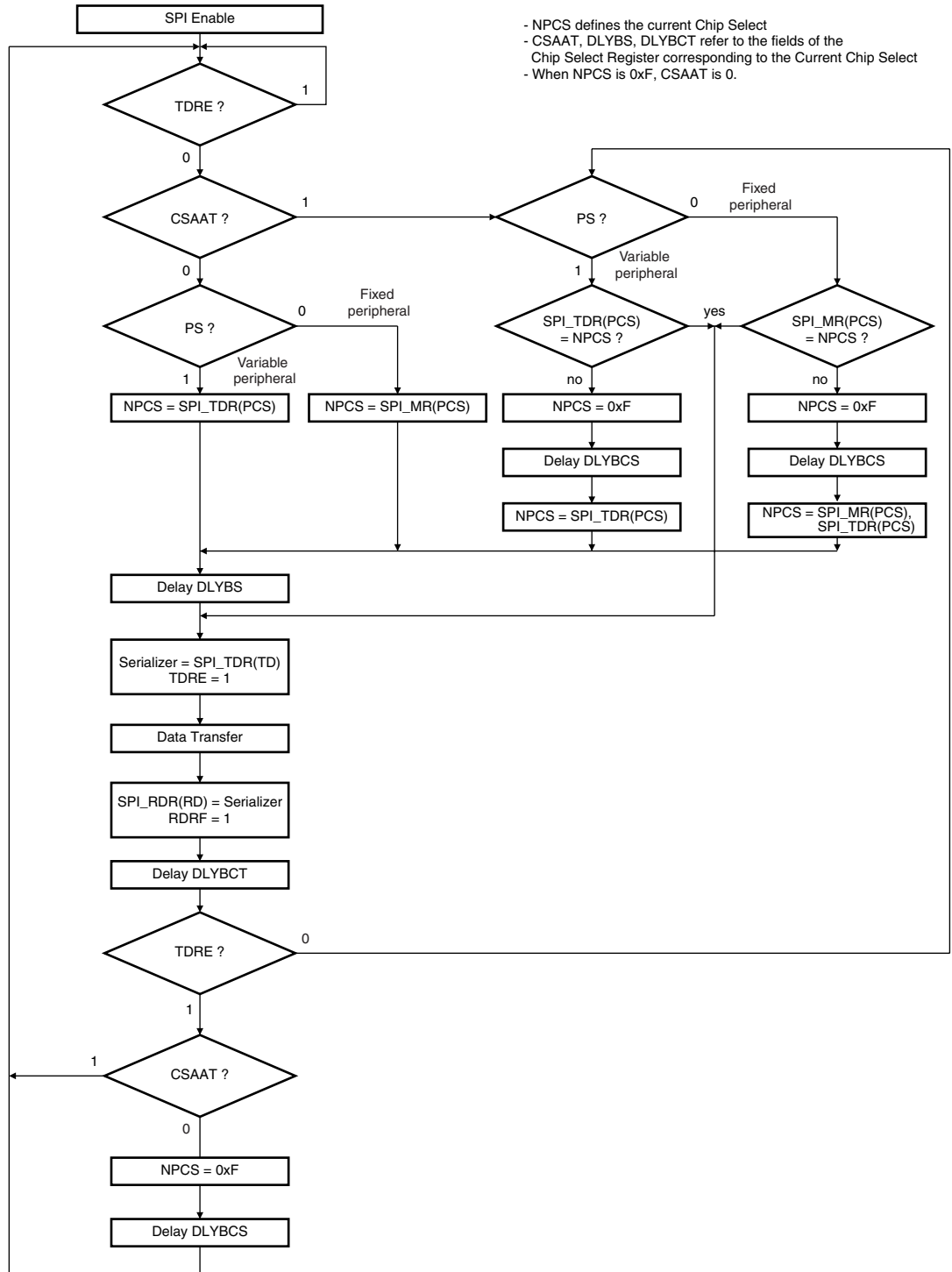
27.6.3.1 Master Mode Block Diagram

Figure 27-5. Master Mode Block Diagram



## 27.6.3.2 Master Mode Flow Diagram

**Figure 27-6.** Master Mode Flow Diagram



### 27.6.3.3 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.

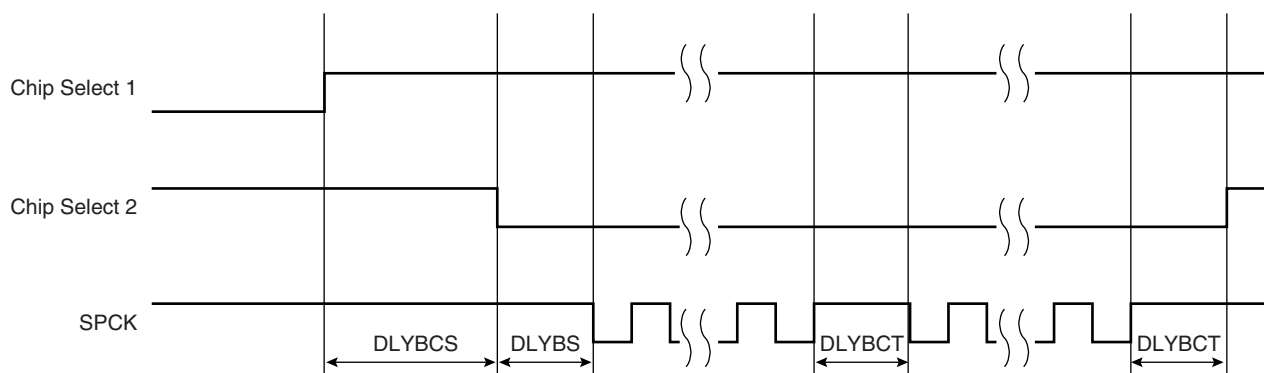
### 27.6.3.4 Transfer Delays

Figure 27-7 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 27-7.** Programmable Delays



### 27.6.3.5 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.

The peripheral selection can be performed in two different ways:



- Fixed Peripheral Select: SPI exchanges data with only one peripheral
- Variable Peripheral Select: Data can be exchanged with more than 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 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 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 to 16) 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.

### 27.6.3.6 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 an external logic. This can be enabled by writing the PCS-DEC 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. 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 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\_CRSD0 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.

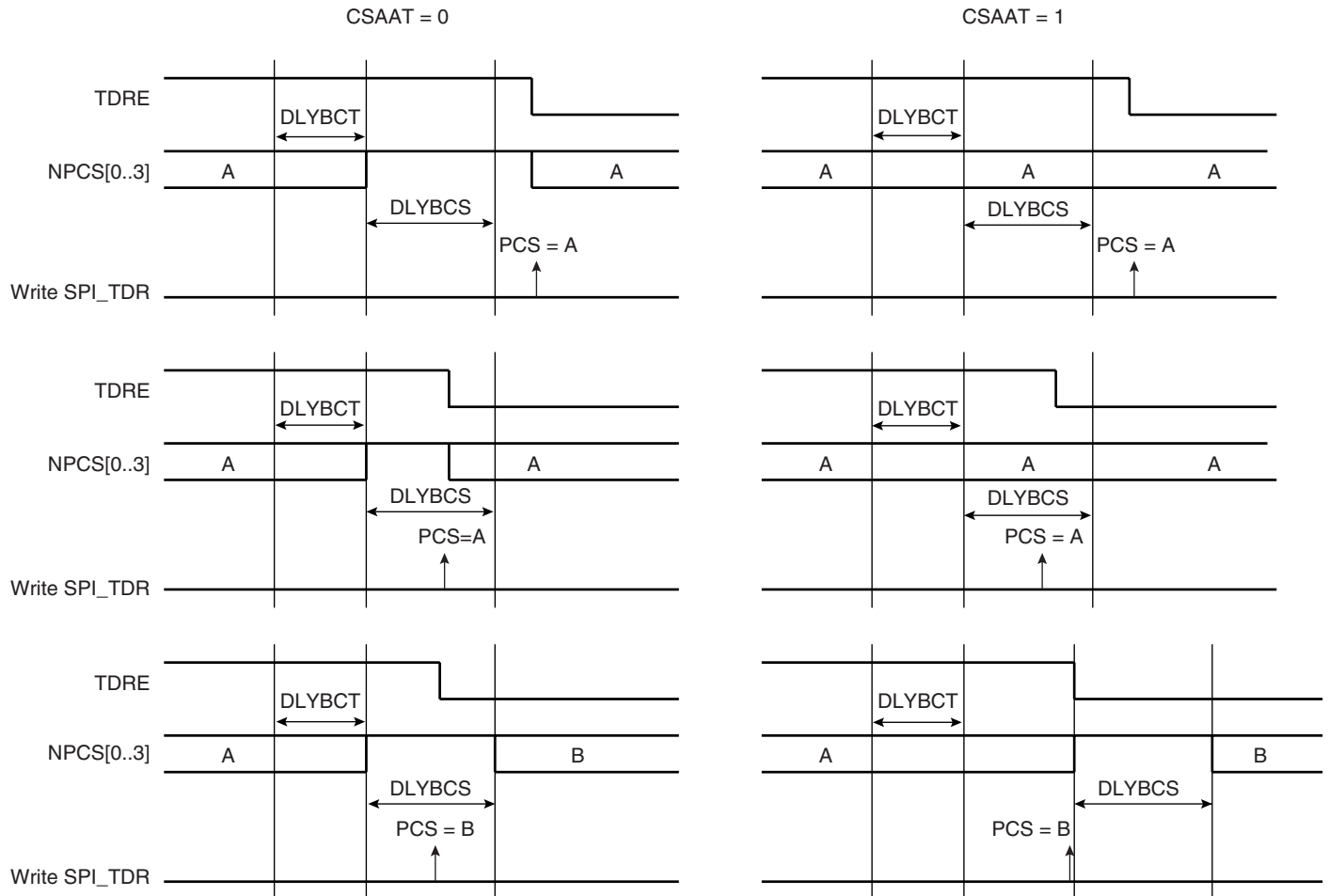
### 27.6.3.7 Peripheral Deselection

When operating normally, as soon as the transfer of the last data written in SPI\_TDR is completed, the NPCS lines all rise. This might lead to runtime error if the processor is too long in responding to an interrupt, and thus might lead to difficulties for interfacing with some serial peripherals requiring the chip select line to remain active during a full set of transfers.

To facilitate interfacing with such devices, the Chip Select Register 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 peripheral is required.

Figure 27-8 shows different peripheral deselection cases and the effect of the CSAAT bit.

**Figure 27-8.** Peripheral Deselection



### 27.6.3.8 Mode Fault Detection

A mode fault is detected when the SPI is programmed in Master Mode and a low level is driven by an external master on the NPCS0/NSS signal. NPCS0, MOSI, MISO and SPCK must be configured in open drain through the PIO controller, so that external pull up resistors are needed to guarantee high level.

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).

## 27.6.4 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.

When all the bits are processed, the received data is transferred in the Receive Data Register and the RDRF bit rises. If RDRF is already high when the data is transferred, the Overrun bit rises and the data transfer to SPI\_RDR is aborted.

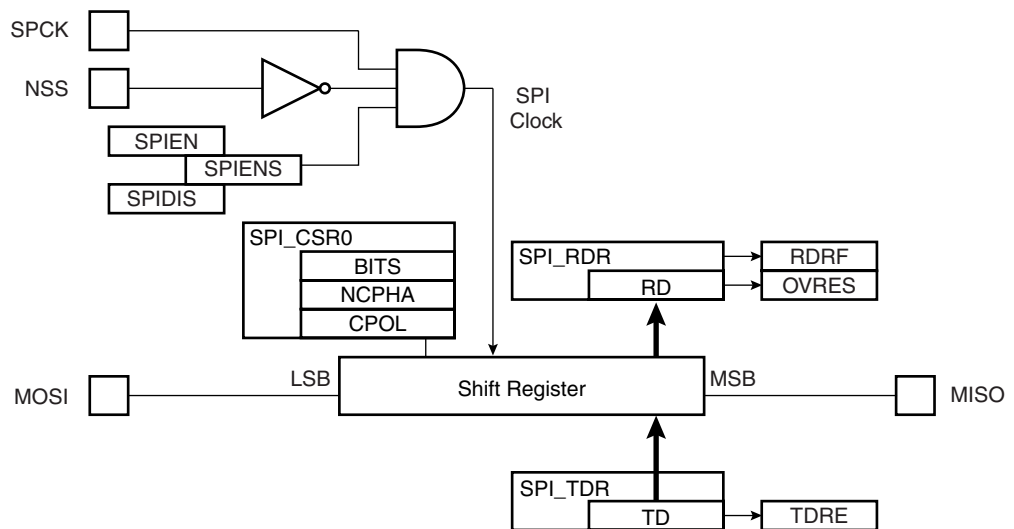
When a transfer starts, the data shifted out is the data present in the Shift Register. If no data has been written in the Transmit Data Register (SPI\_TDR), the last data received is transferred. If no data has been received since the last reset, all bits are transmitted low, as the Shift Register resets 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.

Figure 27-9 shows a block diagram of the SPI when operating in Slave Mode.

**Figure 27-9.** Slave Mode Functional Block Diagram



## 27.7 Serial Peripheral Interface (SPI) User Interface

**Table 27-3.** SPI Register Mapping

Offset	Register	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
0x004C - 0x00F8	Reserved	–	–	–
0x004C - 0x00FC	Reserved	–	–	–
0x100 - 0x124	Reserved for the PDC			

## 27.7.1 SPI Control Register

**Name:** SPI\_CR

**Access Type:** 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.

## 27.7.2 SPI Mode Register

**Name:** SPI\_MR

**Access Type:** 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	–	–	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.

- **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:

$$\text{Delay Between Chip Selects} = \frac{DLYBCS}{MCK}$$

### 27.7.3 SPI Receive Data Register

**Name:** SPI\_RDR

**Access Type:** 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.



## 27.7.4 SPI Transmit Data Register

**Name:** SPI\_TDR

**Access Type:** 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).

### 27.7.5 SPI Status Register

**Name:** SPI\_SR

**Access Type:** 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
–	–	–	–	–	–	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<sup>(1)</sup> or SPI\_RNCR<sup>(1)</sup>.

1 = The Receive Counter Register has reached 0 since the last write in SPI\_RCR<sup>(1)</sup> or SPI\_RNCR<sup>(1)</sup>.

- **ENDTX: End of TX buffer**

0 = The Transmit Counter Register has not reached 0 since the last write in SPI\_TCR<sup>(1)</sup> or SPI\_TNCR<sup>(1)</sup>.

1 = The Transmit Counter Register has reached 0 since the last write in SPI\_TCR<sup>(1)</sup> or SPI\_TNCR<sup>(1)</sup>.

- **RXBUFF: RX Buffer Full**

0 = SPI\_RCR<sup>(1)</sup> or SPI\_RNCR<sup>(1)</sup> has a value other than 0.

1 = Both SPI\_RCR<sup>(1)</sup> and SPI\_RNCR<sup>(1)</sup> have a value of 0.

- **TXBUFE: TX Buffer Empty**

0 = SPI\_TCR<sup>(1)</sup> or SPI\_TNCR<sup>(1)</sup> has a value other than 0.

1 = Both SPI\_TCR<sup>(1)</sup> and SPI\_TNCR<sup>(1)</sup> 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.

- **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.



### 27.7.6 SPI Interrupt Enable Register

Name: SPI\_IER

Access Type: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

- **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**
- **TXEMPTY: Transmission Registers Empty Enable**
- **NSSR: NSS Rising Interrupt Enable**

0 = No effect.

1 = Enables the corresponding interrupt.

## 27.7.7 SPI Interrupt Disable Register

**Name:** SPI\_IDR

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

- **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**
- **TXEMPTY: Transmission Registers Empty Disable**
- **NSSR: NSS Rising Interrupt Disable**

0 = No effect.

1 = Disables the corresponding interrupt.

### 27.7.8 SPI Interrupt Mask Register

**Name:** SPI\_IMR

**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
–	–	–	–	–	–	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

- **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**
- **TXEMPTY: Transmission Registers Empty Mask**
- **NSSR: NSS Rising Interrupt Mask**

0 = The corresponding interrupt is not enabled.

1 = The corresponding interrupt is enabled.

## 27.7.9 SPI Chip Select Register

**Name:** SPI\_CSR0... SPI\_CSR3

**Access Type:** 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	-	NCPHA	CPOL

- **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.

- **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**

The BITS field determines the number of data bits transferred. Reserved values should not be used.

BITS	Bits Per Transfer
0000	8
0001	9
0010	10
0011	11
0100	12
0101	13
0110	14
0111	15
1000	16
1001	Reserved
1010	Reserved
1011	Reserved
1100	Reserved
1101	Reserved
1110	Reserved
1111	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.

- **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:

$$\text{Delay Between Consecutive Transfers} = \frac{32 \times DLYBCT}{MCK}$$



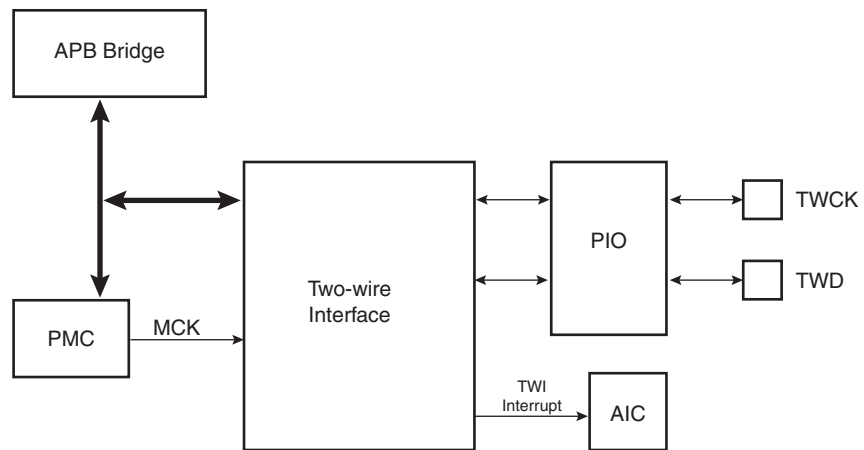
## 28. Two-wire Interface (TWI)

### 28.1 Overview

The Two-wire Interface (TWI) interconnects components on a unique two-wire bus, made up of one clock line and one data line with speeds of up to 400 Kbits per second, based on a byte-oriented transfer format. It can be used with any Atmel two-wire bus Serial EEPROM. The TWI is programmable as a master with sequential or single-byte access. A configurable baud rate generator permits the output data rate to be adapted to a wide range of core clock frequencies.

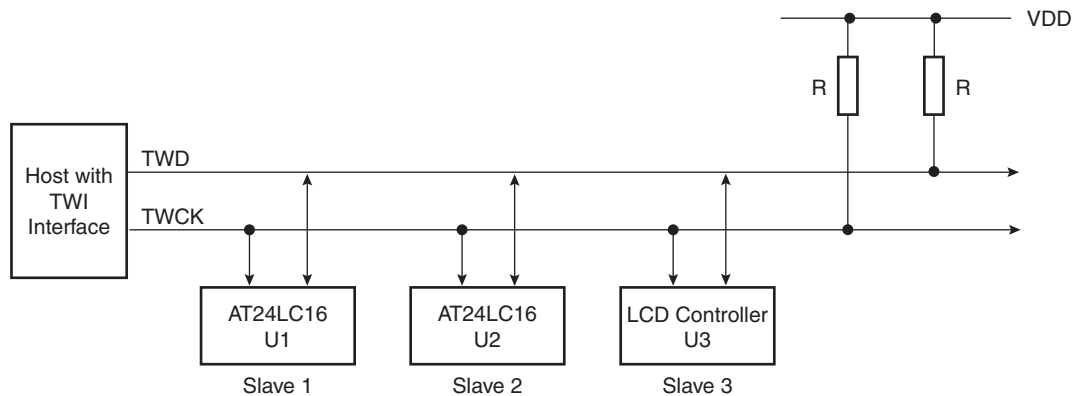
### 28.2 Block Diagram

Figure 28-1. Block Diagram



### 28.3 Application Block Diagram

Figure 28-2. Application Block Diagram



## 28.4 Product Dependencies

### 28.4.1 I/O Lines Description

**Table 28-1.** I/O Lines Description

Pin Name	Pin Description	Type
TWD	Two-wire Serial Data	Input/Output
TWCK	Two-wire Serial Clock	Input/Output

Both TWD and TWCK are bidirectional lines, connected to a positive supply voltage via a current source or pull-up resistor (see [Figure 28-2 on page 257](#)). 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 steps:

- Program the PIO controller to:
  - Dedicate TWD and TWCK as peripheral lines.
  - Define TWD and TWCK as open-drain.

### 28.4.2 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.

### 28.4.3 Interrupt

The TWI interface has an interrupt line connected to the Advanced Interrupt Controller (AIC). In order to handle interrupts, the AIC must be programmed before configuring the TWI.

## 28.5 Functional Description

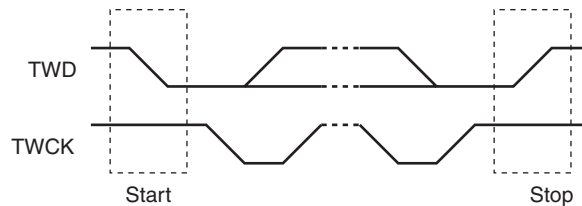
### 28.5.1 Transfer Format

The data put on the TWD line must be 8 bits long. Data is transferred MSB first; each byte must be followed by an acknowledgement. The number of bytes per transfer is unlimited (see [Figure 28-4 on page 259](#)).

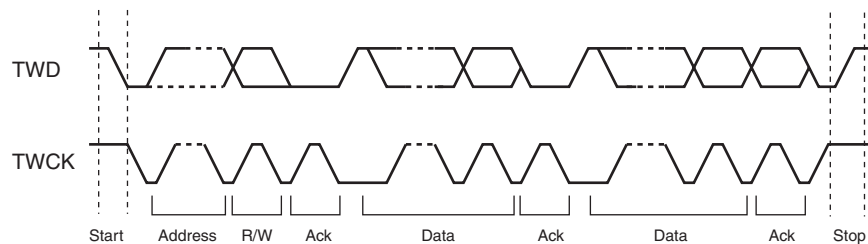
Each transfer begins with a START condition and terminates with a STOP condition (see [Figure 28-3 on page 259](#)).

- A high-to-low transition on the TWD line while TWCK is high defines the START condition.
- A low-to-high transition on the TWD line while TWCK is high defines a STOP condition.

**Figure 28-3.** START and STOP Conditions



**Figure 28-4.** Transfer Format



### 28.5.2 Modes of Operation

The TWI has two modes of operation:

- Master transmitter mode
- Master receiver mode

The TWI Control Register (TWI\_CR) allows configuration of the interface in Master Mode. In this mode, it generates the clock according to the value programmed in the Clock Waveform Generator Register (TWI\_CWGR). This register defines the TWCK signal completely, enabling the interface to be adapted to a wide range of clocks.

### 28.5.3 Transmitting Data

After the master initiates a Start condition, 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 (write or read). If this bit is 0, it indicates a write operation (transmit operation). If the bit is 1, it indicates a request for data read (receive operation).

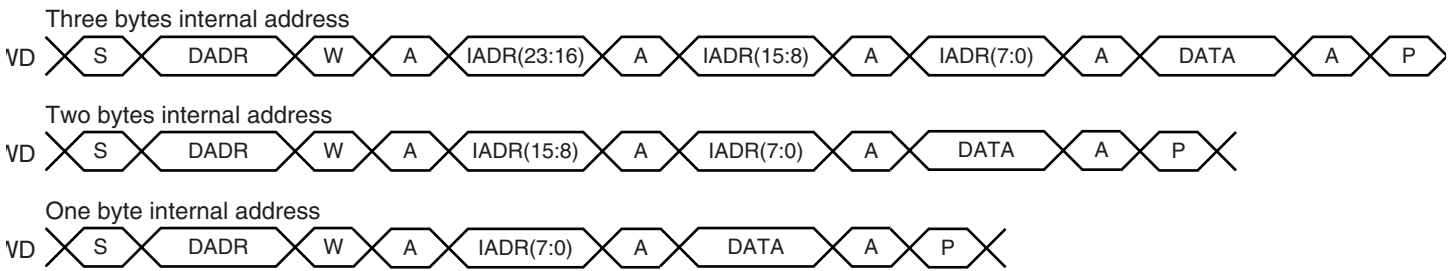
The TWI transfers require the slave to acknowledge each received byte. During the acknowledge clock pulse, the master releases the data line (HIGH), enabling the slave to pull it down in order to generate the acknowledge. The master polls the data line during this clock pulse and

sets the **NAK** bit 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). After writing in the transmit-holding register (TWI\_THR), setting the START bit in the control register starts the transmission. The data is shifted in the internal shifter and when an acknowledge is detected, the TXRDY bit is set until a new write in the TWI\_THR (see Figure 28-6 below). The master generates a stop condition to end the transfer.

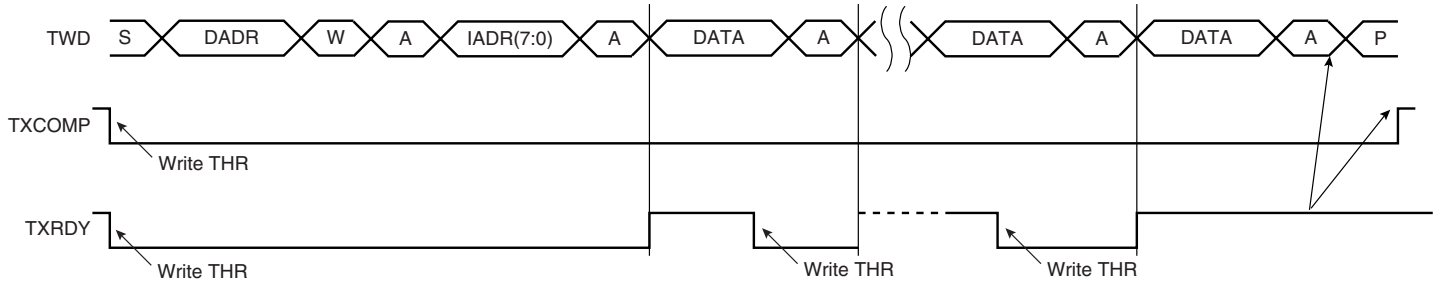
The read sequence begins by setting the START bit. When the RXRDY bit is set in the status register, a character has been received in the receive-holding register (TWI\_RHR). The RXRDY bit is reset when reading the TWI\_RHR.

The TWI interface performs various transfer formats (7-bit slave address, 10-bit slave 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, **IADRSZ** must be set to 0. 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).

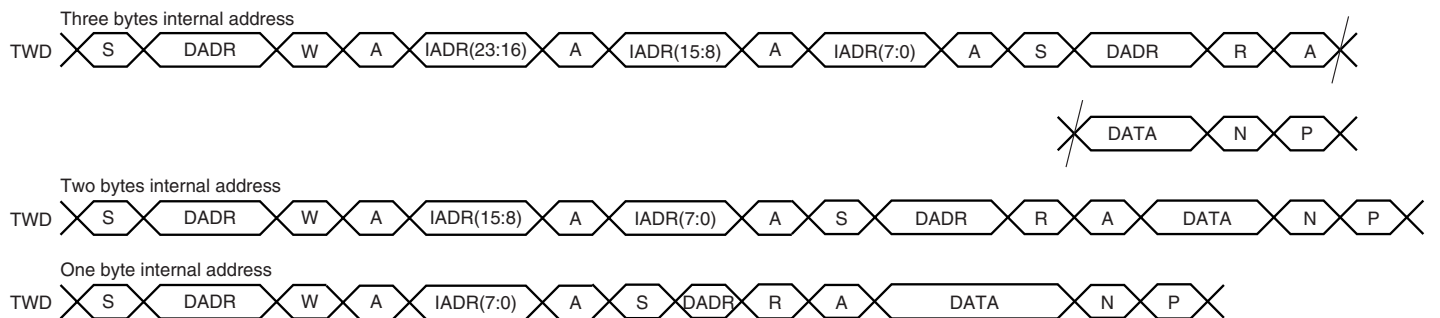
**Figure 28-5.** Master Write with One, Two or Three Bytes Internal Address and One Data Byte



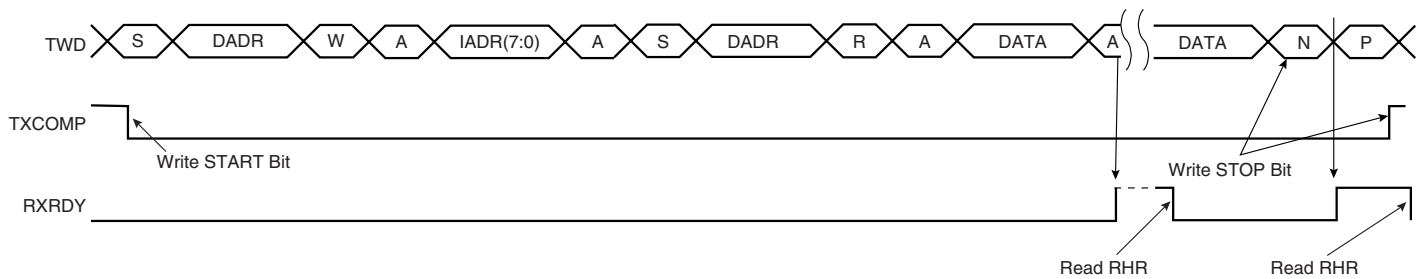
**Figure 28-6.** Master Write with One Byte Internal Address and Multiple Data Bytes



**Figure 28-7.** Master Read with One, Two or Three Bytes Internal Address and One Data Byte



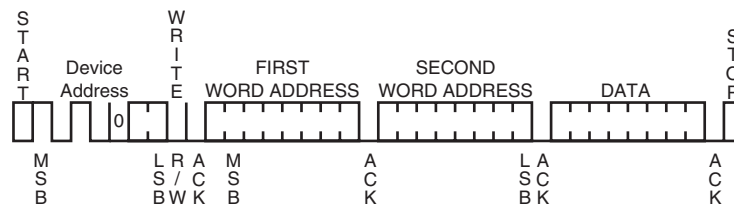
**Figure 28-8.** Master Read with One Byte Internal Address and Multiple Data Bytes



- S = Start
- P = Stop
- W = Write
- R = Read
- A = Acknowledge
- N = Not Acknowledge
- DADR = Device Address
- IADR = Internal Address

Figure 28-9 below shows a byte write to an Atmel AT24LC512 EEPROM. This demonstrates the use of internal addresses to access the device.

**Figure 28-9.** Internal Address Usage



## 28.5.4 Read/Write Flowcharts

The following flowcharts shown in [Figure 28-10 on page 262](#) and in [Figure 28-11 on page 263](#) give examples for read and write operations in Master Mode. A polling or interrupt method can be used to check the status bits. The interrupt method requires that the interrupt enable register (TWI\_IER) be configured first.

**Figure 28-10. TWI Write in Master Mode**

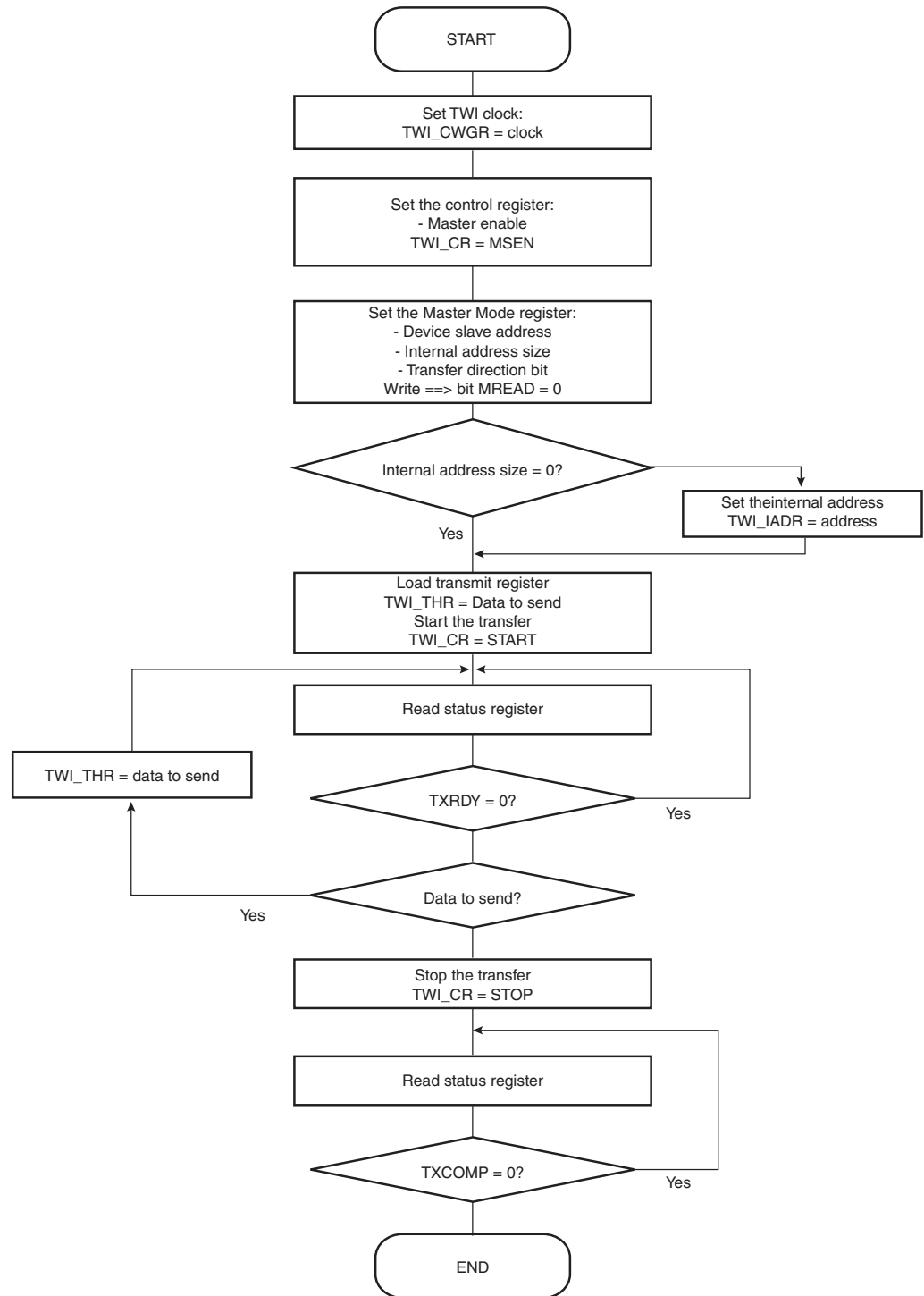
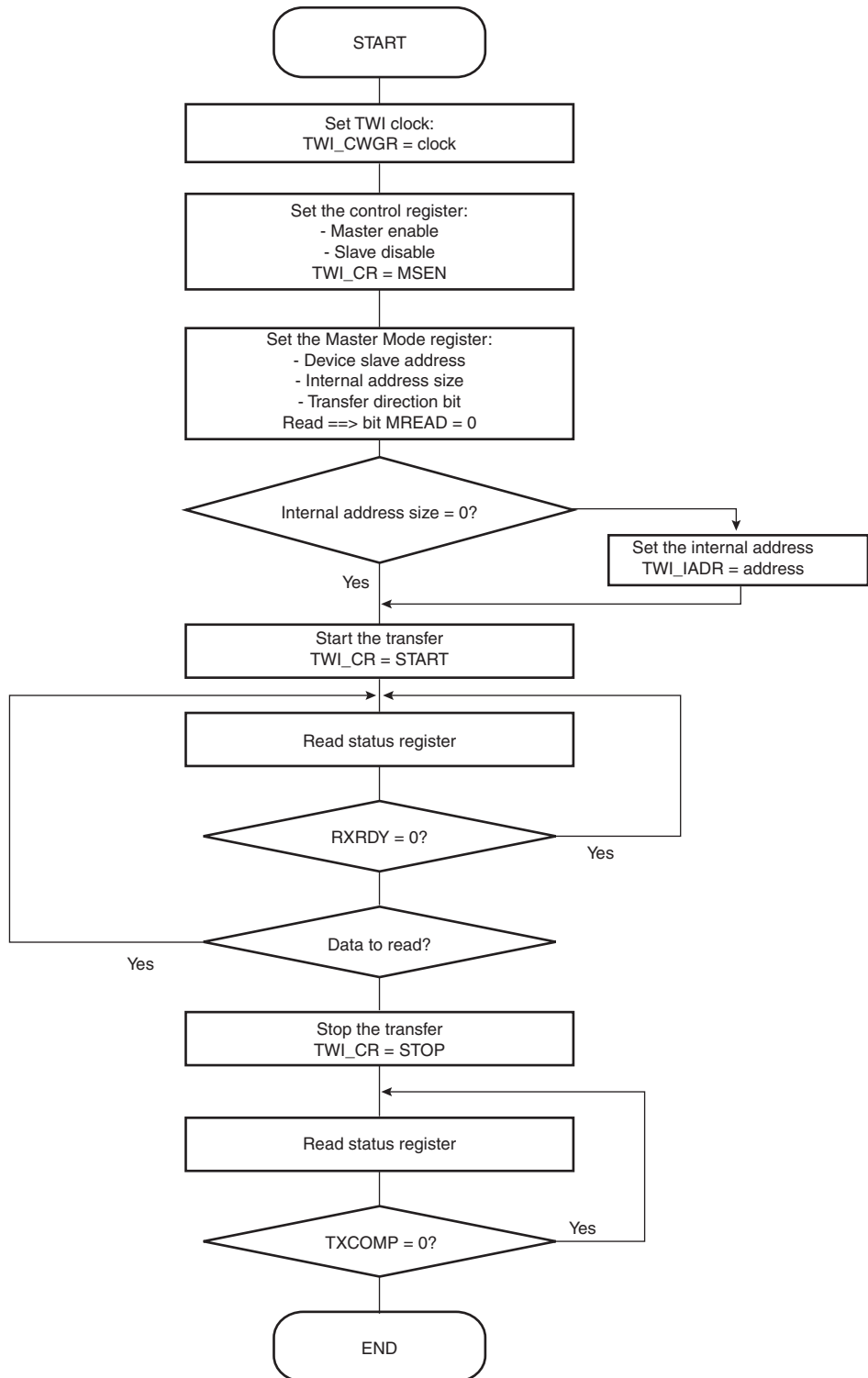


Figure 28-11. TWI Read in Master Mode



## 28.6 Two-wire Interface (TWI) User Interface

**Table 28-2.** Two-wire Interface (TWI) Register Mapping

Offset	Register	Name	Access	Reset Value
0x0000	Control Register	TWI_CR	Write-only	N/A
0x0004	Master Mode Register	TWI_MMR	Read/Write	0x0000
0x0008	Reserved	–	–	–
0x000C	Internal Address Register	TWI_IADR	Read/Write	0x0000
0x0010	Clock Waveform Generator Register	TWI_CWGR	Read/Write	0x0000
0x0020	Status Register	TWI_SR	Read-only	0x0008
0x0024	Interrupt Enable Register	TWI_IER	Write-only	N/A
0x0028	Interrupt Disable Register	TWI_IDR	Write-only	N/A
0x002C	Interrupt Mask Register	TWI_IMR	Read-only	0x0000
0x0030	Receive Holding Register	TWI_RHR	Read-only	0x0000
0x0034	Transmit Holding Register	TWI_THR	Read/Write	0x0000
0x0038-0x00FC	Reserved	–	–	–



## 28.6.1 TWI Control Register

**Register Name:** TWI\_CR

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
SWRST	–	–	–	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 with the mode register as soon as the user writes a character in the holding register.

- **STOP: Send a STOP Condition**

0 = No effect.

1 = STOP Condition is sent just after completing the current byte transmission in master read or write mode.

In single data byte master read or write, the START and STOP must both be set.

In multiple data bytes master read or write, the STOP must be set before ACK/NACK bit transmission.

In master read mode, if a NACK bit is received, the STOP is automatically performed.

In multiple data write operation, when both THR and shift register are empty, a STOP condition is automatically sent.

- **MSEN: TWI Master Transfer Enabled**

0 = No effect.

1 = If MSDIS = 0, the master data transfer is enabled.

- **MSDIS: TWI Master Transfer Disabled**

0 = No effect.

1 = The master data transfer is disabled, all pending data is transmitted. The shifter and holding characters (if they contain data) are transmitted in case of write operation. In read operation, the character being transferred must be completely received before disabling.

- **SWRST: Software Reset**

0 = No effect.

1 = Equivalent to a system reset.

## 28.6.2 TWI Master Mode Register

**Register Name:** TWI\_MMR

**Address Type:** Read/Write

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

- **IADRSZ: Internal Device Address Size**

IADRSZ[9:8]		
0	0	No internal device address (Byte command protocol)
0	1	One-byte internal device address
1	0	Two-byte internal device address
1	1	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 in Master Mode to access slave devices in read or write mode.

## 28.6.3 TWI Internal Address Register

**Register Name:** TWI\_IADR

**Access Type:** Read/Write

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

- **IADR: Internal Address**

0, 1, 2 or 3 bytes depending on IADRSZ.

– Low significant byte address in 10-bit mode addresses.

## 28.6.4 TWI Clock Waveform Generator Register

**Register Name:** TWI\_CWGR

**Access Type:** Read/Write

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

- **CLDIV: Clock Low Divider**

The SCL low period is defined as follows:

$$T_{low} = ((CLDIV \times 2^{CKDIV}) + 3) \times T_{MCK}$$

- **CHDIV: Clock High Divider**

The SCL high period is defined as follows:

$$T_{high} = ((CHDIV \times 2^{CKDIV}) + 3) \times T_{MCK}$$

- **CKDIV: Clock Divider**

The CKDIV is used to increase both SCL high and low periods.



## 28.6.5 TWI Status Register

**Register Name:** TWI\_SR

**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
–	–	–	–	–	–	–	NACK
7	6	5	4	3	2	1	0
–	–	–	–	–	TXRDY	RXRDY	TXCOMP

- **TXCOMP: Transmission Completed**

0 = In master, during the length of the current frame. In slave, from START received to STOP received.

1 = When both holding and shift registers are empty and STOP condition has been sent (in Master) or when MSEN is set (enable TWI).

- **RXRDY: Receive Holding Register Ready**

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.

- **TXRDY: Transmit Holding Register Ready**

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 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).

- **NACK: Not Acknowledged**

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. Reset after read.

## 28.6.6 TWI Interrupt Enable Register

Register Name: TWI\_IER

Access Type: Write-only

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

- **TXCOMP: Transmission Completed**
- **RXRDY: Receive Holding Register Ready**
- **TXRDY: Transmit Holding Register Ready**
- **NACK: Not Acknowledge**

0 = No effect.

1 = Enables the corresponding interrupt.

### 28.6.7 TWI Interrupt Disable Register

**Register Name:** TWI\_IDR

**Access Type:** Write-only

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

- **TXCOMP:** Transmission Completed
- **RXRDY:** Receive Holding Register Ready
- **TXRDY:** Transmit Holding Register Ready
- **NACK:** Not Acknowledge

0 = No effect.

1 = Disables the corresponding interrupt.

## 28.6.8 TWI Interrupt Mask Register

**Register Name:** TWI\_IMR

**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
–	–	–	–	–	–	–	NACK
7	6	5	4	3	2	1	0
–	–	–	–	–	TXRDY	RXRDY	TXCOMP

- **TXCOMP: Transmission Completed**
- **RXRDY: Receive Holding Register Ready**
- **TXRDY: Transmit Holding Register Ready**
- **NACK: Not Acknowledge**

0 = The corresponding interrupt is disabled.

1 = The corresponding interrupt is enabled.



### 28.6.9 TWI Receive Holding Register

Register Name: TWI\_RHR

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
RXDATA							

- RXDATA: Master or Slave Receive Holding Data

### 28.6.10 TWI Transmit Holding Register

Register Name: TWI\_THR

Access Type: Read/Write

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

- TXDATA: Master or Slave Transmit Holding Data





## **29. Universal Synchronous Asynchronous Receiver Transceiver (USART)**

### **29.0.1 Overview**

The Universal Synchronous Asynchronous Receiver Transceiver (USART) provides one full duplex universal synchronous asynchronous serial link. Data frame format is widely programmable (data length, parity, number of stop bits) to support a maximum of standards. The receiver implements parity error, framing error and overrun error detection. The receiver timeout enables handling variable-length frames and the transmitter timeguard facilitates communications with slow remote devices. Multidrop communications are also supported through address bit handling in reception and transmission.

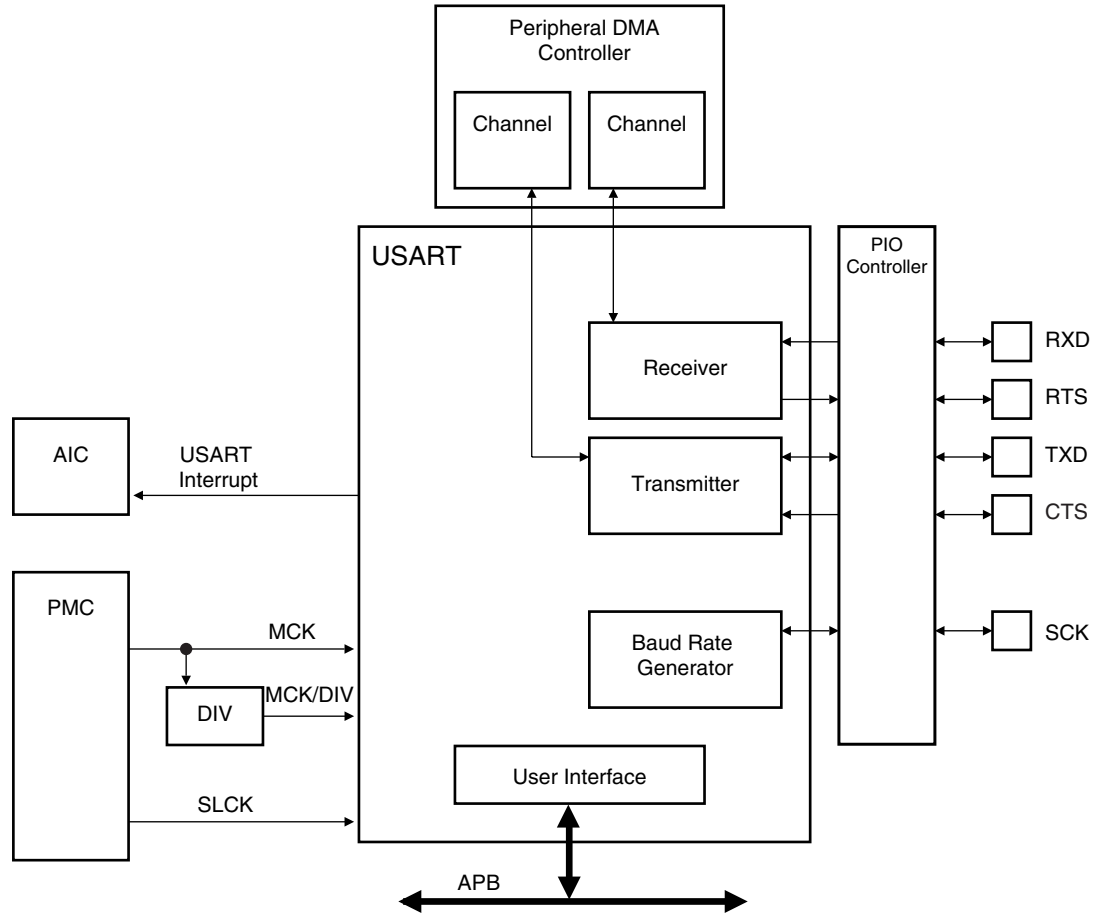
The USART features three test modes: remote loopback, local loopback and automatic echo.

The USART supports specific operating modes providing interfaces on RS485 buses, with ISO7816 T = 0 or T = 1 smart card slots and infrared transceivers. The hardware handshaking feature enables an out-of-band flow control by automatic management of the pins RTS and CTS.

The USART supports the connection to the Peripheral DMA Controller, which enables data transfers to the transmitter and from the receiver. The PDC provides chained buffer management without any intervention of the processor.

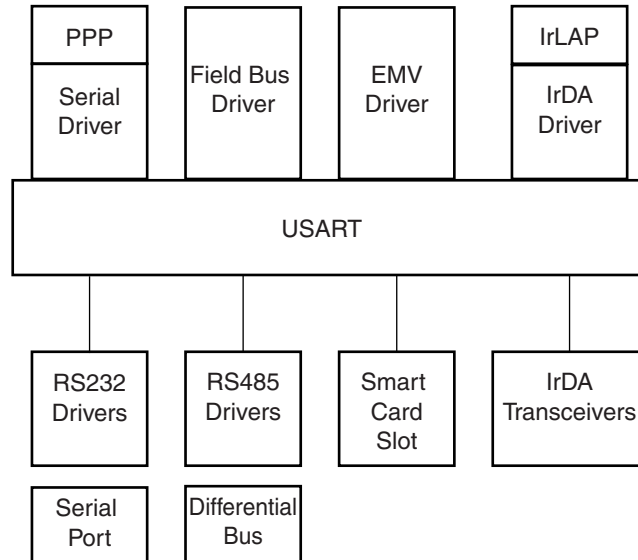
## 29.1 Block Diagram

Figure 29-1. USART Block Diagram



## 29.2 Application Block Diagram

Figure 29-2. Application Block Diagram



## 29.3 I/O Lines Description

Table 29-1. I/O Line Description

Name	Description	Type	Active Level
SCK	Serial Clock	I/O	
TXD	Transmit Serial Data	I/O	
RXD	Receive Serial Data	Input	
CTS	Clear to Send	Input	Low
RTS	Request to Send	Output	Low

## 29.4 Product Dependencies

### 29.4.1 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.

### 29.4.2 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.

### 29.4.3 Interrupt

The USART interrupt line is connected on one of the internal sources of the Advanced Interrupt Controller. Using the USART interrupt requires the AIC to be programmed first. Note that it is not recommended to use the USART interrupt line in edge sensitive mode.

## 29.5 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
  - NACK handling, error counter with repetition and iteration limit
- InfraRed IrDA Modulation and Demodulation
- Test modes
  - Remote loopback, local loopback, automatic echo

### 29.5.1 Baud Rate Generator

The Baud Rate Generator provides the bit period clock named the Baud Rate Clock to both the receiver and the transmitter.

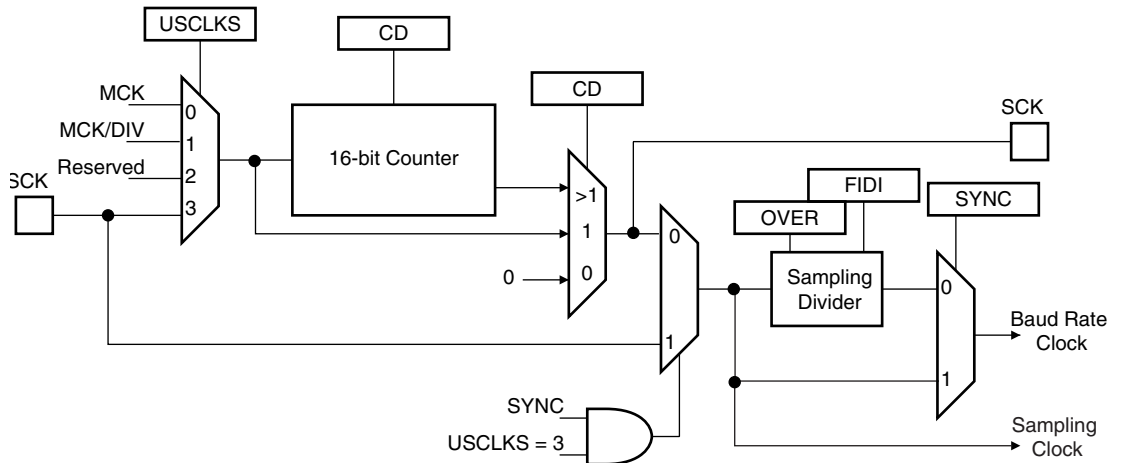
The Baud Rate Generator clock source can be selected by setting the USCLKS field in the Mode Register (US\_MR) between:

- 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 at 0, the Baud Rate Generator does not generate any clock. If CD is programmed at 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 4.5 times lower than MCK.

**Figure 29-3. Baud Rate Generator**



### 29.5.1.1 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 at 1.

### 29.5.1.2 Baud Rate Calculation Example

Table 29-2 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 29-2. 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%

**Table 29-2.** Baud Rate Example (OVER = 0) (Continued)

Source Clock	Expected Baud Rate	Calculation Result	CD	Actual Baud Rate	Error
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%.

$$Error = 1 - \left( \frac{ExpectedBaudRate}{ActualBaudRate} \right)$$

### 29.5.1.3 Baud Rate in Synchronous 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 4.5 times lower than the system clock.

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.

## 29.5.1.4 Baud Rate in ISO 7816 Mode

The ISO7816 specification defines the bit rate with the following formula:

$$B = \frac{D_i}{F_i} \times f$$

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 29-3](#).

**Table 29-3.** 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 29-4](#).

**Table 29-4.** 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 29-5](#) shows the resulting Fi/Di Ratio, which is the ratio between the ISO7816 clock and the baud rate clock.

**Table 29-5.** 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.

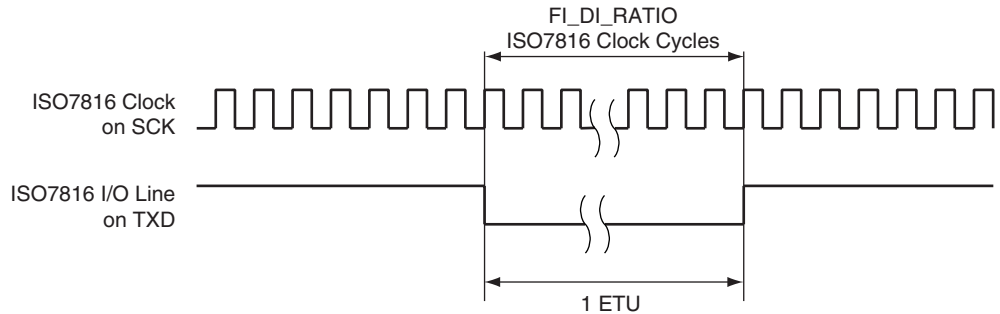
This clock is then divided by the value programmed in the FI\_DI\_RATIO field in the FI\_DI\_Ratio register (US\_FIDI). This is performed by the Sampling Divider, which performs a division by up to 2047 in ISO7816 Mode. The non-integer values of the Fi/Di Ratio are not supported and the user must program 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 ( $F_i = 372$ ,  $D_i = 1$ ).

Figure 29-4 shows the relation between the Elementary Time Unit, corresponding to a bit time, and the ISO 7816 clock.

**Figure 29-4.** Elementary Time Unit (ETU)



## 29.5.2 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 reset commands have the same effect as a hardware reset on the corresponding logic. 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.

## 29.5.3 Synchronous and Asynchronous Modes

### 29.5.3.1 Transmitter Operations

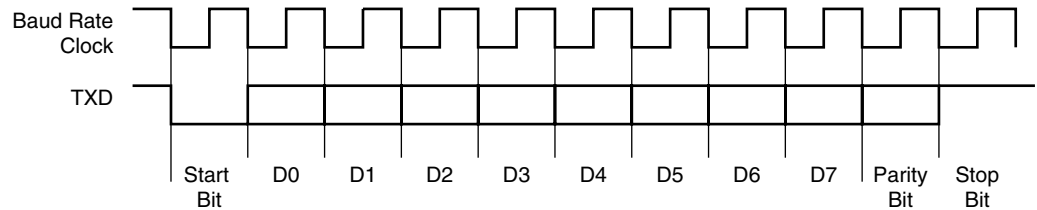
The transmitter performs the same in both synchronous and asynchronous operating modes ( $SYNC = 0$  or  $SYNC = 1$ ). One start bit, up to 9 data bits, one optional parity bit and up to two stop bits are successively shifted out on the TXD pin at each falling edge of the programmed serial clock.

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 at 1, the most significant bit is sent first. At 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 29-5.** Character Transmit

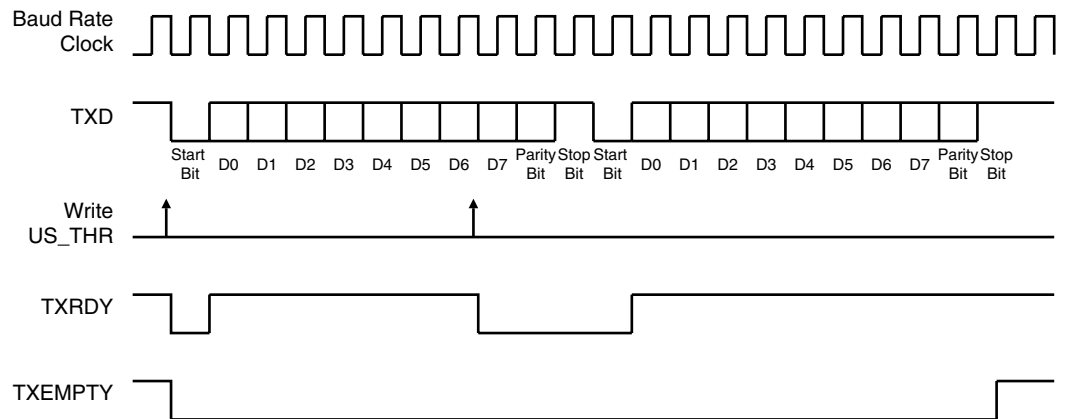
Example: 8-bit, Parity Enabled One Stop



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 raises.

Both TXRDY and TXEMPTY bits are low since the transmitter is disabled. Writing a character in US\_THR while TXRDY is active has no effect and the written character is lost.

**Figure 29-6.** Transmitter Status



### 29.5.3.2 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 at 0, a start bit is detected and data, parity and stop bits are successively sampled on the bit rate clock.

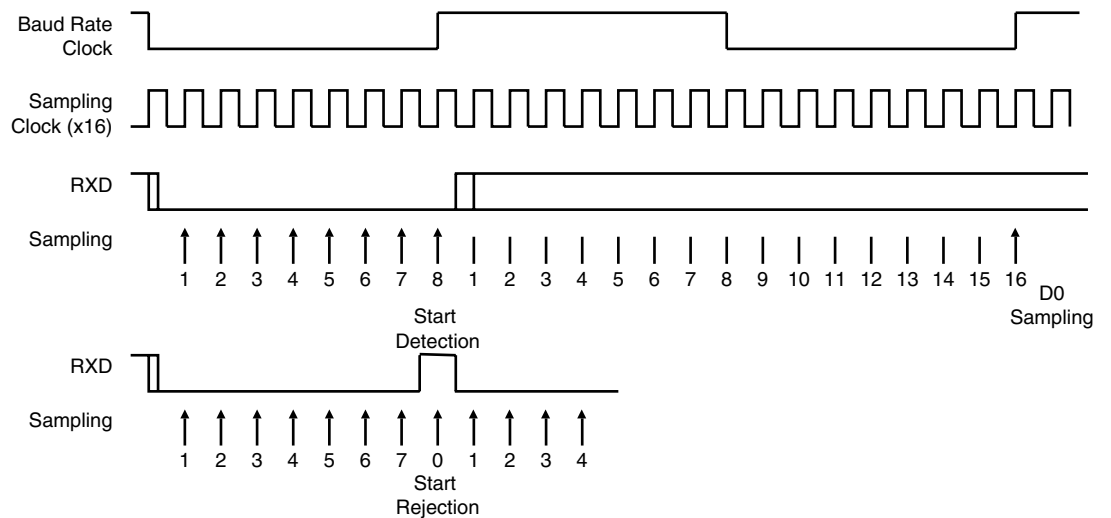
If the oversampling is 16, (OVER at 0), a start is detected at the eighth sample at 0. Then, data bits, parity bit and stop bit are sampled on each 16 sampling clock cycle. If the oversampling is

8 (OVER at 1), a start bit is detected at the fourth sample at 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.

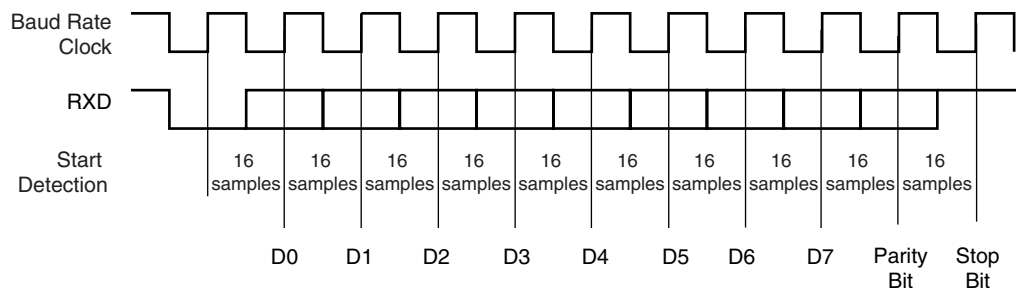
Figure 29-7 and Figure 29-8 illustrate start detection and character reception when USART operates in asynchronous mode.

**Figure 29-7.** Asynchronous Start Detection



**Figure 29-8.** Asynchronous Character Reception

Example: 8-bit, Parity Enabled



## 29.5.3.3 Synchronous Receiver

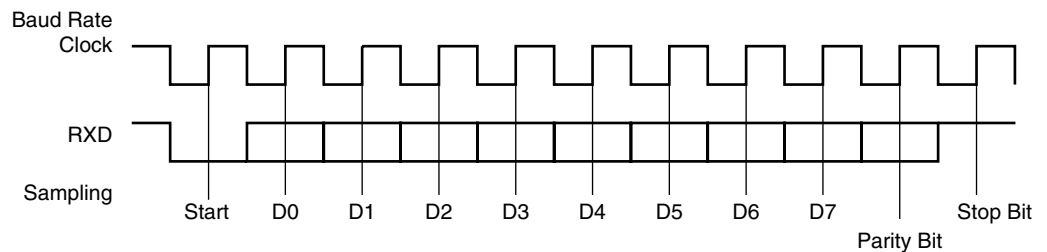
In synchronous mode ( $SYNC = 1$ ), the receiver samples the RXD signal on each rising edge of the Baud Rate Clock. If a low level is detected, it is considered as a start. All data bits, the parity bit and the stop bits are sampled and the receiver waits for the next start bit. Synchronous mode operations provide a high speed transfer capability.

Configuration fields and bits are the same as in asynchronous mode.

Figure 29-9 illustrates a character reception in synchronous mode.

**Figure 29-9.** Synchronous Mode Character Reception

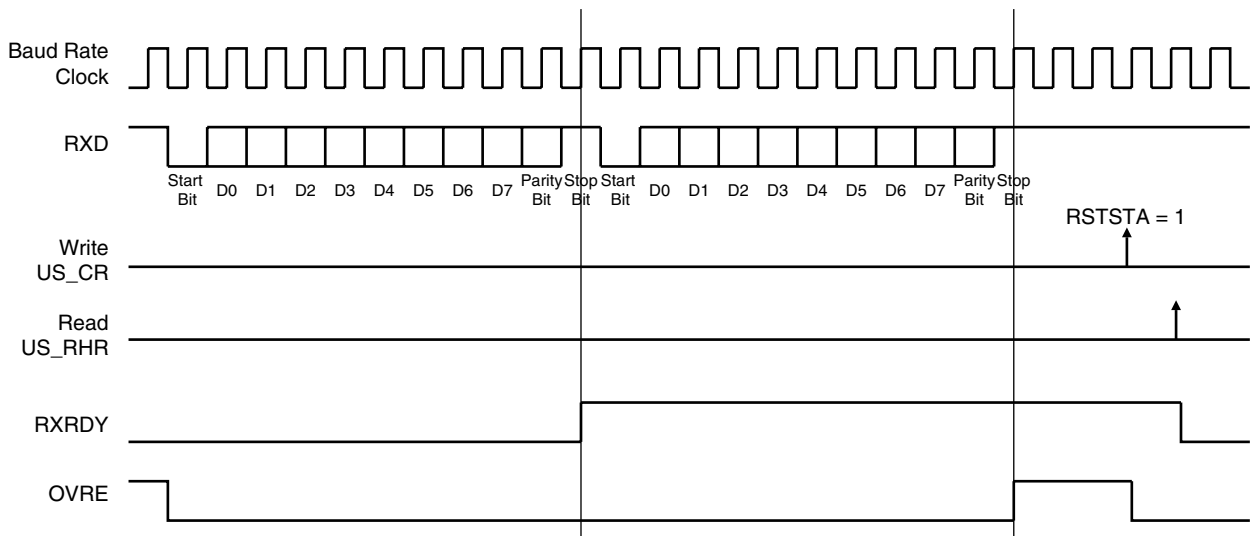
Example: 8-bit, Parity Enabled 1 Stop



## 29.5.3.4 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 at 1.

**Figure 29-10.** Receiver Status



## 29.5.3.5 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 286](#). 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 at 0 if a number of 1s in the character data bit is even, and at 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 at 1 if a number of 1s in the character data bit is even, and at 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 at 1 for all characters. The receiver parity checker reports an error if the parity bit is sampled at 0. If the space parity is used, the parity generator of the transmitter drives the parity bit at 0 for all characters. The receiver parity checker reports an error if the parity bit is sampled at 1. If parity is disabled, the transmitter does not generate any parity bit and the receiver does not report any parity error.

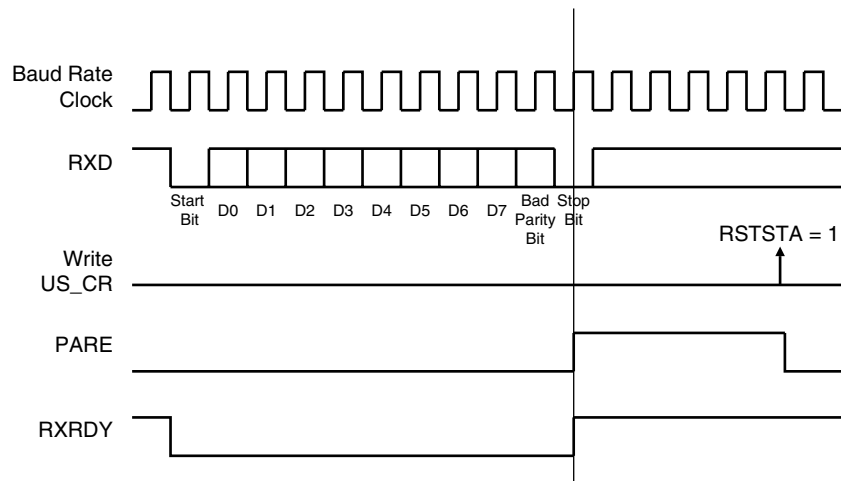
[Table 29-6](#) shows an example of the parity bit for the character 0x41 (character ASCII "A") depending on the configuration of the USART. Because there are two bits at 1, 1 bit is added when a parity is odd, or 0 is added when a parity is even.

**Table 29-6.** 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 at 1. [Figure 29-11](#) illustrates the parity bit status setting and clearing.

**Figure 29-11. Parity Error**



### 29.5.3.6 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 at 0 and addresses are transmitted with the parity bit at 1.

If the USART is configured in multidrop mode, the receiver sets the PARE parity error bit when the parity bit is high and the transmitter is able to send a character with the parity bit high when the Control Register is written with the SENDA bit at 1.

To handle parity error, the PARE bit is cleared when the Control Register is written with the bit RSTSTA at 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 at 0.

### 29.5.3.7 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 at 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 29-12](#), the behavior of TXRDY and TXEMPTY status bits is modified by the programming of a timeguard. TXRDY rises only when the start bit of the next character is sent, and thus remains at 0 during the timeguard transmission if a character has been written in US\_THR. TXEMPTY remains low until the timeguard transmission is completed as the timeguard is part of the current character being transmitted.

**Figure 29-12.** Timeguard Operations

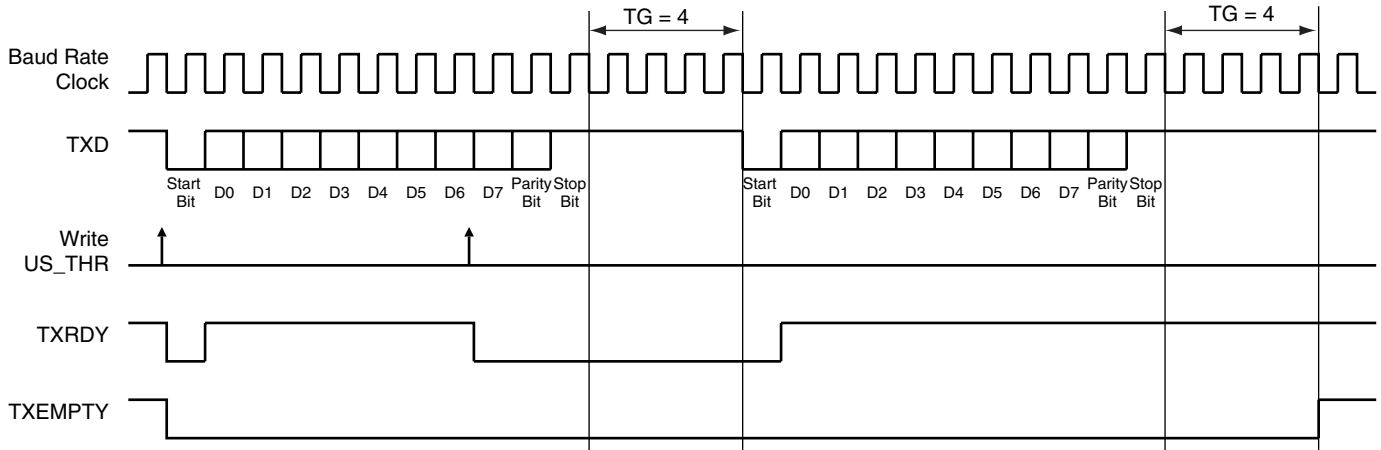


Table 29-7 indicates the maximum length of a timeguard period that the transmitter can handle in relation to the function of the Baud Rate.

**Table 29-7.** Maximum Timeguard Length Depending on Baud Rate

Baud Rate	Bit time	Timeguard
Bit/sec	$\mu$ s	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

### 29.5.3.8 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 at 0, the Receiver Time-out is disabled and no time-out is detected. The TIMEOUT bit in US\_CSR remains at 0. Otherwise, the receiver loads a 16-bit counter with the value programmed in TO. This counter is decremented at each bit period and reloaded each time a new character is received. If the counter reaches 0, the TIMEOUT bit in the Status Register rises.

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 at 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 at 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 29-13 shows the block diagram of the Receiver Time-out feature.

**Figure 29-13.** Receiver Time-out Block Diagram

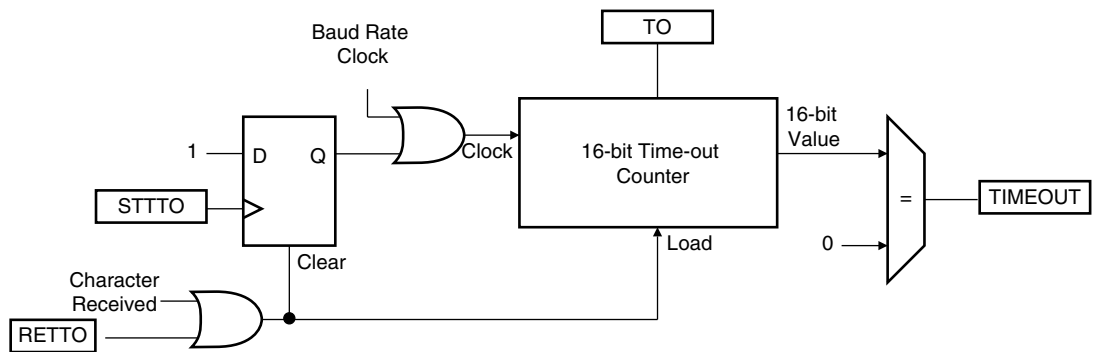


Table 29-8 gives the maximum time-out period for some standard baud rates.

**Table 29-8.** Maximum Time-out Period

Baud Rate	Bit Time	Time-out
bit/sec	$\mu$ s	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



**Table 29-8.** Maximum Time-out Period (Continued)

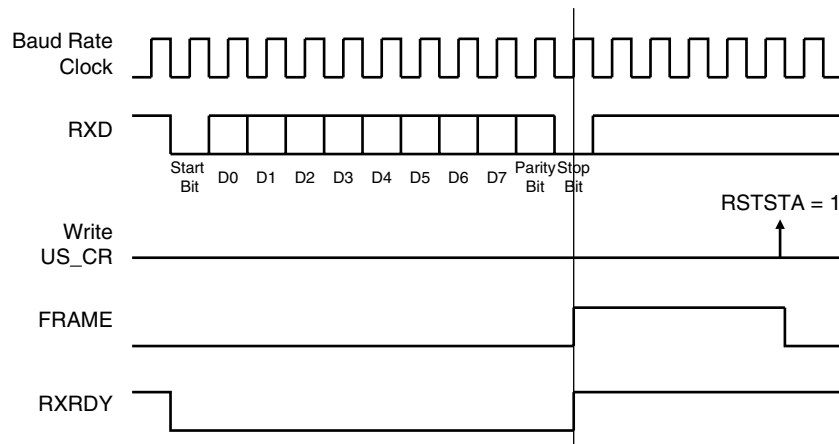
Baud Rate	Bit Time	Time-out
56000	18	1 170
57600	17	1 138
200000	5	328

### 29.5.3.9 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 at 1.

**Figure 29-14.** Framing Error Status



### 29.5.3.10 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 at 0. However, the transmitter holds the TXD line at least during one character until the user requests the break condition to be removed.

A break is transmitted by writing the Control Register (US\_CR) with the STTBK bit at 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 STTBK command is requested further STTBK commands are ignored until the end of the break is completed.

The break condition is removed by writing US\_CR with the STPBRK bit at 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 STTBKR and STPBKR commands are taken into account only if the TXRDY bit in US\_CSR is at 1 and the start of the break condition clears the TXRDY and TXEMPTY bits as if a character is processed.

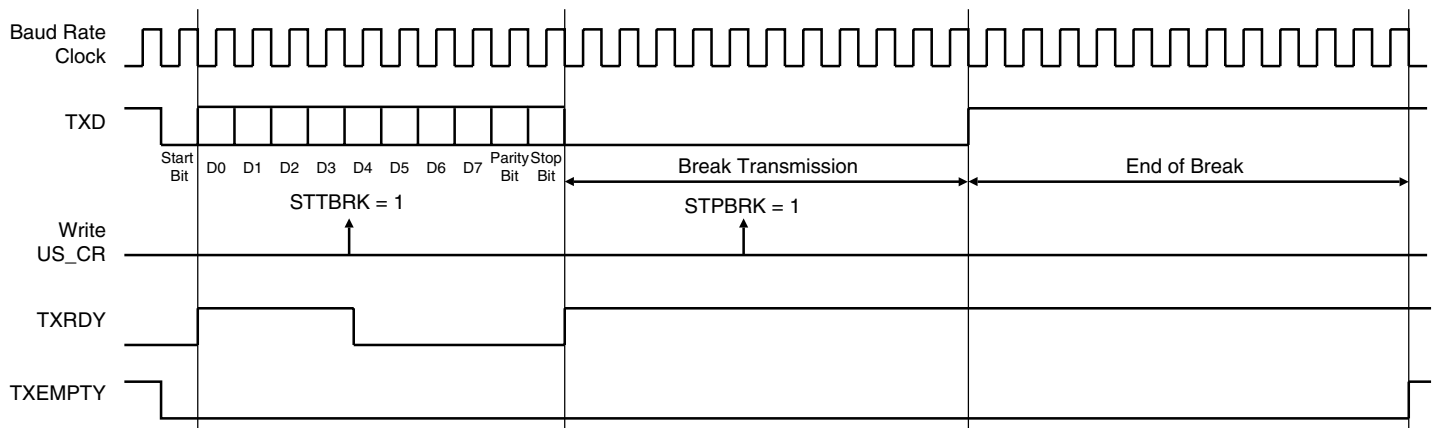
Writing US\_CR with the both STTBKR and STPBKR bits at 1 can lead to an unpredictable result. All STPBKR commands requested without a previous STTBKR command are ignored. A byte written into the Transmit Holding Register while a break is pending, but not started, is ignored.

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 29-15 illustrates the effect of both the Start Break (STTBKR) and Stop Break (STPBKR) commands on the TXD line.

**Figure 29-15.** Break Transmission



### 29.5.3.11 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 at 0x00, but FRAME remains low.

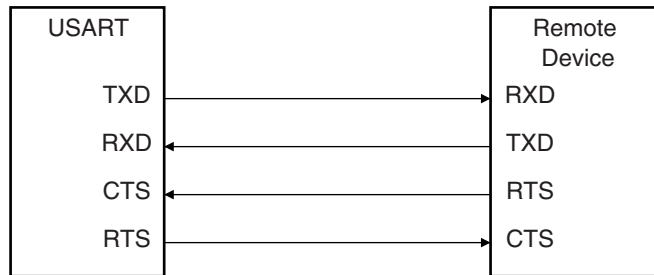
When the low stop bit is detected, the receiver asserts the RXBRK bit in US\_CSR. This bit may be cleared by writing the Control Register (US\_CR) with the bit RSTSTA at 1.

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.

### 29.5.3.12 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 29-16.

**Figure 29-16.** Connection with a Remote Device for Hardware Handshaking



Setting the USART to operate with hardware handshaking is performed by writing the USART\_MODE field in 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 29-17 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 29-17.** Receiver Behavior when Operating with Hardware Handshaking

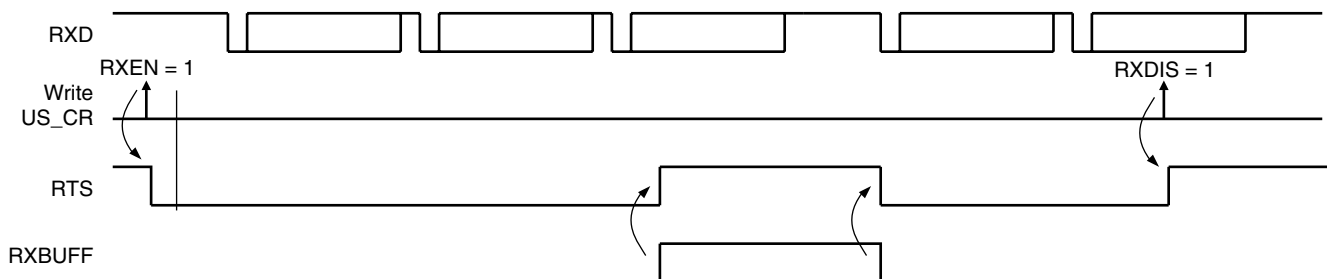


Figure 29-18 shows how the transmitter operates if hardware handshaking is enabled. The CTS pin disables the transmitter. If a character is being processing, the transmitter is disabled only after the completion of the current character and transmission of the next character happens as soon as the pin CTS falls.

**Figure 29-18.** Transmitter Behavior when Operating with Hardware Handshaking



## 29.5.4 ISO7816 Mode

The USART features an ISO7816-compatible operating mode. This mode permits interfacing with smart cards and Security Access Modules (SAM) communicating through an ISO7816 link. Both T = 0 and T = 1 protocols defined by the ISO7816 specification are supported.

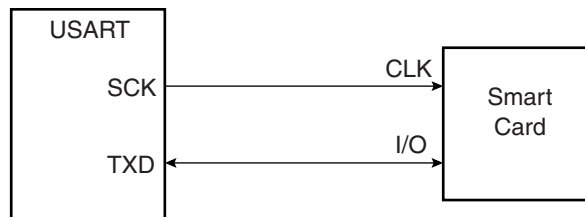
Setting the USART in ISO7816 mode is performed by writing the USART\_MODE field in the Mode Register (US\_MR) to the value 0x4 for protocol T = 0 and to the value 0x5 for protocol T = 1.

### 29.5.4.1 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 277](#)).

The USART connects to a smart card as shown in [Figure 29-19](#). 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 29-19.** 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 303](#) and ["PAR: Parity Type" on page 304](#).

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).

### 29.5.4.2 Protocol T = 0

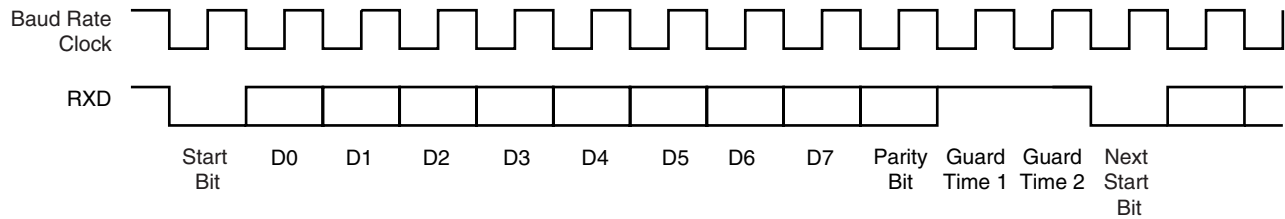
In T = 0 protocol, a character is made up of one start bit, eight data bits, one parity bit and one guard time, which lasts two bit times. The transmitter shifts out the bits and does not drive the I/O line during the guard time.

If no parity error is detected, the I/O line remains at 1 during the guard time and the transmitter can continue with the transmission of the next character, as shown in [Figure 29-20](#).

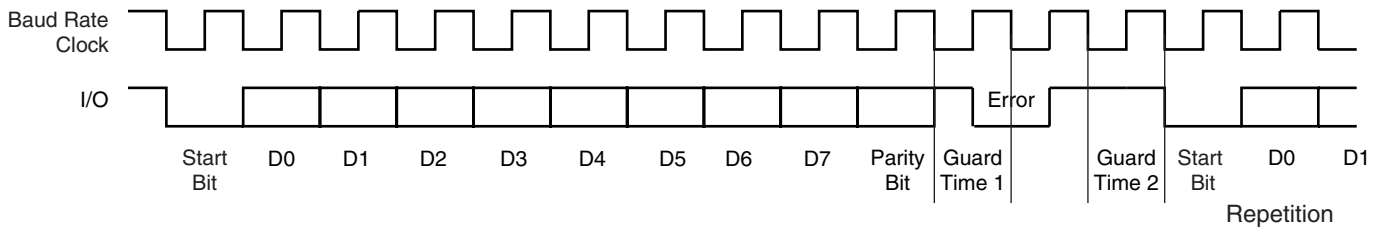
If a parity error is detected by the receiver, it drives the I/O line at 0 during the guard time, as shown in [Figure 29-21](#). This error bit is also named NACK, for Non Acknowledge. In this case, the character lasts 1 bit time more, as the guard time length is the same and is added to the error bit 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 29-20.** T = 0 Protocol without Parity Error



**Figure 29-21.** T = 0 Protocol with Parity Error



### 29.5.4.3 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.

### 29.5.4.4 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 at 1, no error signal is driven on the I/O line even if a parity bit is detected, but the INACK bit is set in the Status Register (US\_SR). The INACK bit can be cleared by writing the Control Register (US\_CR) with the RSTNACK bit at 1.

Moreover, if INACK is set, the erroneous received character is stored in the Receive Holding Register, as if no error occurred. However, the RXRDY bit does not raise.

### 29.5.4.5 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 at 1.

#### 29.5.4.6 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.

#### 29.5.4.7 Protocol T = 1

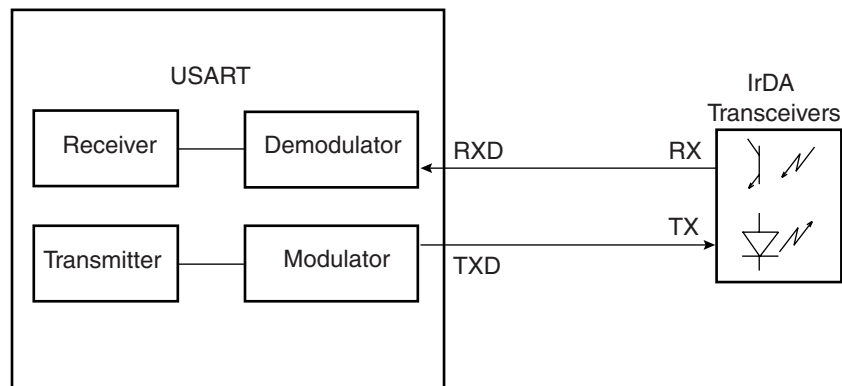
When operating in ISO7816 protocol T = 1, the transmission is similar to an asynchronous format with only one stop bit. The parity is generated when transmitting and checked when receiving. Parity error detection sets the PARE bit in the Channel Status Register (US\_CSR).

### 29.5.5 IrDA Mode

The USART features an IrDA mode supplying half-duplex point-to-point wireless communication. It embeds the modulator and demodulator which allows a glueless connection to the infrared transceivers, as shown in [Figure 29-22](#). 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 29-22.** Connection to IrDA Transceivers



The receiver and the transmitter must be enabled or disabled according to the direction of the transmission to be managed.

## 29.5.5.1 IrDA Modulation

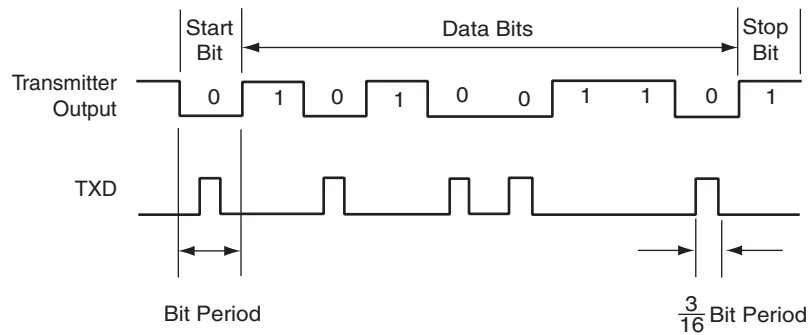
For baud rates up to and including 115.2 Kbits/sec, the RZI modulation scheme is used. “0” is represented by a light pulse of 3/16th of a bit time. Some examples of signal pulse duration are shown in [Table 29-9](#).

**Table 29-9.** IrDA Pulse Duration

Baud Rate	Pulse Duration (3/16)
2.4 Kb/s	78.13 $\mu$ s
9.6 Kb/s	19.53 $\mu$ s
19.2 Kb/s	9.77 $\mu$ s
38.4 Kb/s	4.88 $\mu$ s
57.6 Kb/s	3.26 $\mu$ s
115.2 Kb/s	1.63 $\mu$ s

[Figure 29-23](#) shows an example of character transmission.

**Figure 29-23.** IrDA Modulation



## 29.5.5.2 IrDA Baud Rate

[Table 29-10](#) gives some examples of CD values, baud rate error and pulse duration. Note that the requirement on the maximum acceptable error of  $\pm 1.87\%$  must be met.

**Table 29-10.** 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

**Table 29-10.** IrDA Baud Rate Error (Continued)

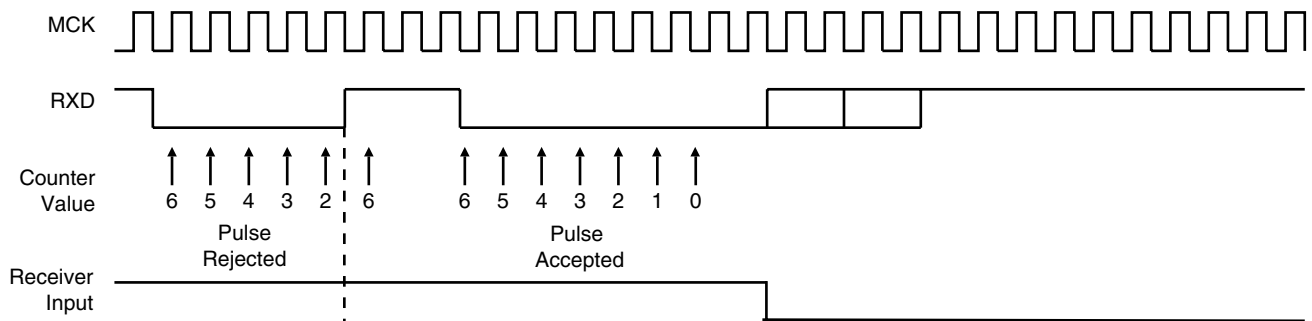
Peripheral Clock	Baud Rate	CD	Baud Rate Error	Pulse Time
32 768 000	38 400	53	0.63%	4.88
40 000 000	38 400	65	0.16%	4.88
3 686 400	19 200	12	0.00%	9.77
20 000 000	19 200	65	0.16%	9.77
32 768 000	19 200	107	0.31%	9.77
40 000 000	19 200	130	0.16%	9.77
3 686 400	9 600	24	0.00%	19.53
20 000 000	9 600	130	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

### 29.5.5.3 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 29-24 illustrates the operations of the IrDA demodulator.

**Figure 29-24.** IrDA Demodulator Operations



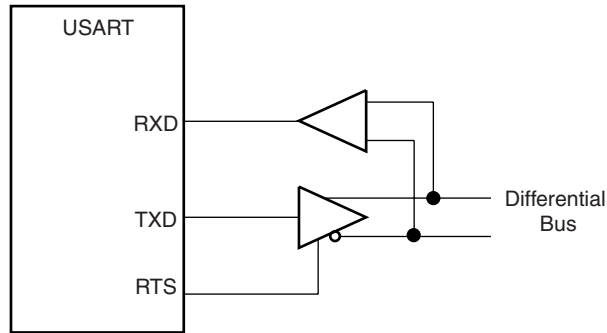
As the IrDA mode uses the same logic as the ISO7816, note that the FI\_DI\_RATIO field in US\_FIDI must be set to a value higher than 0 in order to assure IrDA communications operate correctly.



## 29.5.6 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 29-25](#).

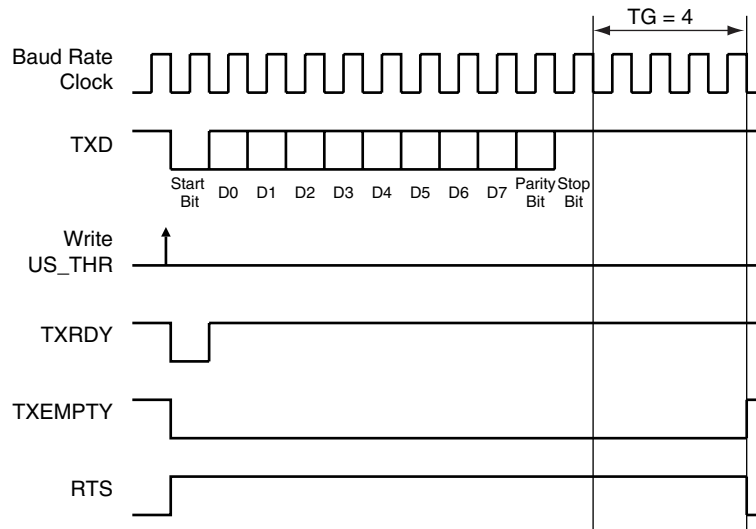
**Figure 29-25.** Typical Connection to a RS485 Bus



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 29-26](#) gives an example of the RTS waveform during a character transmission when the timeguard is enabled.

**Figure 29-26.** Example of RTS Drive with Timeguard



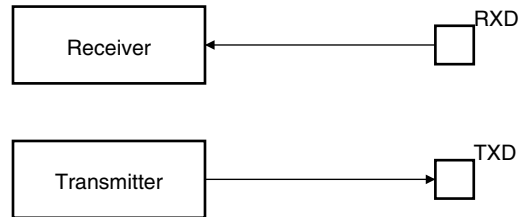
## 29.5.7 Test Modes

The USART can be programmed to operate in three different test modes. The internal loop-back capability allows on-board diagnostics. In the loopback mode the USART interface pins are disconnected or not and reconfigured for loopback internally or externally.

### 29.5.7.1 Normal Mode

Normal mode connects the RXD pin on the receiver input and the transmitter output on the TXD pin.

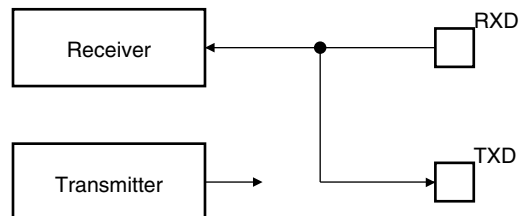
**Figure 29-27.** Normal Mode Configuration



### 29.5.7.2 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 29-28](#). 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.

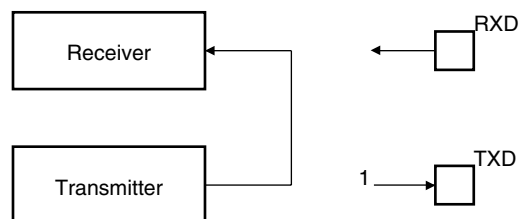
**Figure 29-28.** Automatic Echo Mode Configuration



### 29.5.7.3 Local Loopback Mode

Local loopback mode connects the output of the transmitter directly to the input of the receiver, as shown in [Figure 29-29](#). 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.

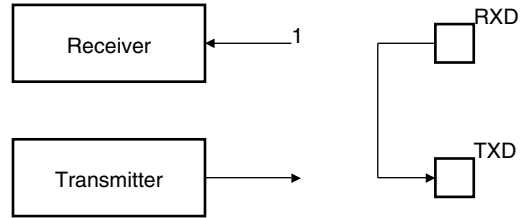
**Figure 29-29.** Local Loopback Mode Configuration



## 29.5.7.4 Remote Loopback Mode

Remote loopback mode directly connects the RXD pin to the TXD pin, as shown in [Figure 29-30](#). The transmitter and the receiver are disabled and have no effect. This mode allows bit-by-bit retransmission.

**Figure 29-30.** Remote Loopback Mode Configuration



## 29.6 USART User Interface

**Table 29-11.** USART Memory Map

Offset	Register	Name	Access	Reset State
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
0x5C - 0xFC	Reserved	–	–	–
0x100 - 0x128	Reserved for PDC Registers	–	–	–

## 29.6.1 USART Control Register

**Name:** US\_CR

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	RTSDIS	RTSEN	–	–
15	14	13	12	11	10	9	8
RETTO	RSTNACK	RSTIT	SENDA	STTTO	STPBRK	STTBRK	RSTSTA
7	6	5	4	3	2	1	0
TXDIS	TXEN	RXDIS	RXEN	RSTTX	RSTRX	–	–

- **RSTRX: Reset Receiver**

0: No effect.

1: Resets the receiver.

- **RSTTX: Reset Transmitter**

0: No effect.

1: Resets the transmitter.

- **RXEN: Receiver Enable**

0: No effect.

1: Enables the receiver, if RXDIS is 0.

- **RXDIS: Receiver Disable**

0: No effect.

1: Disables the receiver.

- **TXEN: Transmitter Enable**

0: No effect.

1: Enables the transmitter if TXDIS is 0.

- **TXDIS: Transmitter Disable**

0: No effect.

1: Disables the transmitter.

- **RSTSTA: Reset Status Bits**

0: No effect.

1: Resets the status bits PARE, FRAME, OVRE, and RXBRK in US\_CSR.

- **STTBRK: Start Break**

0: No effect.

1: Starts transmission of a break after the characters present in US\_THR and the Transmit Shift Register have been transmitted. No effect if a break is already being transmitted.

- **STPBRK: Stop Break**

0: No effect.

1: Stops transmission of the break after a minimum of one character length and transmits a high level during 12-bit periods. No effect if no break is being transmitted.

- **STTTO: Start Time-out**

0: No effect.

1: Starts waiting for a character before clocking the time-out counter. Resets the status bit TIMEOUT in US\_CSR.

- **SENDA: Send Address**

0: No effect.

1: In Multidrop Mode only, the next character written to the US\_THR is sent with the address bit set.

- **RSTIT: Reset Iterations**

0: No effect.

1: Resets ITERATION in US\_CSR. No effect if the ISO7816 is not enabled.

- **RSTNACK: Reset Non Acknowledge**

0: No effect

1: Resets NACK in US\_CSR.

- **RETTO: Rearm Time-out**

0: No effect

1: Restart Time-out

- **RTSEN: Request to Send Enable**

0: No effect.

1: Drives the pin RTS to 0.

- **RTSDIS: Request to Send Disable**

0: No effect.

1: Drives the pin RTS to 1.

## 29.6.2 USART Mode Register

**Name:** US\_MR  
**Access Type:** Read/Write

31	30	29	28	27	26	25	24
-	-	-	FILTER	-	MAX_ITERATION		
23	22	21	20	19	18	17	16
-	-	DSNACK	INACK	OVER	CLKO	MODE9	MSBF
15	14	13	12	11	10	9	8
CHMODE		NBSTOP		PAR			SYNC
7	6	5	4	3	2	1	0
CHRL		USCLKS		USART_MODE			

### • USART\_MODE

USART_MODE				Mode of the USART
0	0	0	0	Normal
0	0	0	1	RS485
0	0	1	0	Hardware Handshaking
0	0	1	1	Reserved
0	1	0	0	ISO7816 Protocol: T = 0
0	1	0	1	Reserved
0	1	1	0	ISO7816 Protocol: T = 1
0	1	1	1	Reserved
1	0	0	0	IrDA
1	1	x	x	Reserved

### • USCLKS: Clock Selection

USCLKS		Selected Clock
0	0	MCK
0	1	MCK / DIV
1	0	Reserved
1	1	SCK

- **CHRL: Character Length.**

CHRL		Character Length
0	0	5 bits
0	1	6 bits
1	0	7 bits
1	1	8 bits

- **SYNC: Synchronous Mode Select**

0: USART operates in Asynchronous Mode.

1: USART operates in Synchronous Mode.

- **PAR: Parity Type**

PAR			Parity Type
0	0	0	Even parity
0	0	1	Odd parity
0	1	0	Parity forced to 0 (Space)
0	1	1	Parity forced to 1 (Mark)
1	0	x	No parity
1	1	x	Multidrop mode

- **NBSTOP: Number of Stop Bits**

NBSTOP		Asynchronous (SYNC = 0)	Synchronous (SYNC = 1)
0	0	1 stop bit	1 stop bit
0	1	1.5 stop bits	Reserved
1	0	2 stop bits	2 stop bits
1	1	Reserved	Reserved

- **CHMODE: Channel Mode**

CHMODE		Mode Description
0	0	Normal Mode
0	1	Automatic Echo. Receiver input is connected to the TXD pin.
1	0	Local Loopback. Transmitter output is connected to the Receiver Input..
1	1	Remote Loopback. RXD pin is internally connected to the TXD pin.

- **MSBF: Bit Order**

0: Least Significant Bit is sent/received first.

1: Most Significant Bit is sent/received first.



- **MODE9: 9-bit Character Length**

0: CHRL defines character length.

1: 9-bit character length.

- **CLKO: Clock Output Select**

0: The USART does not drive the SCK pin.

1: The USART drives the SCK pin if USCLKS does not select the external clock SCK.

- **OVER: Oversampling Mode**

0: 16x Oversampling.

1: 8x Oversampling.

- **INACK: Inhibit Non Acknowledge**

0: The NACK is generated.

1: The NACK is not generated.

- **DSNACK: Disable Successive NACK**

0: NACK is sent on the ISO line as soon as a parity error occurs in the received character (unless INACK is set).

1: Successive parity errors are counted up to the value specified in the MAX\_ITERATION field. These parity errors generate a NACK on the ISO line. As soon as this value is reached, no additional NACK is sent on the ISO line. The flag ITERATION is asserted.

- **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).

## 29.6.3 USART Interrupt Enable Register

**Name:** US\_IER

**Access Type:** 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	ITERATION	TXEMPTY	TIMEOUT
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY

- **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
- **ITERATION:** Iteration Interrupt Enable
- **TXBUFE:** Buffer Empty Interrupt Enable
- **RXBUFF:** Buffer Full Interrupt Enable
- **NACK:** Non Acknowledge Interrupt Enable
- **CTSIC:** Clear to Send Input Change Interrupt Enable

## 29.6.4 USART Interrupt Disable Register

**Name:** US\_IDR

**Access Type:** 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	ITERATION	TXEMPTY	TIMEOUT
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY

- **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
- **ITERATION:** Iteration Interrupt Disable
- **TXBUFE:** Buffer Empty Interrupt Disable
- **RXBUFF:** Buffer Full Interrupt Disable
- **NACK:** Non Acknowledge Interrupt Disable
- **CTSIC:** Clear to Send Input Change Interrupt Disable

## 29.6.5 USART Interrupt Mask Register

**Name:** US\_IMR

**Access Type:** 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	ITERATION	TXEMPTY	TIMEOUT
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY

- **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**
- **ITERATION: Iteration Interrupt Mask**
- **TXBUFE: Buffer Empty Interrupt Mask**
- **RXBUFF: Buffer Full Interrupt Mask**
- **NACK: Non Acknowledge Interrupt Mask**
- **CTSIC: Clear to Send Input Change Interrupt Mask**

## 29.6.6 USART Channel Status Register

**Name:** US\_CSR

**Access Type:** 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	ITERATION	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 is at least one character in either US\_THR or the Transmit Shift Register.

- **ITERATION: Max number of Repetitions Reached**

0: Maximum number of repetitions has not been reached since the last RSIT.

1: Maximum number of repetitions has been reached since the last RSIT.

- **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 Acknowledge**

0: No 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 at 0.

1: CTS is at 1.

## 29.6.7 USART Receive Holding Register

**Name:** US\_RHR  
**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
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.

## 29.6.8 USART Transmit Holding Register

**Name:** US\_THR

**Access Type:** Write-only

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



## 29.6.9 USART Baud Rate Generator Register

**Name:** US\_BRGR

**Access Type:** Read/Write

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

- **CD: Clock Divider**

CD	USART_MODE ≠ ISO7816			USART_MODE = ISO7816
	SYNC = 0		SYNC = 1	
	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/CD/FI_DI_RATIO

## 29.6.10 USART Receiver Time-out Register

**Name:** US\_RTOR  
**Access Type:** Read/Write

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

- **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.

## 29.6.11 USART Transmitter Timeguard Register

**Name:** US\_TTGR  
**Access Type:** Read/Write

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

- **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.

## 29.6.12 USART FI DI RATIO Register

**Name:** US\_FIDI  
**Access Type:** 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							

- **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.

## 29.6.13 USART Number of Errors Register

**Name:** US\_NER  
**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
NB_ERRORS							

- **NB\_ERRORS: Number of Errors**

Total number of errors that occurred during an ISO7816 transfer. This register automatically clears when read.

## 29.6.14 USART IrDA FILTER Register

**Name:** US\_IF  
**Access Type:** Read/Write

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

- **IRDA\_FILTER: IrDA Filter**

Sets the filter of the IrDA demodulator.

## **30. Synchronous Serial Controller (SSC)**

### **30.1 Overview**

The Atmel Synchronous Serial Controller (SSC) provides a synchronous communication link with external devices. It supports many serial synchronous communication protocols generally used in audio and telecom applications such as I2S, Short Frame Sync, Long Frame Sync, etc.

The SSC contains an independent receiver and transmitter and a common clock divider. The receiver and the transmitter each interface with three signals: the TD/RD signal for data, the TK/RK signal for the clock and the TF/RF signal for the Frame Sync. The transfers can be programmed to start automatically or on different events detected on the Frame Sync signal.

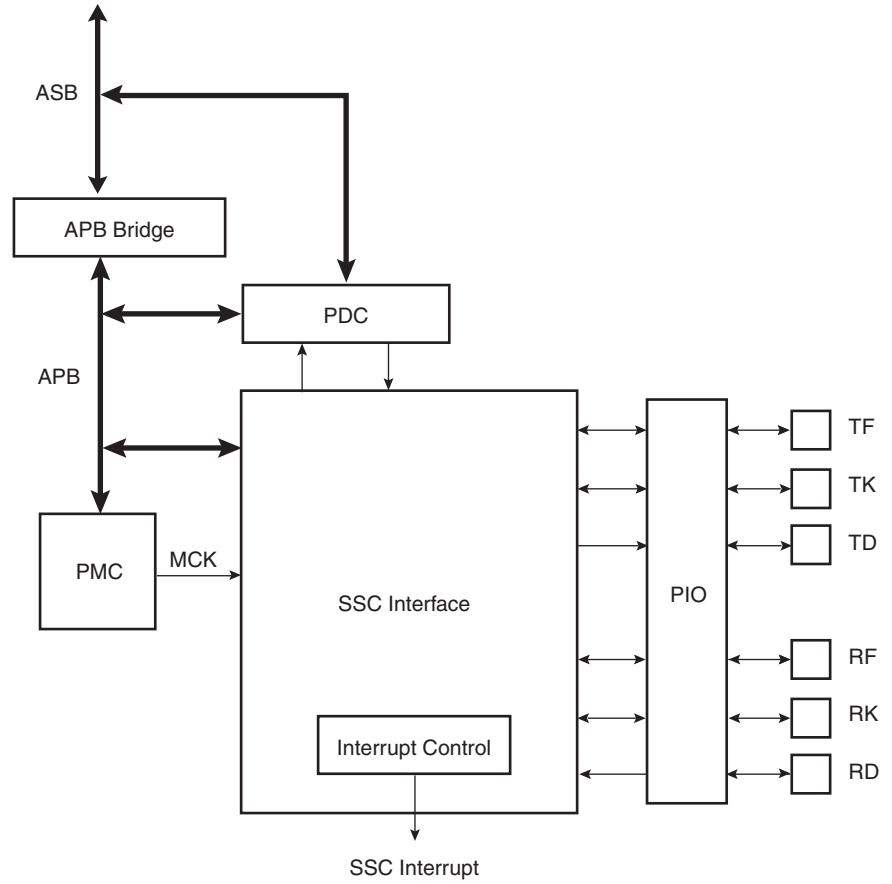
The SSC's high-level of programmability and its two dedicated PDC channels of up to 32 bits permit a continuous high bit rate data transfer without processor intervention.

Featuring connection to two PDC channels, the SSC permits interfacing with low processor overhead to the following:

- CODEC's in master or slave mode
- DAC through dedicated serial interface, particularly I2S
- Magnetic card reader

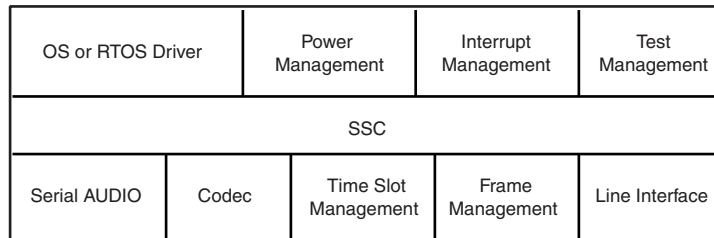
### 30.2 Block Diagram

Figure 30-1. Block Diagram



### 30.3 Application Block Diagram

Figure 30-2. Application Block Diagram



## 30.4 Pin Name List

**Table 30-1.** I/O Lines Description

Pin Name	Pin Description	Type
RF	Receiver Frame Synchro	Input/Output
RK	Receiver Clock	Input/Output
RD	Receiver Data	Input
TF	Transmitter Frame Synchro	Input/Output
TK	Transmitter Clock	Input/Output
TD	Transmitter Data	Output

## 30.5 Product Dependencies

### 30.5.1 I/O Lines

The pins used for interfacing the compliant external devices may be multiplexed with PIO lines.

Before using the SSC receiver, the PIO controller must be configured to dedicate the SSC receiver I/O lines to the SSC peripheral mode.

Before using the SSC transmitter, the PIO controller must be configured to dedicate the SSC transmitter I/O lines to the SSC peripheral mode.

### 30.5.2 Power Management

The SSC is not continuously clocked. The SSC interface may be clocked through the Power Management Controller (PMC), therefore the programmer must first configure the PMC to enable the SSC clock.

### 30.5.3 Interrupt

The SSC interface has an interrupt line connected to the Advanced Interrupt Controller (AIC). Handling interrupts requires programming the AIC before configuring the SSC.

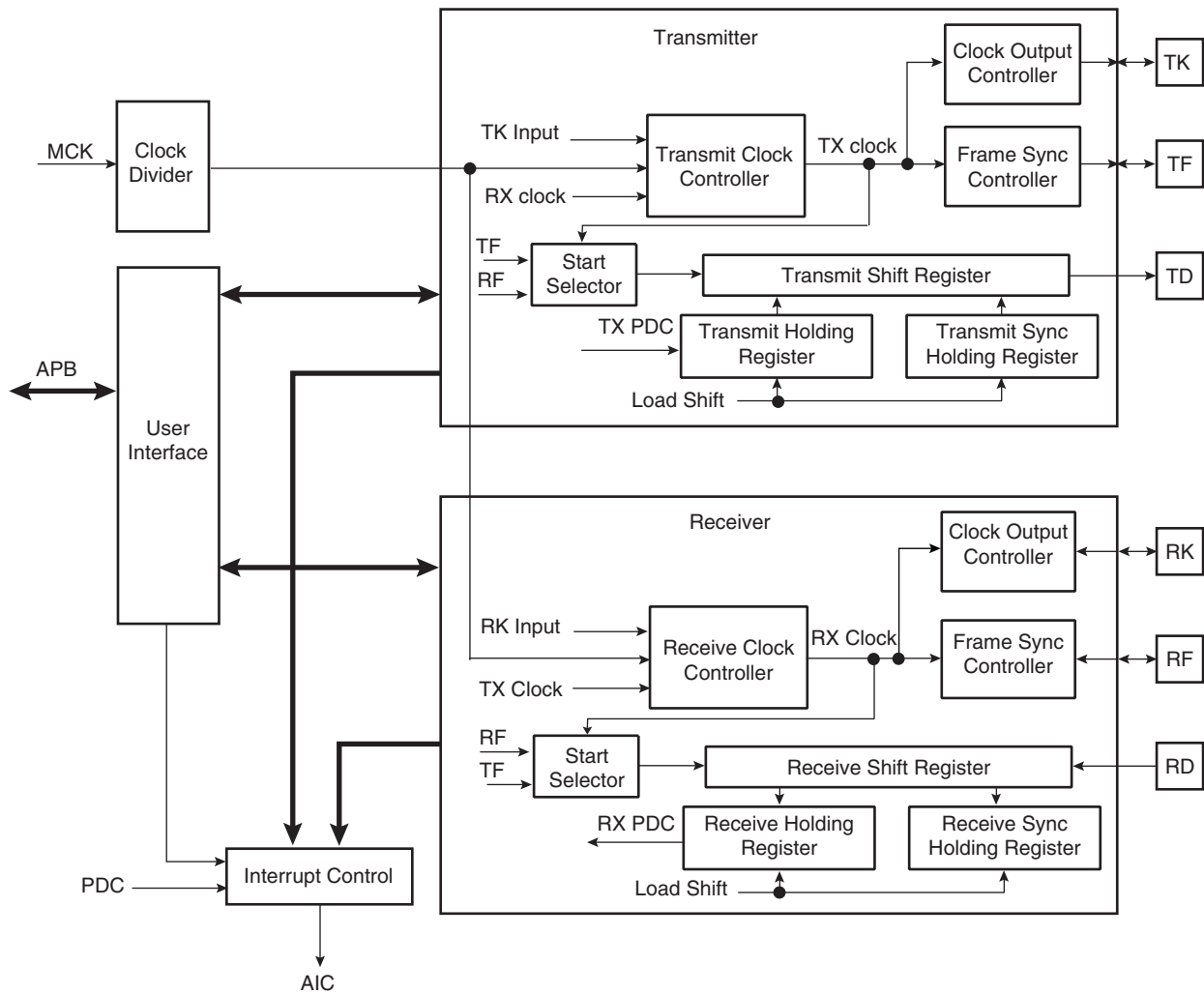
All SSC interrupts can be enabled/disabled configuring the SSC Interrupt mask register. Each pending and unmasked SSC interrupt will assert the SSC interrupt line. The SSC interrupt service routine can get the interrupt origin by reading the SSC interrupt status register.

## 30.6 Functional Description

This chapter contains the functional description of the following: SSC Functional Block, Clock Management, Data format, Start, Transmitter, Receiver and Frame Sync.

The receiver and transmitter operate separately. However, they can work synchronously by programming the receiver to use the transmit clock and/or to start a data transfer when transmission starts. Alternatively, this can be done by programming the transmitter to use the receive clock and/or to start a data transfer when reception starts. The transmitter and the receiver can be programmed to operate with the clock signals provided on either the TK or RK pins. This allows the SSC to support many slave-mode data transfers. The maximum clock speed allowed on the TK and RK pins is the master clock divided by 2.

**Figure 30-3. SSC Functional Block Diagram**



### 30.6.1 Clock Management

The transmitter clock can be generated by:

- an external clock received on the TK I/O pad
- the receiver clock
- the internal clock divider

The receiver clock can be generated by:

- an external clock received on the RK I/O pad
- the transmitter clock
- the internal clock divider

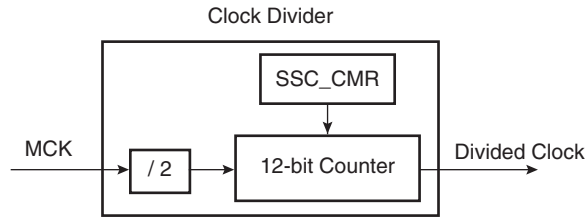
Furthermore, the transmitter block can generate an external clock on the TK I/O pad, and the receiver block can generate an external clock on the RK I/O pad.

This allows the SSC to support many Master and Slave Mode data transfers.



## 30.6.1.1 Clock Divider

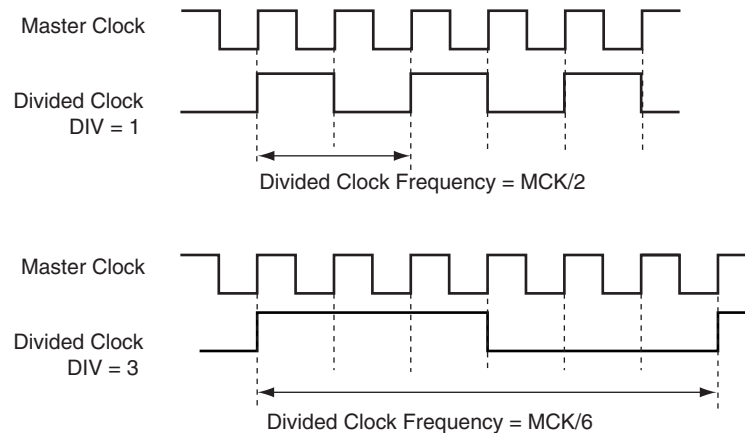
**Figure 30-4.** Divided Clock Block Diagram



The Master Clock divider is determined by the 12-bit field DIV counter and comparator (so its maximal value is 4095) in the Clock Mode Register SSC\_CMCR, allowing a Master Clock division by up to 8190. The Divided Clock is provided to both the Receiver and Transmitter. When this field is programmed to 0, the Clock Divider is not used and remains inactive.

When DIV is set to a value equal to or greater than 1, the Divided Clock has a frequency of Master Clock divided by 2 times DIV. Each level of the Divided Clock has a duration of the Master Clock multiplied by DIV. This ensures a 50% duty cycle for the Divided Clock regardless of whether the DIV value is even or odd.

**Figure 30-5.** Divided Clock Generation



Maximum	Minimum
MCK / 2	MCK / 8190

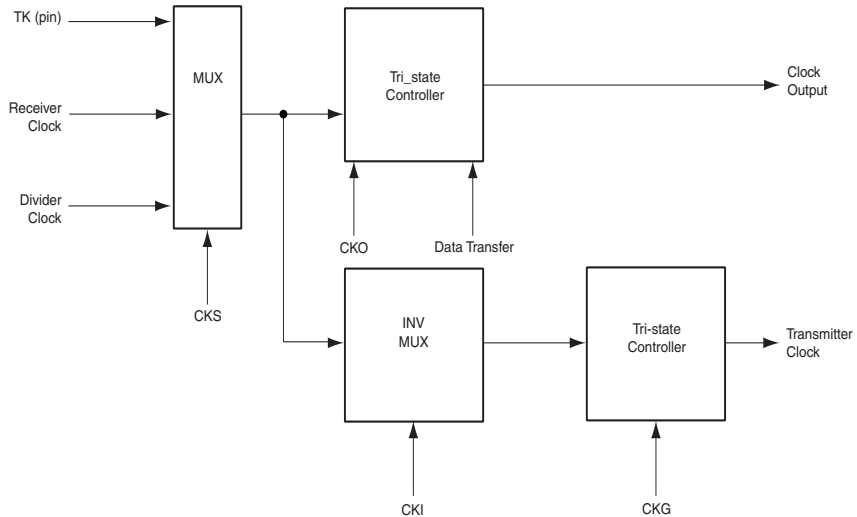
## 30.6.1.2 Transmitter Clock Management

The transmitter clock is generated from the receiver clock or the divider clock or an external clock scanned on the TK I/O pad. The transmitter clock is selected by the CKS field in SSC\_TCMR (Transmit Clock Mode Register). Transmit Clock can be inverted independently by the CKI bits in SSC\_TCMR.

The transmitter can also drive the TK I/O pad continuously or be limited to the actual data transfer. The clock output is configured by the SSC\_TCMR register. The Transmit Clock Inver-

sion (CKI) bits have no effect on the clock outputs. Programming the TCMR register to select TK pin (CKS field) and at the same time Continuous Transmit Clock (CKO field) might lead to unpredictable results.

**Figure 30-6.** Transmitter Clock Management

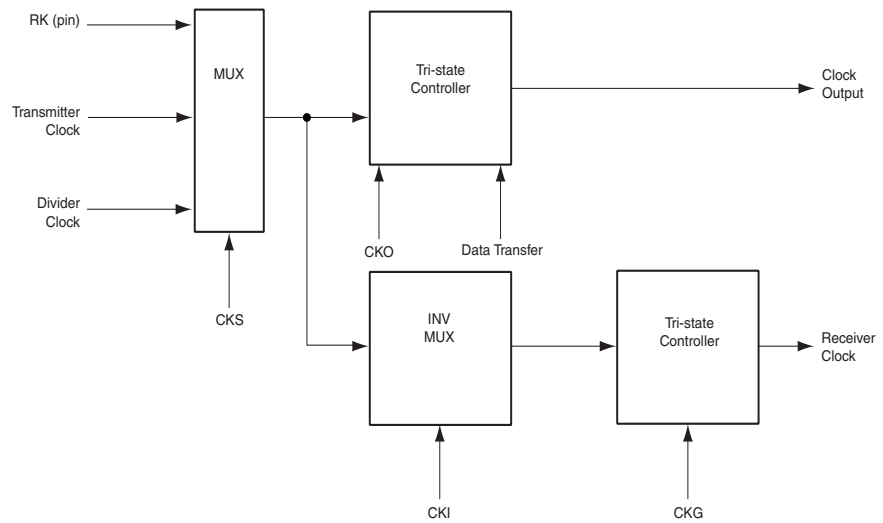


### 30.6.1.3 Receiver Clock Management

The receiver clock is generated from the transmitter clock or the divider clock or an external clock scanned on the RK I/O pad. The Receive Clock is selected by the CKS field in SSC\_RCMR (Receive Clock Mode Register). Receive Clocks can be inverted independently by the CKI bits in SSC\_RCMR.

The receiver can also drive the RK I/O pad continuously or be limited to the actual data transfer. The clock output is configured by the SSC\_RCMR register. The Receive Clock Inversion (CKI) bits have no effect on the clock outputs. Programming the RCMR register to select RK pin (CKS field) and at the same time Continuous Receive Clock (CKO field) can lead to unpredictable results.

**Figure 30-7. Receiver Clock Management**



### 30.6.1.4 Serial Clock Ratio Considerations

The Transmitter and the Receiver can be programmed to operate with the clock signals provided on either the TK or RK pins. This allows the SSC to support many slave-mode data transfers. In this case, the maximum clock speed allowed on the RK pin is:

- Master Clock divided by 2 if Receiver Frame Synchro is input
- Master Clock divided by 3 if Receiver Frame Synchro is output

In addition, the maximum clock speed allowed on the TK pin is:

- Master Clock divided by 6 if Transmit Frame Synchro is input
- Master Clock divided by 2 if Transmit Frame Synchro is output

### 30.6.2 Transmitter Operations

A transmitted frame is triggered by a start event and can be followed by synchronization data before data transmission.

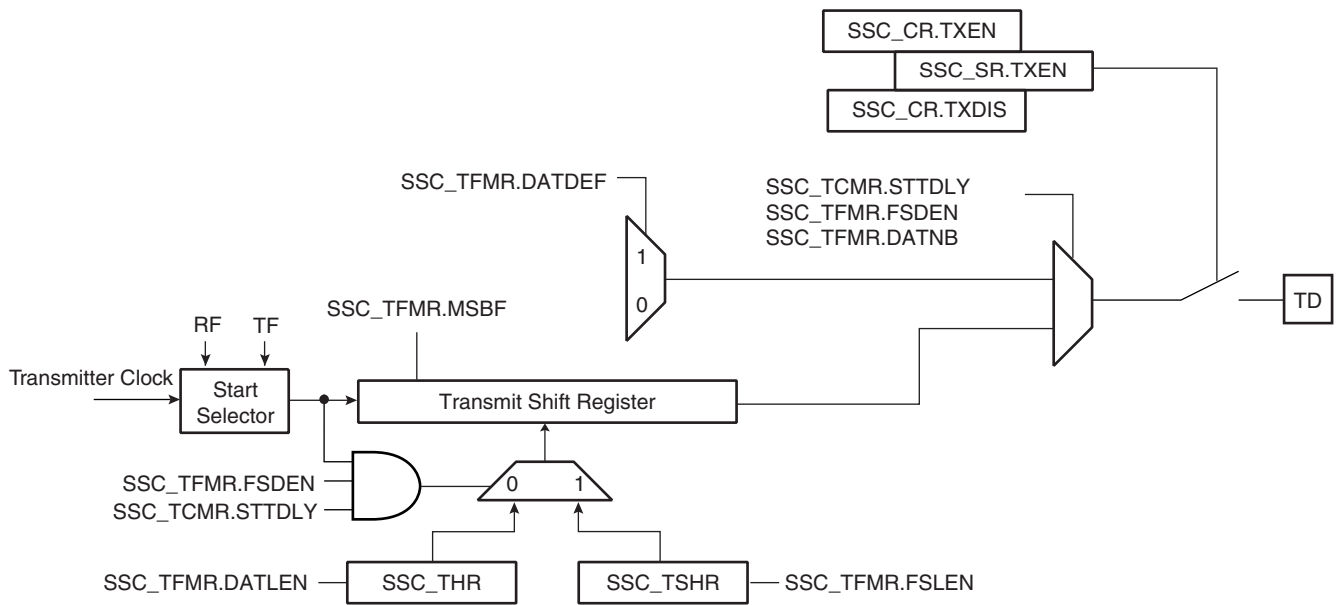
The start event is configured by setting the Transmit Clock Mode Register (SSC\_TCMR). See [“Start” on page 325](#).

The frame synchronization is configured setting the Transmit Frame Mode Register (SSC\_TFMR). See [“Frame Sync” on page 327](#).

To transmit data, the transmitter uses a shift register clocked by the transmitter clock signal and the start mode selected in the SSC\_TCMR. Data is written by the application to the SSC\_THR register then transferred to the shift register according to the data format selected.

When both the SSC\_THR and the transmit shift register are empty, the status flag TXEMPTY is set in SSC\_SR. When the Transmit Holding register is transferred in the Transmit shift register, the status flag TXRDY is set in SSC\_SR and additional data can be loaded in the holding register.

**Figure 30-8.** Transmitter Block Diagram



### 30.6.3 Receiver Operations

A received frame is triggered by a start event and can be followed by synchronization data before data transmission.

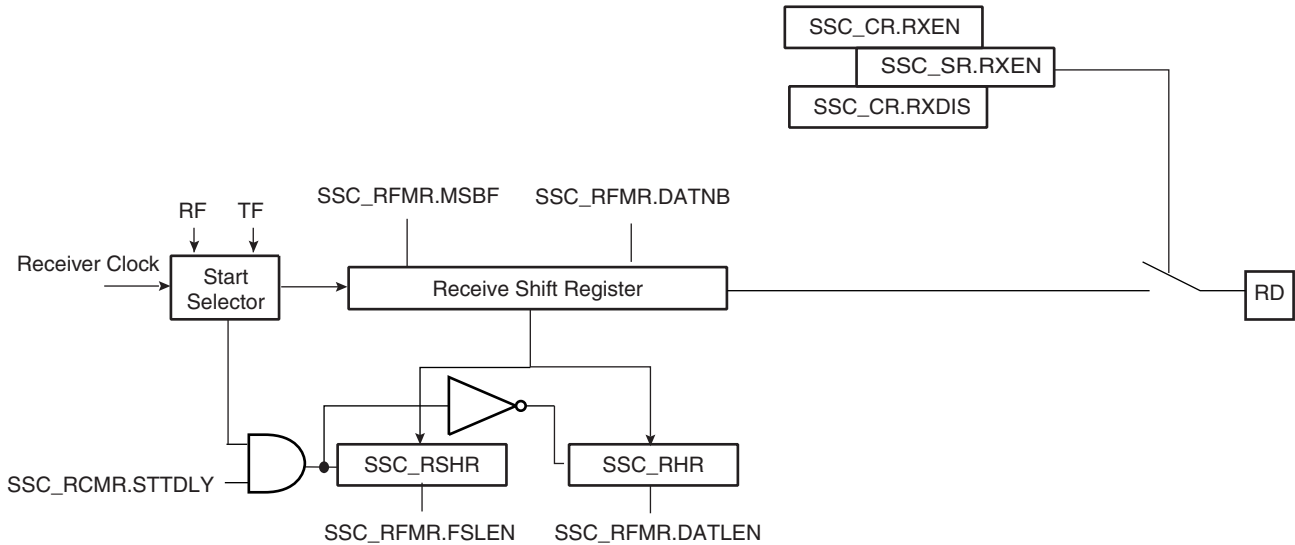
The start event is configured setting the Receive Clock Mode Register (SSC\_RCMR). See [“Start” on page 325](#).

The frame synchronization is configured setting the Receive Frame Mode Register (SSC\_RFMR). See [“Frame Sync” on page 327](#).

The receiver uses a shift register clocked by the receiver clock signal and the start mode selected in the SSC\_RCMR. The data is transferred from the shift register depending on the data format selected.

When the receiver shift register is full, the SSC transfers this data in the holding register, the status flag RXRDY is set in SSC\_SR and the data can be read in the receiver holding register. If another transfer occurs before read of the RHR register, the status flag OVERUN is set in SSC\_SR and the receiver shift register is transferred in the RHR register.

**Figure 30-9.** Receiver Block Diagram



### 30.6.4 Start

The transmitter and receiver can both be programmed to start their operations when an event occurs, respectively in the Transmit Start Selection (START) field of SSC\_TCMR and in the Receive Start Selection (START) field of SSC\_RCMR.

Under the following conditions the start event is independently programmable:

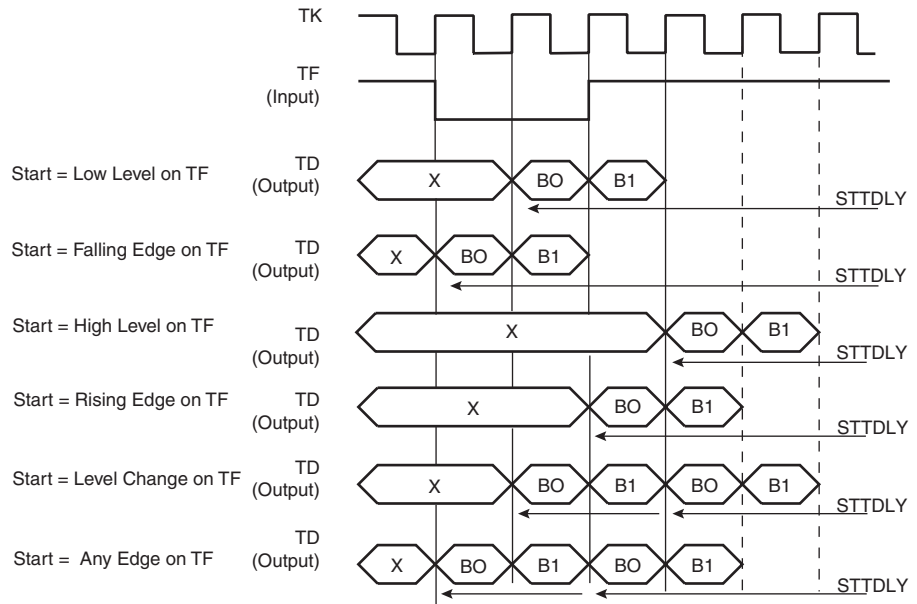
- Continuous. In this case, the transmission starts as soon as a word is written in SSC\_THR and the reception starts as soon as the Receiver is enabled.
- Synchronously with the transmitter/receiver
- On detection of a falling/rising edge on TF/RF
- On detection of a low level/high level on TF/RF
- On detection of a level change or an edge on TF/RF

A start can be programmed in the same manner on either side of the Transmit/Receive Clock Register (RCMR/TCMR). Thus, the start could be on TF (Transmit) or RF (Receive).

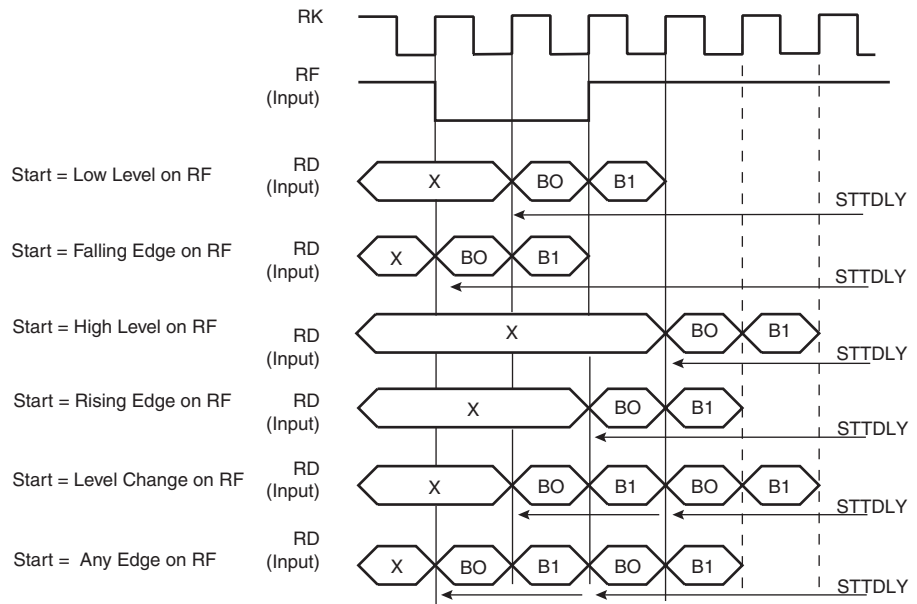
Moreover, the Receiver can start when data is detected in the bit stream with the Compare Functions.

Detection on TF/RF input/output is done by the field FSOS of the Transmit/Receive Frame Mode Register (TFMR/RFMR).

**Figure 30-10. Transmit Start Mode**



**Figure 30-11. Receive Pulse/Edge Start Modes**



## 30.6.5 Frame Sync

The Transmitter and Receiver Frame Sync pins, TF and RF, can be programmed to generate different kinds of frame synchronization signals. The Frame Sync Output Selection (FSOS) field in the Receive Frame Mode Register (SSC\_RFMR) and in the Transmit Frame Mode Register (SSC\_TFMR) are used to select the required waveform.

- Programmable low or high levels during data transfer are supported.
- Programmable high levels before the start of data transfers or toggling are also supported.

If a pulse waveform is selected, the Frame Sync Length (FSLEN) field in SSC\_RFMR and SSC\_TFMR programs the length of the pulse, from 1 bit time up to 16 bit time.

The periodicity of the Receive and Transmit Frame Sync pulse output can be programmed through the Period Divider Selection (PERIOD) field in SSC\_RCMR and SSC\_TCMR.

### 30.6.5.1 Frame Sync Data

Frame Sync Data transmits or receives a specific tag during the Frame Sync signal.

During the Frame Sync signal, the Receiver can sample the RD line and store the data in the Receive Sync Holding Register and the transmitter can transfer Transmit Sync Holding Register in the Shifter Register. The data length to be sampled/shifted out during the Frame Sync signal is programmed by the FSLEN field in SSC\_RFMR/SSC\_TFMR.

Concerning the Receive Frame Sync Data operation, if the Frame Sync Length is equal to or lower than the delay between the start event and the actual data reception, the data sampling operation is performed in the Receive Sync Holding Register through the Receive Shift Register.

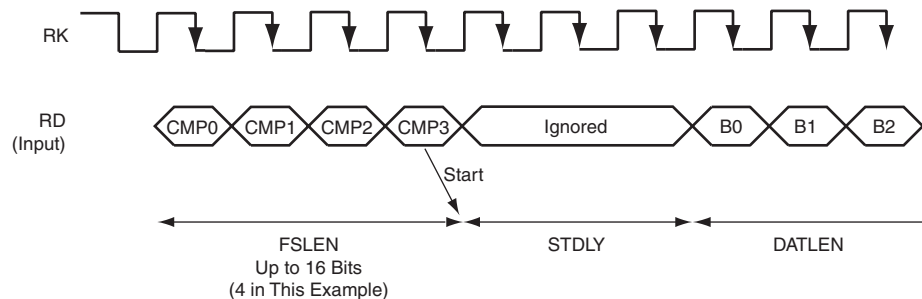
The Transmit Frame Sync Operation is performed by the transmitter only if the bit Frame Sync Data Enable (FSDEN) in SSC\_TFMR is set. If the Frame Sync length is equal to or lower than the delay between the start event and the actual data transmission, the normal transmission has priority and the data contained in the Transmit Sync Holding Register is transferred in the Transmit Register, then shifted out.

### 30.6.5.2 Frame Sync Edge Detection

The Frame Sync Edge detection is programmed by the FSEDGE field in SSC\_RFMR/SSC\_TFMR. This sets the corresponding flags RXSYN/TXSYN in the SSC Status Register (SSC\_SR) on frame synchro edge detection (signals RF/TF).

## 30.6.6 Receive Compare Modes

**Figure 30-12.** Receive Compare Modes



### 30.6.6.1 Compare Functions

Compare 0 can be one start event of the Receiver. In this case, the receiver compares at each new sample the last FSLEN bits received at the FSLEN lower bit of the data contained in the Compare 0 Register (SSC\_RC0R). When this start event is selected, the user can program the Receiver to start a new data transfer either by writing a new Compare 0, or by receiving continuously until Compare 1 occurs. This selection is done with the bit (STOP) in SSC\_RCMR.

### 30.6.7 Data Format

The data framing format of both the transmitter and the receiver are programmable through the Transmitter Frame Mode Register (SSC\_TFMR) and the Receiver Frame Mode Register (SSC\_RFMR). In either case, the user can independently select:

- the event that starts the data transfer (START)
- the delay in number of bit periods between the start event and the first data bit (STTDLY)
- the length of the data (DATLEN)
- the number of data to be transferred for each start event (DATNB).
- the length of synchronization transferred for each start event (FSLEN)
- the bit sense: most or lowest significant bit first (MSBF).

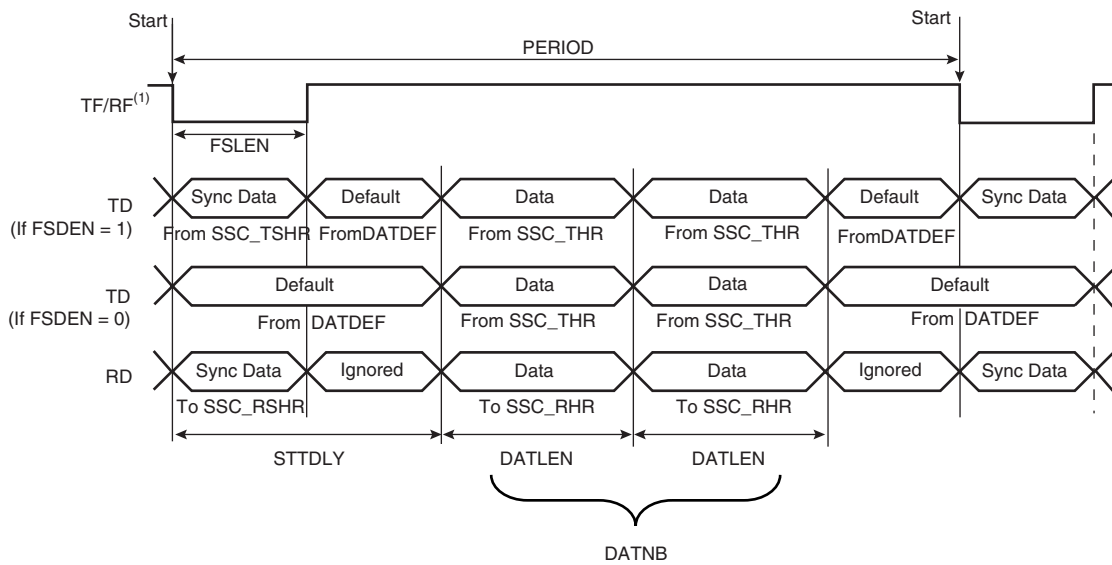
Additionally, the transmitter can be used to transfer synchronization and select the level driven on the TD pin while not in data transfer operation. This is done respectively by the Frame Sync Data Enable (FSDEN) and by the Data Default Value (DATDEF) bits in SSC\_TFMR.



**Table 30-2.** Data Frame Registers

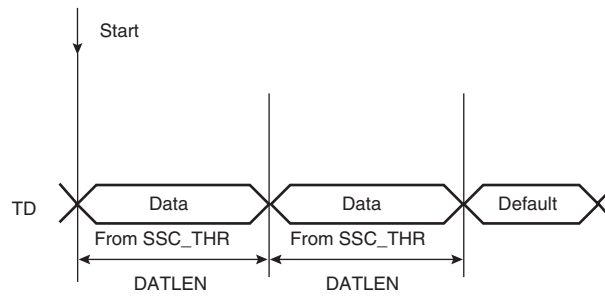
Transmitter	Receiver	Field	Length	Comment
SSC_TFMR	SSC_RFMR	DATLEN	Up to 32	Size of word
SSC_TFMR	SSC_RFMR	DATNB	Up to 16	Number of words transmitted in frame
SSC_TFMR	SSC_RFMR	MSBF		Most significant bit first
SSC_TFMR	SSC_RFMR	FSLEN	Up to 16	Size of Synchro data register
SSC_TFMR		DATDEF	0 or 1	Data default value ended
SSC_TFMR		FSDEN		Enable send SSC_TSHR
SSC_TCMR	SSC_RCMR	PERIOD	Up to 512	Frame size
SSC_TCMR	SSC_RCMR	STTDLY	Up to 255	Size of transmit start delay

**Figure 30-13.** Transmit and Receive Frame Format in Edge/Pulse Start Modes



Note: 1. Example of input on falling edge of TF/RF.

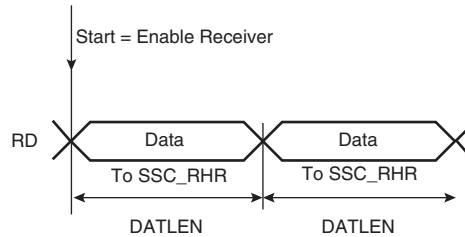
**Figure 30-14.** Transmit Frame Format in Continuous Mode



Start: 1. TXEMPTY set to 1  
2. Write into the SSC\_THR

Note: 1. STTDLY is set to 0. In this example, SSC\_THR is loaded twice. FSDEN value has no effect on the transmission. SyncData cannot be output in continuous mode.

**Figure 30-15.** Receive Frame Format in Continuous Mode



Note: 1. STTDLY is set to 0.

### 30.6.8 Loop Mode

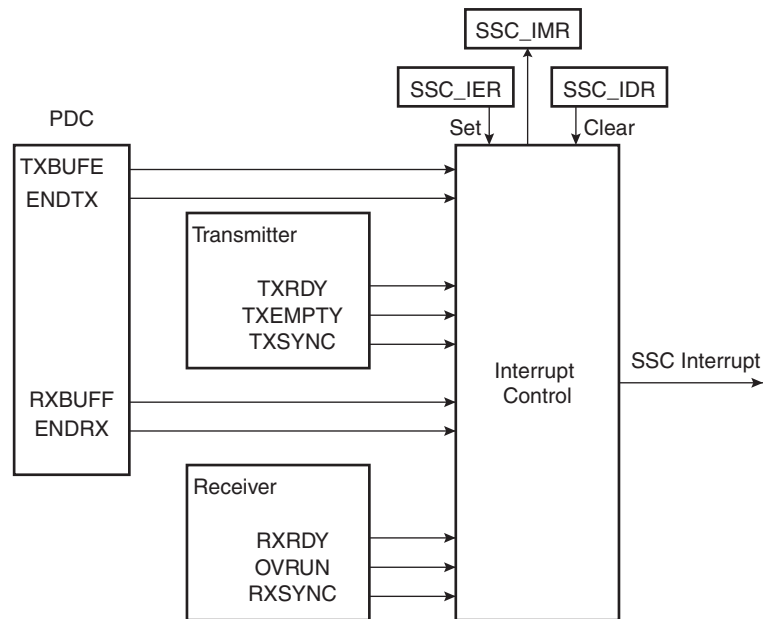
The receiver can be programmed to receive transmissions from the transmitter. This is done by setting the Loop Mode (LOOP) bit in SSC\_RFMR. In this case, RD is connected to TD, RF is connected to TF and RK is connected to TK.

### 30.6.9 Interrupt

Most bits in SSC\_SR have a corresponding bit in interrupt management registers.

The SSC can be programmed to generate an interrupt when it detects an event. The interrupt is controlled by writing SSC\_IER (Interrupt Enable Register) and SSC\_IDR (Interrupt Disable Register) These registers enable and disable, respectively, the corresponding interrupt by setting and clearing the corresponding bit in SSC\_IMR (Interrupt Mask Register), which controls the generation of interrupts by asserting the SSC interrupt line connected to the AIC.

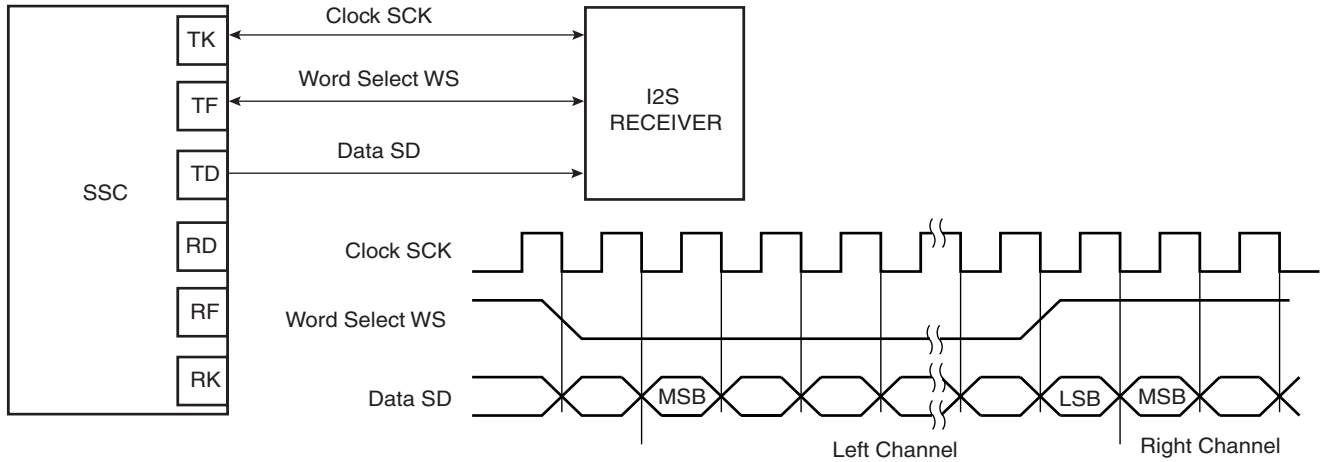
Figure 30-16. Interrupt Block Diagram



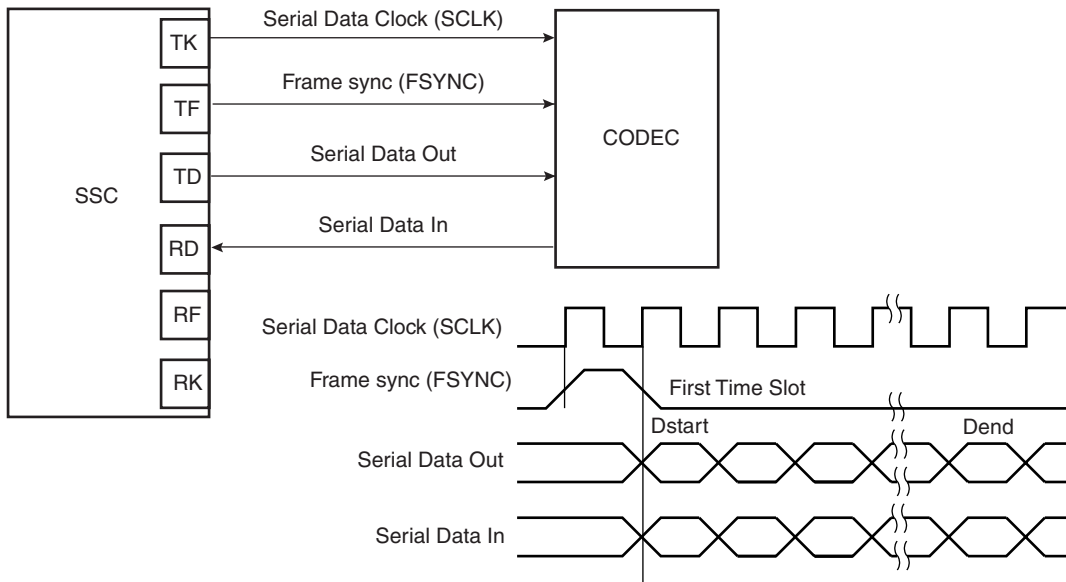
## 30.7 SSC Application Examples

The SSC can support several serial communication modes used in audio or high speed serial links. Some standard applications are shown in the following figures. All serial link applications supported by the SSC are not listed here.

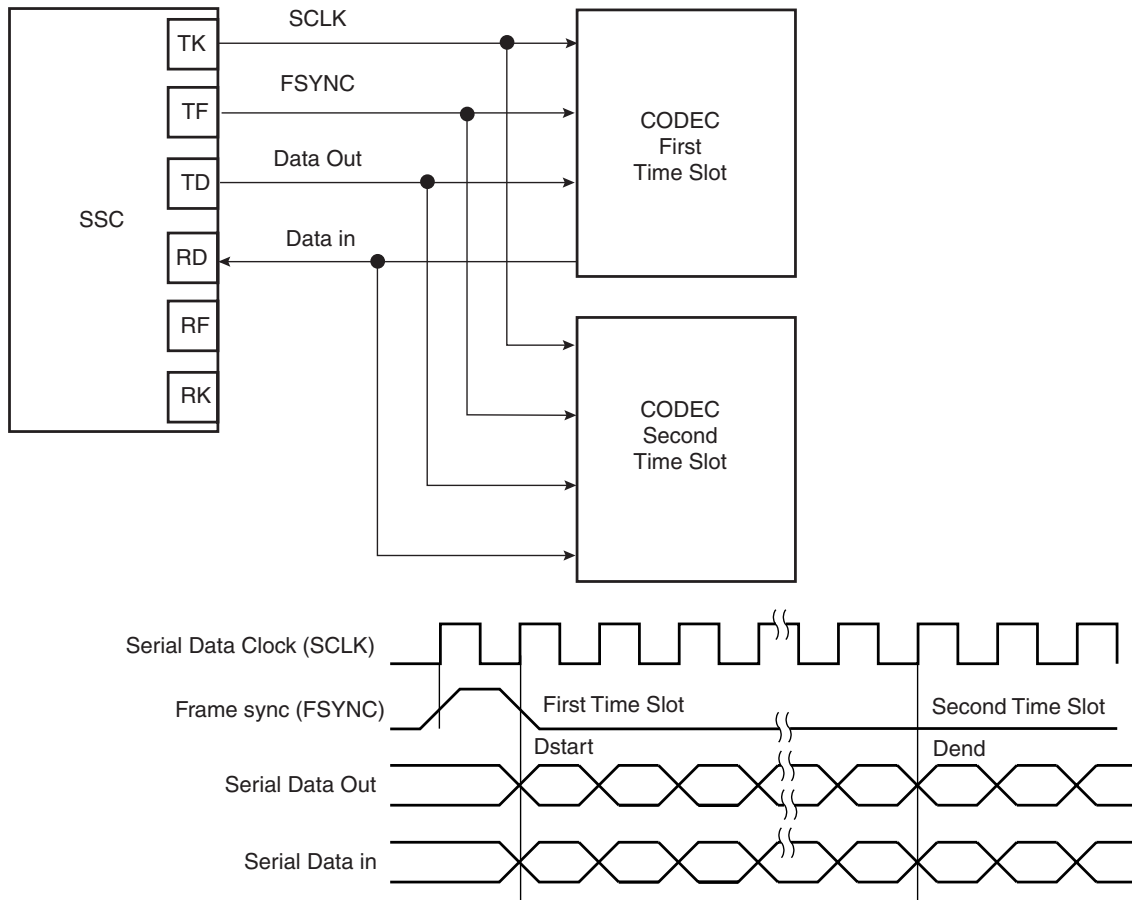
**Figure 30-17.** Audio Application Block Diagram



**Figure 30-18.** Codec Application Block Diagram



**Figure 30-19.** Time Slot Application Block Diagram



## 30.8 Synchronous Serial Controller (SSC) User Interface

**Table 30-3.** Register Mapping

Offset	Register	Register Name	Access	Reset
0x0	Control Register	SSC_CR	Write	–
0x4	Clock Mode Register	SSC_CMR	Read/Write	0x0
0x8	Reserved	–	–	–
0xC	Reserved	–	–	–
0x10	Receive Clock Mode Register	SSC_RCMR	Read/Write	0x0
0x14	Receive Frame Mode Register	SSC_RFMR	Read/Write	0x0
0x18	Transmit Clock Mode Register	SSC_TCMR	Read/Write	0x0
0x1C	Transmit Frame Mode Register	SSC_TFMR	Read/Write	0x0
0x20	Receive Holding Register	SSC_RHR	Read	0x0
0x24	Transmit Holding Register	SSC_THR	Write	–
0x28	Reserved	–	–	–
0x2C	Reserved	–	–	–
0x30	Receive Sync. Holding Register	SSC_RSHR	Read	0x0
0x34	Transmit Sync. Holding Register	SSC_TSHR	Read/Write	0x0
0x38	Receive Compare 0 Register	SSC_RC0R	Read/Write	0x0
0x3C	Receive Compare 1 Register	SSC_RC1R	Read/Write	0x0
0x40	Status Register	SSC_SR	Read	0x000000CC
0x44	Interrupt Enable Register	SSC_IER	Write	–
0x48	Interrupt Disable Register	SSC_IDR	Write	–
0x4C	Interrupt Mask Register	SSC_IMR	Read	0x0
0x50-0xFC	Reserved	–	–	–
0x100- 0x124	Reserved for Peripheral Data Controller (PDC)	–	–	–

## 30.8.1 SSC Control Register

**Name:** SSC\_CR  
**Access Type:** Write-only

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

- **RXEN: Receive Enable**

0: No effect.

1: Enables Receive if RXDIS is not set.

- **RXDIS: Receive Disable**

0: No effect.

1: Disables Receive. If a character is currently being received, disables at end of current character reception.

- **TXEN: Transmit Enable**

0: No effect.

1: Enables Transmit if TXDIS is not set.

- **TXDIS: Transmit Disable**

0: No effect.

1: Disables Transmit. If a character is currently being transmitted, disables at end of current character transmission.

- **SWRST: Software Reset**

0: No effect.

1: Performs a software reset. Has priority on any other bit in SSC\_CR.

## 30.8.2 SSC Clock Mode Register

**Name:** SSC\_CMCR

**Access Type:** Read/Write

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

- **DIV: Clock Divider**

0: The Clock Divider is not active.

Any Other Value: The Divided Clock equals the Master Clock divided by 2 times DIV. The maximum bit rate is  $MCK/2$ . The minimum bit rate is  $MCK/2 \times 4095 = MCK/8190$ .



## 30.8.3 SSC Receive Clock Mode Register

**Name:** SSC\_RCMR

**Access Type:** Read/Write

31	30	29	28	27	26	25	24	
PERIOD								
23	22	21	20	19	18	17	16	
STDDL								
15	14	13	12	11	10	9	8	
-	-	-	STOP	START				
7	6	5	4	3	2	1	0	
CKG		CKI	CKO			CKS		

### • CKS: Receive Clock Selection

CKS	Selected Receive Clock
0x0	Divided Clock
0x1	TK Clock signal
0x2	RK pin
0x3	Reserved

### • CKO: Receive Clock Output Mode Selection

CKO	Receive Clock Output Mode	RK pin
0x0	None	Input-only
0x1	Continuous Receive Clock	Output
0x2	Receive Clock only during data transfers	Output
0x3-0x7	Reserved	

### • CKI: Receive Clock Inversion

0: The data inputs (Data and Frame Sync signals) are sampled on Receive Clock falling edge. The Frame Sync signal output is shifted out on Receive Clock rising edge.

1: The data inputs (Data and Frame Sync signals) are sampled on Receive Clock rising edge. The Frame Sync signal output is shifted out on Receive Clock falling edge.

CKI affects only the Receive Clock and not the output clock signal.

- **CKG: Receive Clock Gating Selection**

CKG	Receive Clock Gating
0x0	None, continuous clock
0x1	Receive Clock enabled only if RF Low
0x2	Receive Clock enabled only if RF High
0x3	Reserved

- **START: Receive Start Selection**

START	Receive Start
0x0	Continuous, as soon as the receiver is enabled, and immediately after the end of transfer of the previous data.
0x1	Transmit start
0x2	Detection of a low level on RF signal
0x3	Detection of a high level on RF signal
0x4	Detection of a falling edge on RF signal
0x5	Detection of a rising edge on RF signal
0x6	Detection of any level change on RF signal
0x7	Detection of any edge on RF signal
0x8	Compare 0
0x9-0xF	Reserved

- **STOP: Receive Stop Selection**

0: After completion of a data transfer when starting with a Compare 0, the receiver stops the data transfer and waits for a new compare 0.

1: After starting a receive with a Compare 0, the receiver operates in a continuous mode until a Compare 1 is detected.

- **STTDLY: Receive Start Delay**

If STTDLY is not 0, a delay of STTDLY clock cycles is inserted between the start event and the actual start of reception. When the Receiver is programmed to start synchronously with the Transmitter, the delay is also applied.

Note: It is very important that STTDLY be set carefully. If STTDLY must be set, it should be done in relation to TAG (Receive Sync Data) reception.

- **PERIOD: Receive Period Divider Selection**

This field selects the divider to apply to the selected Receive Clock in order to generate a new Frame Sync Signal. If 0, no PERIOD signal is generated. If not 0, a PERIOD signal is generated each 2 x (PERIOD+1) Receive Clock.

## 30.8.4 SSC Receive Frame Mode Register

**Name:** SSC\_RFMR

**Access Type:** Read/Write

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

- **DATLEN: Data Length**

0: Forbidden value (1-bit data length not supported).

Any other value: The bit stream contains DATLEN + 1 data bits. Moreover, it defines the transfer size performed by the PDC2 assigned to the Receiver. If DATLEN is lower or equal to 7, data transfers are in bytes. If DATLEN is between 8 and 15 (included), half-words are transferred, and for any other value, 32-bit words are transferred.

- **LOOP: Loop Mode**

0: Normal operating mode.

1: RD is driven by TD, RF is driven by TF and TK drives RK.

- **MSBF: Most Significant Bit First**

0: The lowest significant bit of the data register is sampled first in the bit stream.

1: The most significant bit of the data register is sampled first in the bit stream.

- **DATNB: Data Number per Frame**

This field defines the number of data words to be received after each transfer start, which is equal to (DATNB + 1).

- **FSLEN: Receive Frame Sync Length**

This field defines the length of the Receive Frame Sync Signal and the number of bits sampled and stored in the Receive Sync Data Register. When this mode is selected by the START field in the Receive Clock Mode Register, it also determines the length of the sampled data to be compared to the Compare 0 or Compare 1 register.

Pulse length is equal to (FSLEN + 1) Receive Clock periods. Thus, if FSLEN is 0, the Receive Frame Sync signal is generated during one Receive Clock period.

- **FSOS: Receive Frame Sync Output Selection**

FSOS	Selected Receive Frame Sync Signal	RF Pin
0x0	None	Input-only
0x1	Negative Pulse	Output
0x2	Positive Pulse	Output
0x3	Driven Low during data transfer	Output
0x4	Driven High during data transfer	Output
0x5	Toggling at each start of data transfer	Output
0x6-0x7	Reserved	Undefined

- **FSEDGE: Frame Sync Edge Detection**

Determines which edge on Frame Sync will generate the interrupt RXSYN in the SSC Status Register.

FSEDGE	Frame Sync Edge Detection
0x0	Positive Edge Detection
0x1	Negative Edge Detection

## 30.8.5 SSC Transmit Clock Mode Register

Name: SSC\_TCMR

Access Type: Read/Write

31	30	29	28	27	26	25	24
PERIOD							
23	22	21	20	19	18	17	16
STTDLY							
15	14	13	12	11	10	9	8
-	-	-	-	START			
7	6	5	4	3	2	1	0
CKG		CKI	CKO			CKS	

### • CKS: Transmit Clock Selection

CKS	Selected Transmit Clock
0x0	Divided Clock
0x1	RK Clock signal
0x2	TK Pin
0x3	Reserved

### • CKO: Transmit Clock Output Mode Selection

CKO	Transmit Clock Output Mode	TK pin
0x0	None	Input-only
0x1	Continuous Transmit Clock	Output
0x2	Transmit Clock only during data transfers	Output
0x3-0x7	Reserved	

### • CKI: Transmit Clock Inversion

0: The data outputs (Data and Frame Sync signals) are shifted out on Transmit Clock falling edge. The Frame sync signal input is sampled on Transmit clock rising edge.

1: The data outputs (Data and Frame Sync signals) are shifted out on Transmit Clock rising edge. The Frame sync signal input is sampled on Transmit clock falling edge.

CKI affects only the Transmit Clock and not the output clock signal.

### • CKG: Transmit Clock Gating Selection

CKG	Transmit Clock Gating
0x0	None, continuous clock
0x1	Transmit Clock enabled only if TF Low
0x2	Transmit Clock enabled only if TF High
0x3	Reserved

- **START: Transmit Start Selection**

START	Transmit Start
0x0	Continuous, as soon as a word is written in the SSC_THR Register (if Transmit is enabled), and immediately after the end of transfer of the previous data.
0x1	Receive start
0x2	Detection of a low level on TF signal
0x3	Detection of a high level on TF signal
0x4	Detection of a falling edge on TF signal
0x5	Detection of a rising edge on TF signal
0x6	Detection of any level change on TF signal
0x7	Detection of any edge on TF signal
0x8 - 0xF	Reserved

- **STTDLY: Transmit Start Delay**

If STTDLY is not 0, a delay of STTDLY clock cycles is inserted between the start event and the actual start of transmission of data. When the Transmitter is programmed to start synchronously with the Receiver, the delay is also applied.

Note: STTDLY must be set carefully. If STTDLY is too short in respect to TAG (Transmit Sync Data) emission, data is emitted instead of the end of TAG.

- **PERIOD: Transmit Period Divider Selection**

This field selects the divider to apply to the selected Transmit Clock to generate a new Frame Sync Signal. If 0, no period signal is generated. If not 0, a period signal is generated at each  $2 \times (\text{PERIOD} + 1)$  Transmit Clock.

## 30.8.6 SSC Transmit Frame Mode Register

**Name:** SSC\_TFMR

**Access Type:** Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	FSEDGE
23	22	21	20	19	18	17	16
FSDEN	FSOS			FSLEN			
15	14	13	12	11	10	9	8
–	–	–	–	DATNB			
7	6	5	4	3	2	1	0
MSBF	–	DATDEF	DATLEN				

- **DATLEN: Data Length**

0: Forbidden value (1-bit data length not supported).

Any other value: The bit stream contains DATLEN + 1 data bits. Moreover, it defines the transfer size performed by the PDC2 assigned to the Transmit. If DATLEN is lower or equal to 7, data transfers are bytes, if DATLEN is between 8 and 15 (included), half-words are transferred, and for any other value, 32-bit words are transferred.

- **DATDEF: Data Default Value**

This bit defines the level driven on the TD pin while out of transmission. Note that if the pin is defined as multi-drive by the PIO Controller, the pin is enabled only if the SCC TD output is 1.

- **MSBF: Most Significant Bit First**

0: The lowest significant bit of the data register is shifted out first in the bit stream.

1: The most significant bit of the data register is shifted out first in the bit stream.

- **DATNB: Data Number per frame**

This field defines the number of data words to be transferred after each transfer start, which is equal to (DATNB + 1).

- **FSLEN: Transmit Frame Sync Length**

This field defines the length of the Transmit Frame Sync signal and the number of bits shifted out from the Transmit Sync Data Register if FSDEN is 1.

Pulse length is equal to (FSLEN + 1) Transmit Clock periods, i.e., the pulse length can range from 1 to 16 Transmit Clock periods. If FSLEN is 0, the Transmit Frame Sync signal is generated during one Transmit Clock period.

- **FSOS: Transmit Frame Sync Output Selection**

FSOS	Selected Transmit Frame Sync Signal	TF Pin
0x0	None	Input-only
0x1	Negative Pulse	Output
0x2	Positive Pulse	Output
0x3	Driven Low during data transfer	Output
0x4	Driven High during data transfer	Output
0x5	Toggling at each start of data transfer	Output
0x6-0x7	Reserved	Undefined

- **FSDEN: Frame Sync Data Enable**

0: The TD line is driven with the default value during the Transmit Frame Sync signal.

1: SSC\_TSHR value is shifted out during the transmission of the Transmit Frame Sync signal.

- **FSEEDGE: Frame Sync Edge Detection**

Determines which edge on frame sync will generate the interrupt TXSYN (Status Register).

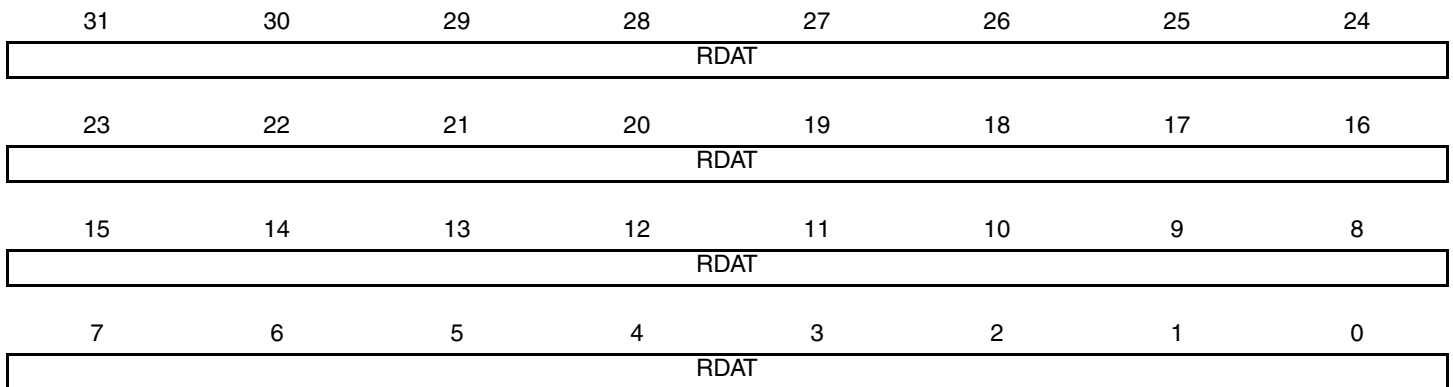
FSEEDGE	Frame Sync Edge Detection
0x0	Positive Edge Detection
0x1	Negative Edge Detection



## 30.8.7 SSC Receive Holding Register

**Name:** SSC\_RHR

**Access Type:** Read-only



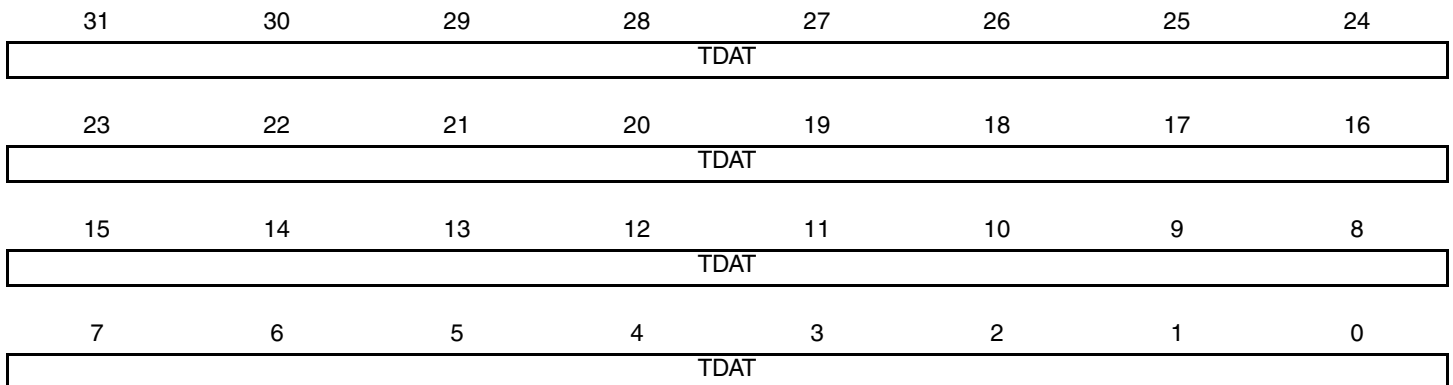
- **RDAT: Receive Data**

Right aligned regardless of the number of data bits defined by DATLEN in SSC\_RFMR.

## 30.8.8 SSC Transmit Holding Register

**Name:** SSC\_THR

**Access Type:** Write-only



- **TDAT: Transmit Data**

Right aligned regardless of the number of data bits defined by DATLEN in SSC\_TFMR.

## 30.8.9 SSC Receive Synchronization Holding Register

**Name:** SSC\_RSHR

**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
RSDAT							
7	6	5	4	3	2	1	0
RSDAT							

- **RSDAT: Receive Synchronization Data**

## 30.8.10 SSC Transmit Synchronization Holding Register

**Name:** SSC\_TSHR

**Access Type:** Read/Write

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

- **TSDAT: Transmit Synchronization Data**

## 30.8.11 SSC Receive Compare 0 Register

**Name:** SSC\_RC0R

**Access Type:** Read/Write

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
CP0							
7	6	5	4	3	2	1	0
CP0							

- CP0: Receive Compare Data 0

## 30.8.12 SSC Receive Compare 1 Register

**Name:** SSC\_RC1R

**Access Type:** Read/Write

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
CP1							
7	6	5	4	3	2	1	0
CP1							

- CP1: Receive Compare Data 1

## 30.8.13 SSC Status Register

**Name:** SSC\_SR

**Access Type:** Read-only

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	RXEN	TXEN
15	14	13	12	11	10	9	8
-	-	-	-	RXSYN	TXSYN	CP1	CP0
7	6	5	4	3	2	1	0
RXBUFF	ENDRX	OVRUN	RXRDY	TXBUFE	ENDTX	TXEMPTY	TXRDY

- **TXRDY: Transmit Ready**

0: Data has been loaded in SSC\_THR and is waiting to be loaded in the Transmit Shift Register (TSR).

1: SSC\_THR is empty.

- **TXEMPTY: Transmit Empty**

0: Data remains in SSC\_THR or is currently transmitted from TSR.

1: Last data written in SSC\_THR has been loaded in TSR and last data loaded in TSR has been transmitted.

- **ENDTX: End of Transmission**

0: The register SSC\_TCR has not reached 0 since the last write in SSC\_TCR or SSC\_TNCR.

1: The register SSC\_TCR has reached 0 since the last write in SSC\_TCR or SSC\_TNCR.

- **TXBUFE: Transmit Buffer Empty**

0: SSC\_TCR or SSC\_TNCR have a value other than 0.

1: Both SSC\_TCR and SSC\_TNCR have a value of 0.

- **RXRDY: Receive Ready**

0: SSC\_RHR is empty.

1: Data has been received and loaded in SSC\_RHR.

- **OVRUN: Receive Overrun**

0: No data has been loaded in SSC\_RHR while previous data has not been read since the last read of the Status Register.

1: Data has been loaded in SSC\_RHR while previous data has not yet been read since the last read of the Status Register.

- **ENDRX: End of Reception**

0: Data is written on the Receive Counter Register or Receive Next Counter Register.

1: End of PDC transfer when Receive Counter Register has arrived at zero.

- **RXBUFF: Receive Buffer Full**

0: SSC\_RCR or SSC\_RNCR have a value other than 0.

1: Both SSC\_RCR and SSC\_RNCR have a value of 0.

- **CP0: Compare 0**

0: A compare 0 has not occurred since the last read of the Status Register.

1: A compare 0 has occurred since the last read of the Status Register.

- **CP1: Compare 1**

0: A compare 1 has not occurred since the last read of the Status Register.

1: A compare 1 has occurred since the last read of the Status Register.

- **TXSYN: Transmit Sync**

0: A Tx Sync has not occurred since the last read of the Status Register.

1: A Tx Sync has occurred since the last read of the Status Register.

- **RXSYN: Receive Sync**

0: An Rx Sync has not occurred since the last read of the Status Register.

1: An Rx Sync has occurred since the last read of the Status Register.

- **TXEN: Transmit Enable**

0: Transmit is disabled.

1: Transmit is enabled.

- **RXEN: Receive Enable**

0: Receive is disabled.

1: Receive is enabled.

## 30.8.14 SSC Interrupt Enable Register

**Name:** SSC\_IER

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	RXSYN	TXSYN	CP1	CP0
7	6	5	4	3	2	1	0
RXBUFF	ENDRX	OVRUN	RXRDY	TXBUFE	ENDTX	TXEMPTY	TXRDY

- **TXRDY: Transmit Ready Interrupt Enable**

0: No effect.

1: Enables the Transmit Ready Interrupt.

- **TXEMPTY: Transmit Empty Interrupt Enable**

0: No effect.

1: Enables the Transmit Empty Interrupt.

- **ENDTX: End of Transmission Interrupt Enable**

0: No effect.

1: Enables the End of Transmission Interrupt.

- **TXBUFE: Transmit Buffer Empty Interrupt Enable**

0: No effect.

1: Enables the Transmit Buffer Empty Interrupt

- **RXRDY: Receive Ready Interrupt Enable**

0: No effect.

1: Enables the Receive Ready Interrupt.

- **OVRUN: Receive Overrun Interrupt Enable**

0: No effect.

1: Enables the Receive Overrun Interrupt.

- **ENDRX: End of Reception Interrupt Enable**

0: No effect.

1: Enables the End of Reception Interrupt.

- **RXBUFF: Receive Buffer Full Interrupt Enable**

0: No effect.

1: Enables the Receive Buffer Full Interrupt.

- **CP0: Compare 0 Interrupt Enable**

0: No effect.

1: Enables the Compare 0 Interrupt.

- **CP1: Compare 1 Interrupt Enable**

0: No effect.

1: Enables the Compare 1 Interrupt.

- **TXSYN: Tx Sync Interrupt Enable**

0: No effect.

1: Enables the Tx Sync Interrupt.

- **RXSYN: Rx Sync Interrupt Enable**

0: No effect.

1: Enables the Rx Sync Interrupt.



## 30.8.15 SSC Interrupt Disable Register

**Name:** SSC\_IDR

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	RXSYN	TXSYN	CP1	CP0
7	6	5	4	3	2	1	0
RXBUFF	ENDRX	OVRUN	RXRDY	TXBUFE	ENDTX	TXEMPTY	TXRDY

- **TXRDY: Transmit Ready Interrupt Disable**

0: No effect.

1: Disables the Transmit Ready Interrupt.

- **TXEMPTY: Transmit Empty Interrupt Disable**

0: No effect.

1: Disables the Transmit Empty Interrupt.

- **ENDTX: End of Transmission Interrupt Disable**

0: No effect.

1: Disables the End of Transmission Interrupt.

- **TXBUFE: Transmit Buffer Empty Interrupt Disable**

0: No effect.

1: Disables the Transmit Buffer Empty Interrupt.

- **RXRDY: Receive Ready Interrupt Disable**

0: No effect.

1: Disables the Receive Ready Interrupt.

- **OVRUN: Receive Overrun Interrupt Disable**

0: No effect.

1: Disables the Receive Overrun Interrupt.

- **ENDRX: End of Reception Interrupt Disable**

0: No effect.

1: Disables the End of Reception Interrupt.

- **RXBUFF: Receive Buffer Full Interrupt Disable**

0: No effect.

1: Disables the Receive Buffer Full Interrupt.

- **CP0: Compare 0 Interrupt Disable**

0: No effect.

1: Disables the Compare 0 Interrupt.

- **CP1: Compare 1 Interrupt Disable**

0: No effect.

1: Disables the Compare 1 Interrupt.

- **TXSYN: Tx Sync Interrupt Enable**

0: No effect.

1: Disables the Tx Sync Interrupt.

- **RXSYN: Rx Sync Interrupt Enable**

0: No effect.

1: Disables the Rx Sync Interrupt.

## 30.8.16 SSC Interrupt Mask Register

**Name:** SSC\_IMR

**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
–	–	–	–	RXSYN	TXSYN	CP1	CP0
7	6	5	4	3	2	1	0
RXBUF	ENDRX	OVRUN	RXRDY	TXBUFE	ENDTX	TXEMPTY	TXRDY

- **TXRDY: Transmit Ready Interrupt Mask**

0: The Transmit Ready Interrupt is disabled.

1: The Transmit Ready Interrupt is enabled.

- **TXEMPTY: Transmit Empty Interrupt Mask**

0: The Transmit Empty Interrupt is disabled.

1: The Transmit Empty Interrupt is enabled.

- **ENDTX: End of Transmission Interrupt Mask**

0: The End of Transmission Interrupt is disabled.

1: The End of Transmission Interrupt is enabled.

- **TXBUFE: Transmit Buffer Empty Interrupt Mask**

0: The Transmit Buffer Empty Interrupt is disabled.

1: The Transmit Buffer Empty Interrupt is enabled.

- **RXRDY: Receive Ready Interrupt Mask**

0: The Receive Ready Interrupt is disabled.

1: The Receive Ready Interrupt is enabled.

- **OVRUN: Receive Overrun Interrupt Mask**

0: The Receive Overrun Interrupt is disabled.

1: The Receive Overrun Interrupt is enabled.

- **ENDRX: End of Reception Interrupt Mask**

0: The End of Reception Interrupt is disabled.

1: The End of Reception Interrupt is enabled.

- **RXBUFE: Receive Buffer Full Interrupt Mask**

0: The Receive Buffer Full Interrupt is disabled.

1: The Receive Buffer Full Interrupt is enabled.

- **CP0: Compare 0 Interrupt Mask**

0: The Compare 0 Interrupt is disabled.

1: The Compare 0 Interrupt is enabled.

- **CP1: Compare 1 Interrupt Mask**

0: The Compare 1 Interrupt is disabled.

1: The Compare 1 Interrupt is enabled.

- **TXSYN: Tx Sync Interrupt Mask**

0: The Tx Sync Interrupt is disabled.

1: The Tx Sync Interrupt is enabled.

- **RXSYN: Rx Sync Interrupt Mask**

0: The Rx Sync Interrupt is disabled.

1: The Rx Sync Interrupt is enabled.

## 31. Timer/Counter (TC)

### 31.1 Overview

The Timer Counter (TC) includes three identical 16-bit Timer Counter channels.

Each channel can be independently programmed to perform a wide range of functions including frequency measurement, event counting, interval measurement, pulse generation, delay timing and pulse width modulation.

Each channel has three external clock inputs, five internal clock inputs and two multi-purpose input/output signals which can be configured by the user. Each channel drives an internal interrupt signal which can be programmed to generate processor interrupts.

The Timer Counter block has two global registers which act upon all three TC channels.

The Block Control Register allows the three channels to be started simultaneously with the same instruction.

The Block Mode Register defines the external clock inputs for each channel, allowing them to be chained.

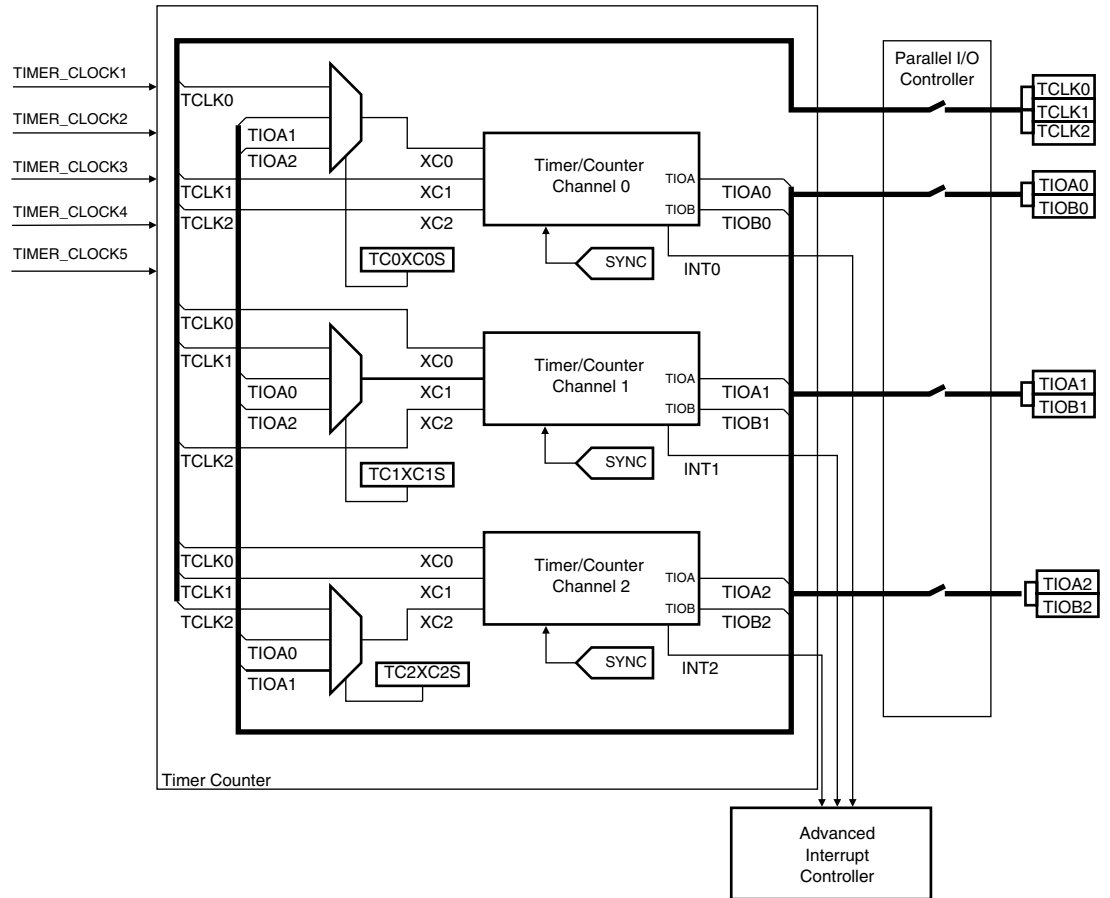
[Table 31-1](#) gives the assignment of the device Timer Counter clock inputs common to Timer Counter 0 to 2.

**Table 31-1.** Timer Counter Clock Assignment

Name	Definition
TIMER_CLOCK1	MCK/2
TIMER_CLOCK2	MCK/8
TIMER_CLOCK3	MCK/32
TIMER_CLOCK4	MCK/128
TIMER_CLOCK5	MCK/1024

## 31.2 Block Diagram

**Figure 31-1.** Timer/Counter Block Diagram



**Table 31-2.** Signal Name Description

Block/Channel	Signal Name	Description
Channel Signal	XC0, XC1, XC2	External Clock Inputs
	TIOA	Capture Mode: Timer/Counter Input Waveform Mode: Timer/Counter Output
	TIOB	Capture Mode: Timer/Counter Input Waveform Mode: Timer/Counter Input/output
	INT	Interrupt Signal Output
	SYNC	Synchronization Input Signal

## 31.3 Pin Name List

**Table 31-3.** TC pin list

Pin Name	Description	Type
TCLK0-TCLK2	External Clock Input	Input
TIOA0-TIOA2	I/O Line A	I/O
TIOB0-TIOB2	I/O Line B	I/O

## 31.4 Product Dependencies

### 31.4.1 I/O Lines

The pins used for interfacing the compliant external devices may be multiplexed with PIO lines. The programmer must first program the PIO controllers to assign the TC pins to their peripheral functions.

### 31.4.2 Power Management

The TC is clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the Timer/Counter clock.

### 31.4.3 Interrupt

The TC has an interrupt line connected to the Advanced Interrupt Controller (AIC). Handling the TC interrupt requires programming the AIC before configuring the TC.

## 31.5 Functional Description

### 31.5.1 TC Description

The three channels of the Timer/Counter are independent and identical in operation. The registers for channel programming are listed in [Table 31-5 on page 373](#).

#### 31.5.1.1 16-bit Counter

Each channel is organized around a 16-bit counter. The value of the counter is incremented at each positive edge of the selected clock. When the counter has reached the value 0xFFFF and passes to 0x0000, an overflow occurs and the COVFS bit in TC\_SR (Status Register) is set.

The current value of the counter is accessible in real time by reading the Counter Value Register, TC\_CV. The counter can be reset by a trigger. In this case, the counter value passes to 0x0000 on the next valid edge of the selected clock.

#### 31.5.1.2 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 configurable internal signals TIOA0, TIOA1 or TIOA2 for chaining by programming the TC\_BMR (Block Mode). See [Figure 31-2](#).

Each channel can independently select an internal or external clock source for its counter:

- Internal clock signals: TIMER\_CLOCK1, TIMER\_CLOCK2, TIMER\_CLOCK3, TIMER\_CLOCK4, TIMER\_CLOCK5
- External clock signals: XC0, XC1 or XC2

This selection is made by the TCCLKS bits in the TC Channel Mode Register.

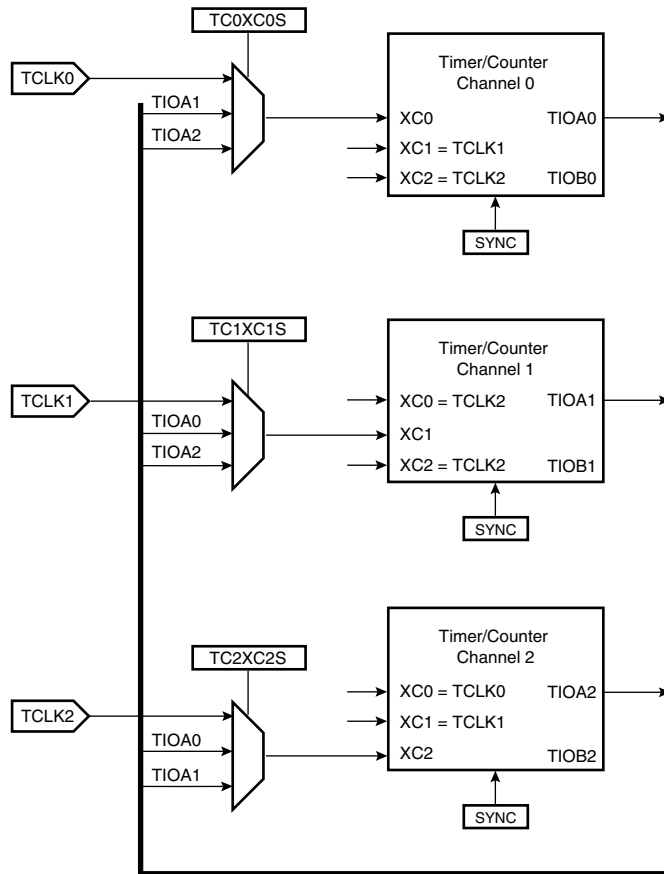
The selected clock can be inverted with the CLKI bit in TC\_CMR. This allows counting on the opposite edges of the clock.

The burst function allows the clock to be validated when an external signal is high. The BURST parameter in the Mode Register defines this signal (none, XC0, XC1, XC2). See [Figure 31-3](#)

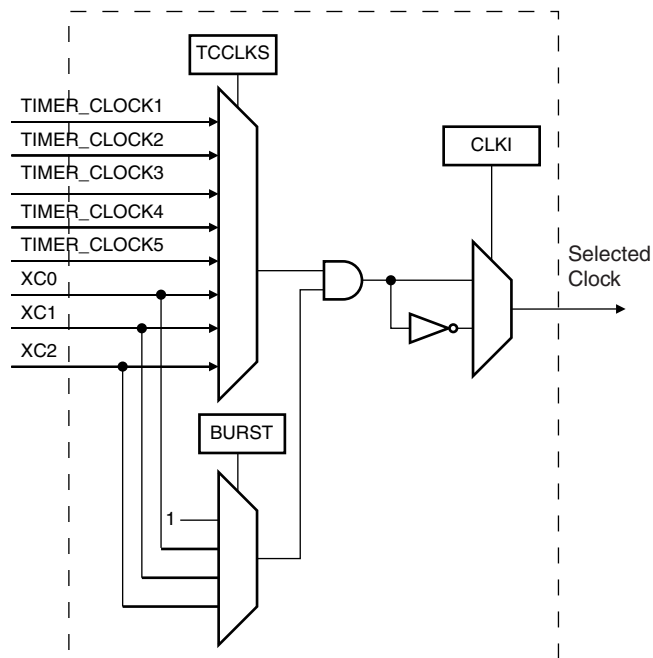
**Note:** In all cases, if an external clock is used, the duration of each of its levels must be longer than the master clock period. The external clock frequency must be at least 2.5 times lower than the master clock



**Figure 31-2.** Clock Chaining Selection



**Figure 31-3.** Clock Selection

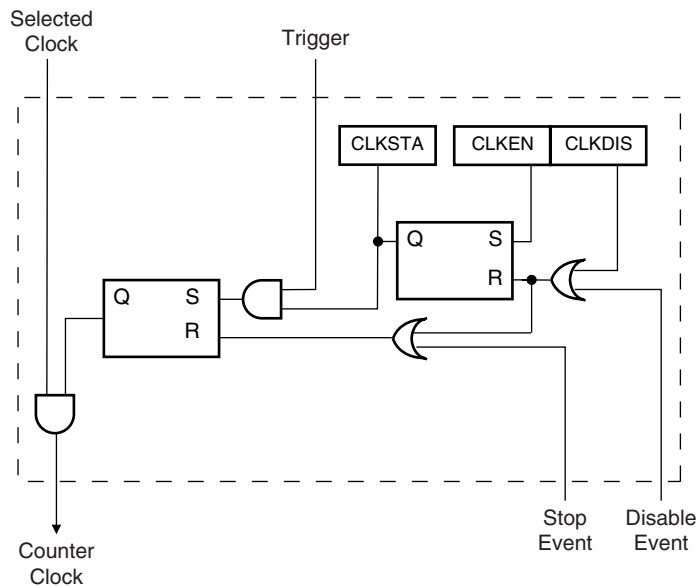


### 31.5.1.3 Clock Control

The clock of each counter can be controlled in two different ways: it can be enabled/disabled and started/stopped. See [Figure 31-4](#).

- The clock can be enabled or disabled by the user with the CLKEN and the CLKDIS commands in the Control Register. In Capture Mode it can be disabled by an RB load event if LDBDIS is set to 1 in 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 Control Register can re-enable the clock. When the clock is enabled, the CLKSTA bit is set in the Status Register.
- The clock can also be started or stopped: a trigger (software, synchro, external or compare) always starts the clock. The clock can be stopped by an RB load event in Capture Mode (LDBSTOP = 1 in TC\_CMR) or a RC compare event in Waveform Mode (CPCSTOP = 1 in TC\_CMR). The start and the stop commands have effect only if the clock is enabled.

**Figure 31-4.** Clock Control



### 31.5.1.4 TC Operating Modes

Each channel can independently operate in two different modes:

- Capture Mode provides measurement on signals.
- Waveform Mode provides wave generation.

The TC Operating Mode is programmed with the WAVE bit in the TC Channel Mode Register.

In Capture Mode, TIOA and TIOB are configured as inputs.

In Waveform Mode, TIOA is always configured to be an output and TIOB is an output if it is not selected to be the external trigger.

### 31.5.1.5 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.

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 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 ENETRГ in TC\_CMR.

If an external trigger is used, the duration of the pulses must be longer than the master clock period in order to be detected.

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.

## 31.5.2 Capture Operating Mode

This mode is entered by clearing the WAVE parameter in TC\_CMR (Channel Mode Register).

Capture Mode allows the TC channel to perform measurements such as pulse timing, frequency, period, duty cycle and phase on TIOA and TIOB signals which are considered as inputs.

Figure 31-5 shows the configuration of the TC channel when programmed in Capture Mode.

### 31.5.2.1 Capture Registers A and B

Registers A and B (RA and RB) are used as capture registers. This means that they can be loaded with the counter value when a programmable event occurs on the signal TIOA.

The LDRA parameter in TC\_CMR defines the TIOA edge for the loading of register A, and the LDRB parameter defines the TIOA 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.

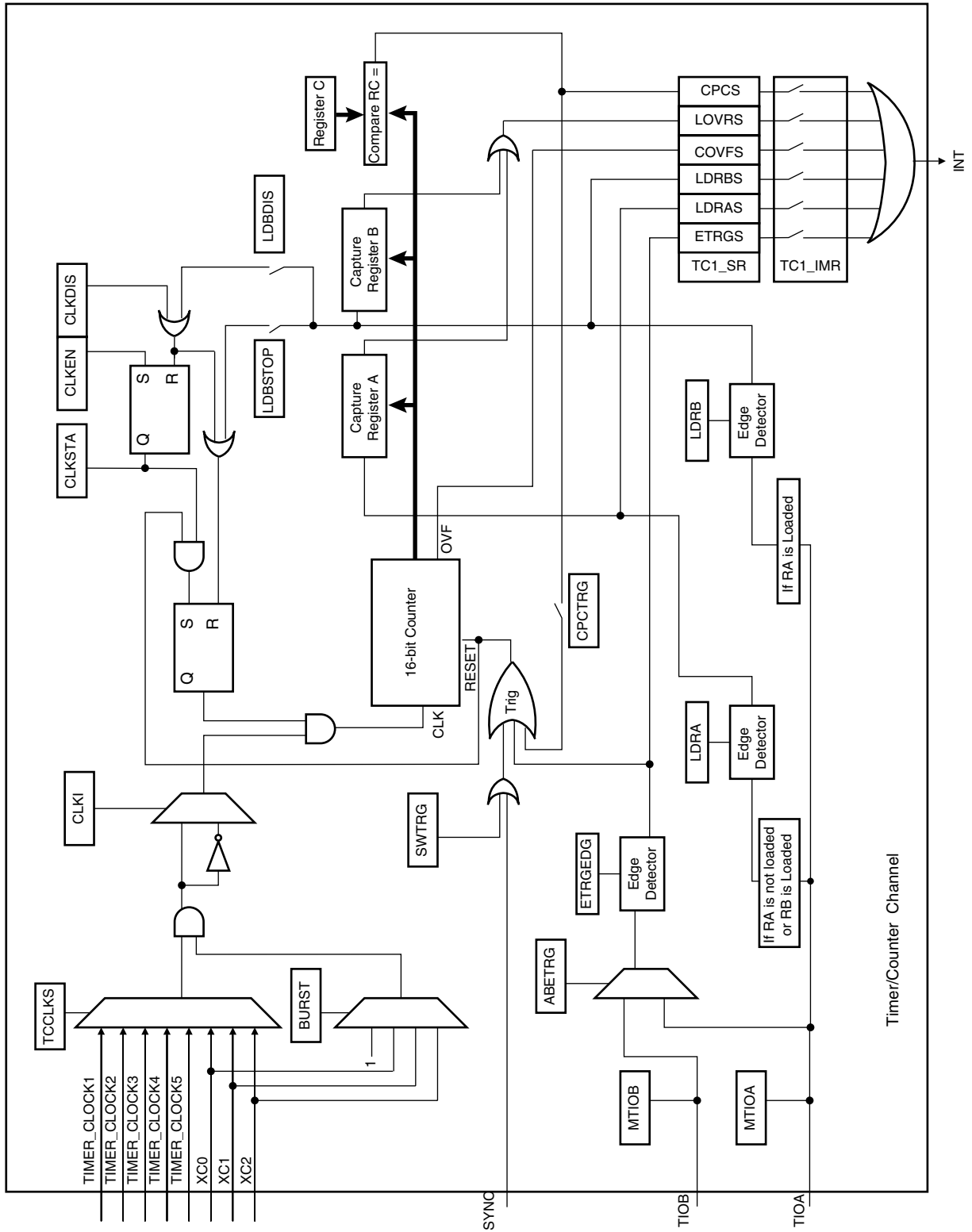
Loading RA or RB before the read of the last value loaded sets the Overrun Error Flag (LOVRS) in TC\_SR (Status Register). In this case, the old value is overwritten.

### 31.5.2.2 Trigger Conditions

In addition to the SYNC signal, the software trigger and the RC compare trigger, an external trigger can be defined.

The ABETRГ bit in TC\_CMR selects TIOA or TIOB input signal as an external trigger. The ETRGEDG parameter defines the edge (rising, falling or both) detected to generate an external trigger. If ETRGEDG = 0 (none), the external trigger is disabled.

Figure 31-5. Capture Mode



## 31.5.3 Waveform Operating Mode

Waveform operating mode is entered by setting the WAVE parameter in TC\_CMR (Channel Mode Register).

In Waveform Operating Mode the TC channel generates 1 or 2 PWM signals with the same frequency and independently programmable duty cycles, or generates different types of one-shot or repetitive pulses.

In this mode, TIOA is configured as an output and TIOB is defined as an output if it is not used as an external event (EEVT parameter in TC\_CMR).

[Figure 31-6](#) shows the configuration of the TC channel when programmed in Waveform Operating Mode.

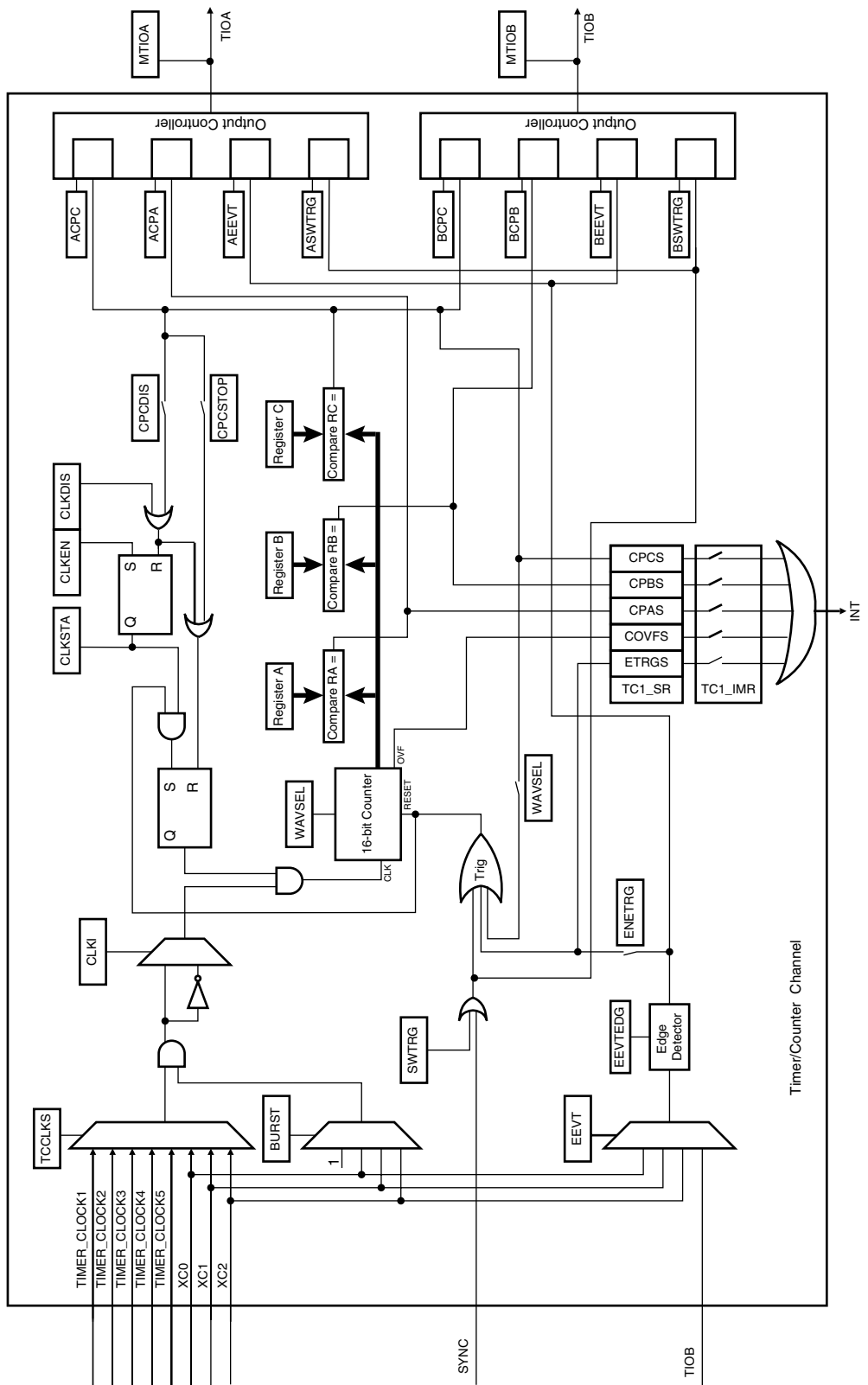
### 31.5.3.1 Waveform Selection

Depending on the WAVSEL parameter in TC\_CMR (Channel Mode Register), the behavior of TC\_CV varies.

With any selection, RA, RB and RC can all be used as compare registers.

RA Compare is used to control the TIOA output, RB Compare is used to control the TIOB output (if correctly configured) and RC Compare is used to control TIOA and/or TIOB outputs.

Figure 31-6. Waveform Mode



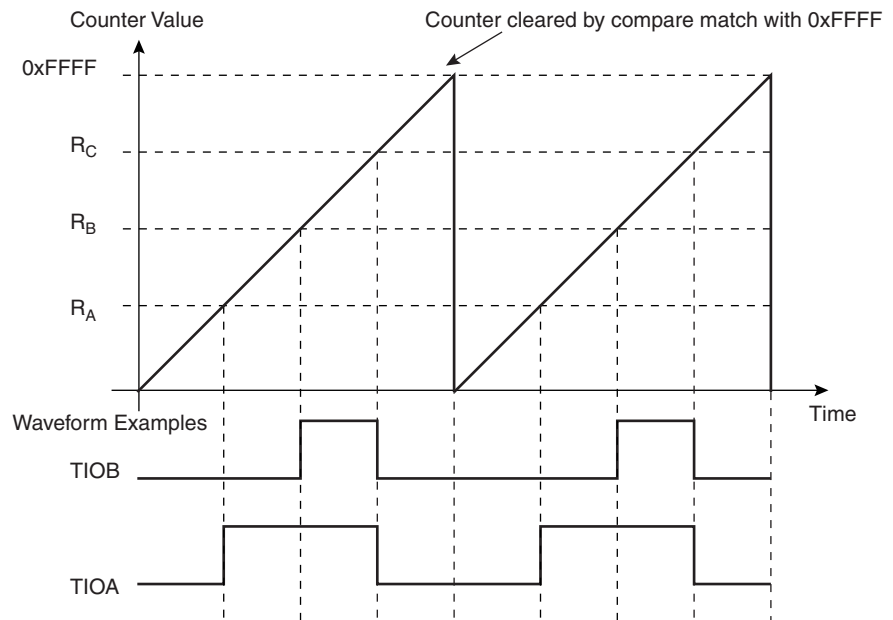
## 31.5.3.2 WAVSEL = 00

When WAVSEL = 00, the value of TC\_CV is incremented from 0 to 0xFFFF. Once 0xFFFF has been reached, the value of TC\_CV is reset. Incrementation of TC\_CV starts again and the cycle continues. See [Figure 31-7](#).

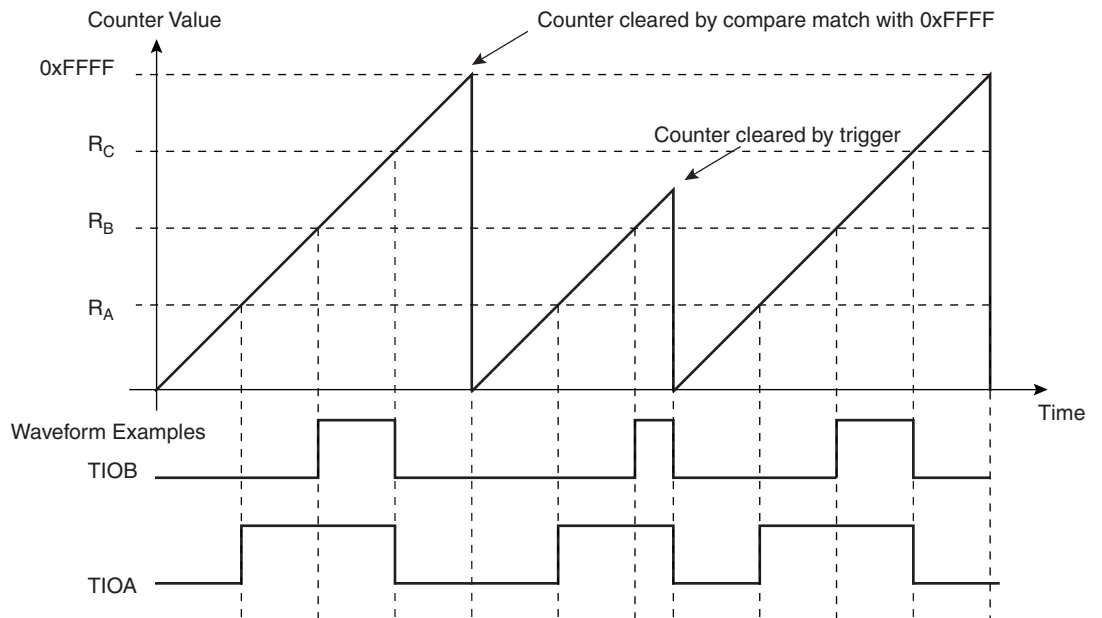
An external event trigger or a software trigger can reset the value of TC\_CV. It is important to note that the trigger may occur at any time. See [Figure 31-8](#).

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 31-7.** WAVSEL= 00 without trigger



**Figure 31-8.** WAVSEL= 00 with trigger



31.5.3.3 WAVSEL = 10

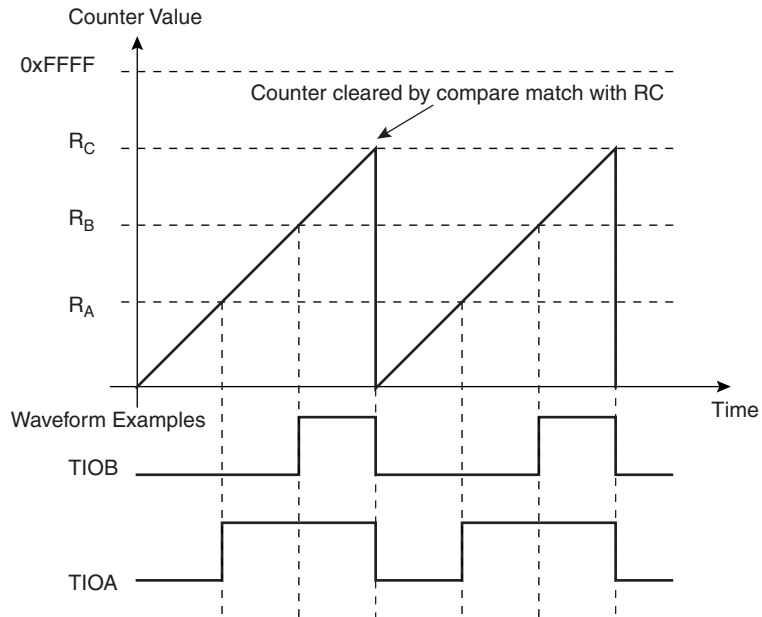
When WAVSEL = 10, the value of TC\_CV is incremented from 0 to the value of RC, then automatically reset on a RC Compare. Once the value of TC\_CV has been reset, it is then incremented and so on. See [Figure 31-9](#).

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 31-10](#).

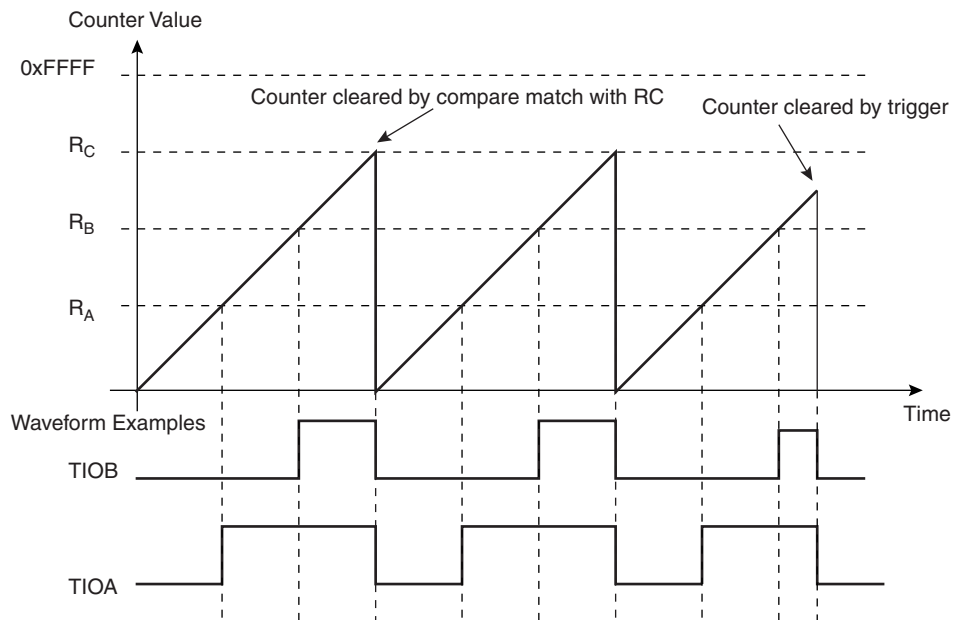
In addition, RC Compare can stop the counter clock (CPCSTOP = 1 in TC\_CMR) and/or disable the counter clock (CPCDIS = 1 in TC\_CMR).



**Figure 31-9.** WAVSEL = 10 Without Trigger



**Figure 31-10.** WAVSEL = 10 With Trigger



### 31.5.3.4 WAVSEL = 01

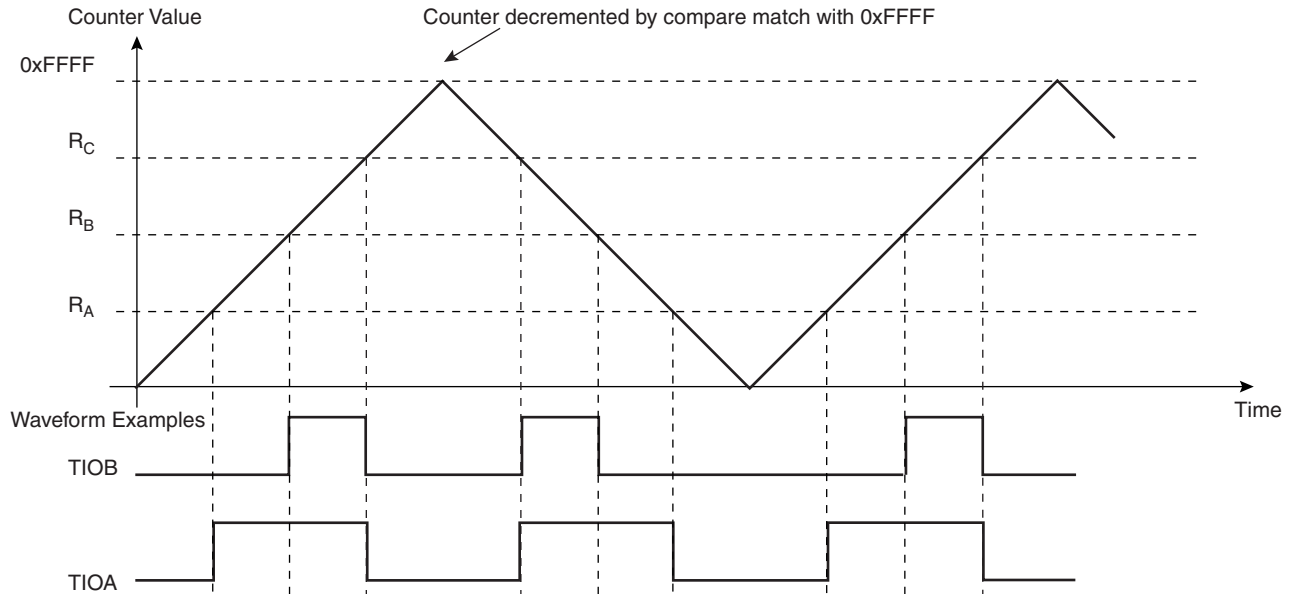
When WAVSEL = 01, the value of TC\_CV is incremented from 0 to 0xFFFF. Once 0xFFFF is reached, the value of TC\_CV is decremented to 0, then re-incremented to 0xFFFF and so on. See [Figure 31-11](#).

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 31-12](#).

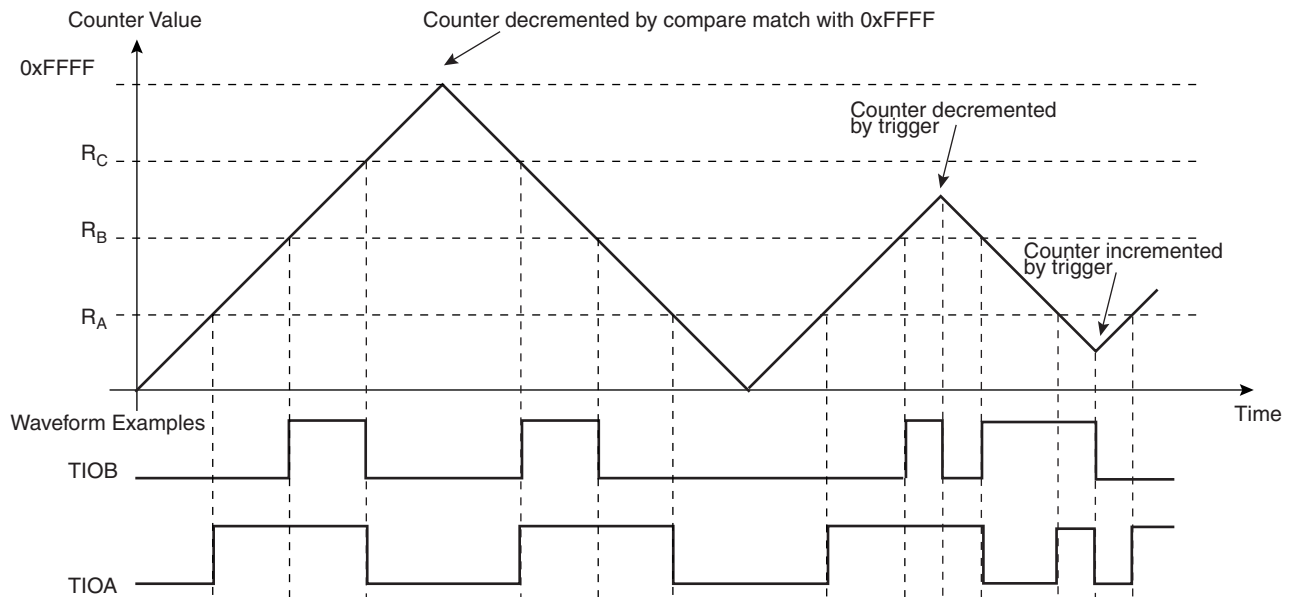
RC Compare cannot be programmed to generate a trigger in this configuration.

At the same time, RC Compare can stop the counter clock (CPCSTOP = 1) and/or disable the counter clock (CPCDIS = 1).

**Figure 31-11. WAVSEL = 01 Without Trigger**



**Figure 31-12. WAVSEL = 01 With Trigger**



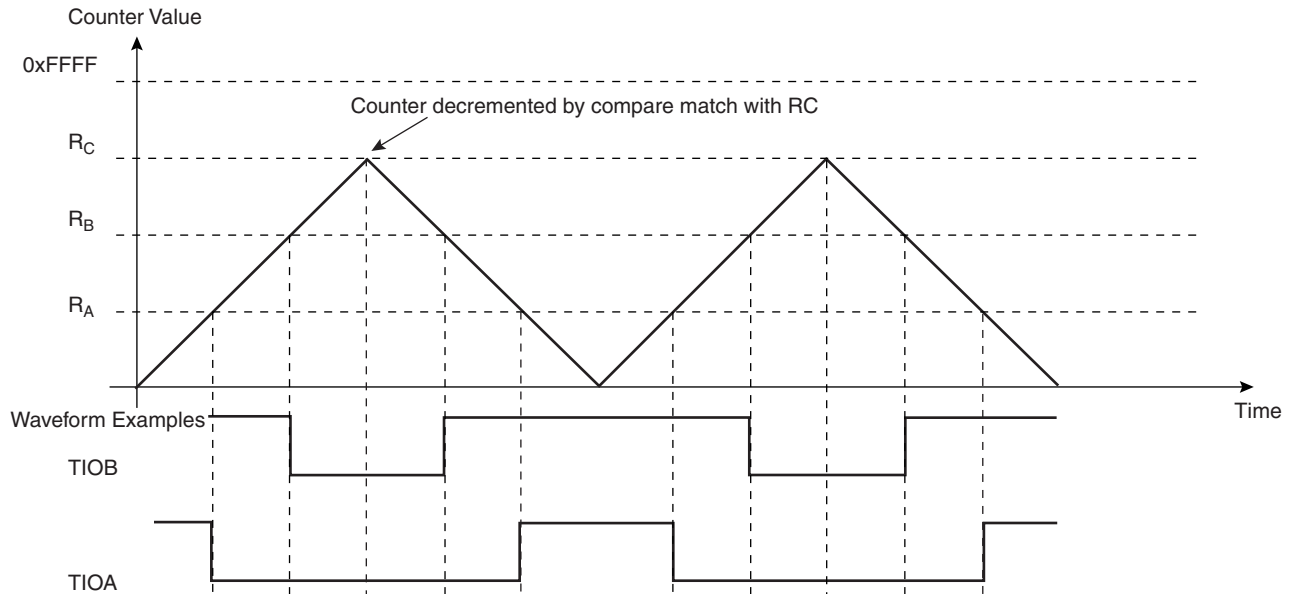
**31.5.3.5 WAVSEL = 11**

When WAVSEL = 11, the value of TC\_CV is incremented from 0 to R<sub>C</sub>. Once R<sub>C</sub> is reached, the value of TC\_CV is decremented to 0, then re-incremented to R<sub>C</sub> and so on. See [Figure 31-13](#).

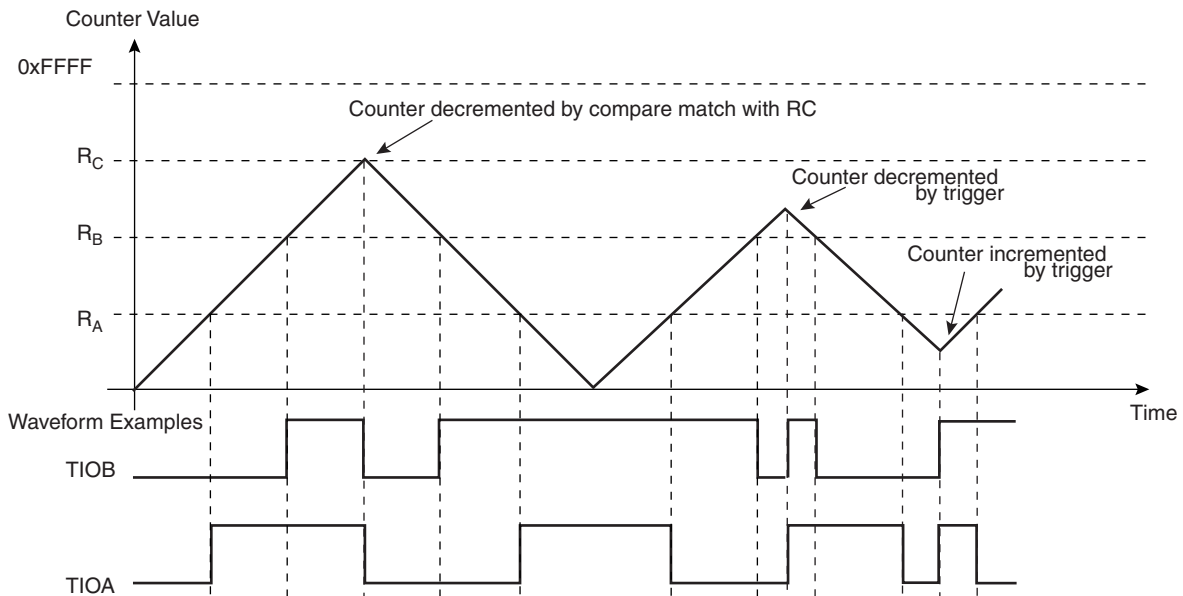
A trigger such as an external event or a software trigger can modify TC\_CV at any time. If a trigger occurs while TC\_CV is incrementing, TC\_CV then decrements. If a trigger is received while TC\_CV is decrementing, TC\_CV then increments. See [Figure 31-14](#).

RC Compare can stop the counter clock (CPCSTOP = 1) and/or disable the counter clock (CPCDIS = 1).

**Figure 31-13. WAVSEL = 11 Without Trigger**



**Figure 31-14. WAVSEL = 11 With Trigger**



### 31.5.4 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 EEVTEG parameter defines the trigger edge for each of the possible external triggers (rising, falling or both). If EEVTEG 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 ENETR in TC\_CMR.

As in Capture Mode, the SYNC signal and the software trigger are also available as triggers. RC Compare can also be used as a trigger depending on the parameter WAVSEL.

### 31.5.5 Output Controller

The output controller defines the output level changes on TIOA and TIOB following an event. TIOB control is used only if TIOB is defined as output (not as an external event).

The following events control TIOA and TIOB: software trigger, external event and RC compare. RA compare controls TIOA and RB compare controls TIOB. Each of these events can be programmed to set, clear or toggle the output as defined in the corresponding parameter in TC\_CMR.

## 31.6 Timer/Counter (TC) User Interface

### 31.6.1 Global Register Mapping

**Table 31-4.** Timer/Counter (TC) Global Register Map

Offset	Channel/Register	Name	Access	Reset Value
0x00	TC Channel 0		See <a href="#">Table 31-5</a>	
0x40	TC Channel 1		See <a href="#">Table 31-5</a>	
0x80	TC Channel 2		See <a href="#">Table 31-5</a>	
0xC0	TC Block Control Register	TC_BCR	Write-only	–
0xC4	TC Block Mode Register	TC_BMR	Read/Write	0

TC\_BCR (Block Control Register) and TC\_BMR (Block Mode Register) control the whole TC block. TC channels are controlled by the registers listed in [Table 31-5](#). The offset of each of the channel registers in [Table 31-5](#) is in relation to the offset of the corresponding channel as mentioned in [Table 31-5](#).

### 31.6.2 Channel Memory Mapping

**Table 31-5.** TC Channel Memory Map

Offset	Register	Name	Access	Reset Value
0x00	Channel Control Register	TC_CCR	Write-only	–
0x04	Channel Mode Register	TC_CMR	Read/Write	0
0x08	Reserved			–
0x0C	Reserved			–
0x10	Counter Value	TC_CV	Read-only	0
0x14	Register A	TC_RA	Read/Write <sup>(1)</sup>	0
0x18	Register B	TC_RB	Read/Write <sup>(1)</sup>	0
0x1C	Register C	TC_RC	Read/Write	0
0x20	Status Register	TC_SR	Read-only	0
0x24	Interrupt Enable Register	TC_IER	Write-only	–
0x28	Interrupt Disable Register	TC_IDR	Write-only	–
0x2C	Interrupt Mask Register	TC_IMR	Read-only	0
0xFC	Reserved	–	–	–

Notes: 1. Read-only if WAVE = 0

### 31.6.3 TC Block Control Register

Register Name: TC\_BCR

Access Type: Write-only

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

- **SYNC: Synchro Command**

0 = No effect.

1 = Asserts the SYNC signal which generates a software trigger simultaneously for each of the channels.

## 31.6.4 TC Block Mode Register

Register Name: TC\_BMR

Access Type: Read/Write

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	-	-	-	-	-
7	6	5	4	3	2	1	0
-	-	TC2XC2S		TCXC1S		TC0XC0S	

- **TC0XC0S: External Clock Signal 0 Selection**

TC0XC0S		Signal Connected to XC0
0	0	TCLK0
0	1	none
1	0	TIOA1
1	1	TIOA2

- **TC1XC1S: External Clock Signal 1 Selection**

TC1XC1S		Signal Connected to XC1
0	0	TCLK1
0	1	none
1	0	TIOA0
1	1	TIOA2

- **TC2XC2S: External Clock Signal 2 Selection**

TC2XC2S		Signal Connected to XC2
0	0	TCLK2
0	1	none
1	0	TIOA0
1	1	TIOA1

### 31.6.5 TC Channel Control Register

**Register Name:** TC\_CCR

**Access Type:** Write-only

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

- **CLKEN: Counter Clock Enable Command**

0 = No effect.

1 = Enables the clock if CLKDIS is not 1.

- **CLKDIS: Counter Clock Disable Command**

0 = No effect.

1 = Disables the clock.

- **SWTRG: Software Trigger Command**

0 = No effect.

1 = A software trigger is performed: the counter is reset and the clock is started.



## 31.6.6 TC Channel Mode Register: Capture Mode

Register Name: TC\_CMCR

Access Type: 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 = 0	CPCTRG	–	–	–	ABETRG	ETRGEDG	
7	6	5	4	3	2	1	0
LDBDIS	LDBSTOP	BURST		CLKI	TCCLKS		

- **TCCLKS: Clock Selection**

TCCLKS			Clock Selected
0	0	0	TIMER_CLOCK1
0	0	1	TIMER_CLOCK2
0	1	0	TIMER_CLOCK3
0	1	1	TIMER_CLOCK4
1	0	0	TIMER_CLOCK5
1	0	1	XC0
1	1	0	XC1
1	1	1	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**

BURST		
0	0	The clock is not gated by an external signal.
0	1	XC0 is ANDed with the selected clock.
1	0	XC1 is ANDed with the selected clock.
1	1	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**

ETRGEDG		Edge
0	0	none
0	1	rising edge
1	0	falling edge
1	1	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**

0 = Capture Mode is enabled.

1 = Capture Mode is disabled (Waveform Mode is enabled).

- **LDRA: RA Loading Selection**

LDRA		Edge
0	0	none
0	1	rising edge of TIOA
1	0	falling edge of TIOA
1	1	each edge of TIOA

- **LDRB: RB Loading Selection**

LDRB		Edge
0	0	none
0	1	rising edge of TIOA
1	0	falling edge of TIOA
1	1	each edge of TIOA

## 31.6.7 TC Channel Mode Register: Waveform Mode

Register Name: TC\_CMCR

Access Type: 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 = 1	WAVSEL		ENETRGR	EEVT		EEVTEDG	
7	6	5	4	3	2	1	0
CPCDIS	CPCSTOP	BURST		CLKI	TCCLKS		

- **TCCLKS: Clock Selection**

TCCLKS			Clock Selected
0	0	0	TIMER_CLOCK1
0	0	1	TIMER_CLOCK2
0	1	0	TIMER_CLOCK3
0	1	1	TIMER_CLOCK4
1	0	0	TIMER_CLOCK5
1	0	1	XC0
1	1	0	XC1
1	1	1	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**

BURST		
0	0	The clock is not gated by an external signal.
0	1	XC0 is ANDed with the selected clock.
1	0	XC1 is ANDed with the selected clock.
1	1	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**

EEVTEDG		Edge
0	0	none
0	1	rising edge
1	0	falling edge
1	1	each edge

- **EEVT: External Event Selection**

EEVT		Signal selected as external event	TIOB Direction
0	0	TIOB	input <sup>(1)</sup>
0	1	XC0	output
1	0	XC1	output
1	1	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.

- **ENETRГ: External Event Trigger Enable**

0 = The external event has no effect on the counter and its clock. In this case, the selected external event only controls the TIOA output.

1 = The external event resets the counter and starts the counter clock.

- **WAVSEL: Waveform Selection**

WAVSEL		Effect
0	0	UP mode without automatic trigger on RC Compare
1	0	UP mode with automatic trigger on RC Compare
0	1	UPDOWN mode without automatic trigger on RC Compare
1	1	UPDOWN mode with automatic trigger on RC Compare

- **WAVE = 1**

0 = Waveform Mode is disabled (Capture Mode is enabled).

1 = Waveform Mode is enabled.

- **ACPA: RA Compare Effect on TIOA**

ACPA		Effect
0	0	none
0	1	set
1	0	clear
1	1	toggle

- **ACPC: RC Compare Effect on TIOA**

ACPC		Effect
0	0	none
0	1	set
1	0	clear
1	1	toggle

- **AEEVT: External Event Effect on TIOA**

AEEVT		Effect
0	0	none
0	1	set
1	0	clear
1	1	toggle

- **ASWTRG: Software Trigger Effect on TIOA**

ASWTRG		Effect
0	0	none
0	1	set
1	0	clear
1	1	toggle

- **BCPB: RB Compare Effect on TIOB**

BCPB		Effect
0	0	none
0	1	set
1	0	clear
1	1	toggle

- **BCPC: RC Compare Effect on TIOB**

BCPC		Effect
0	0	none
0	1	set
1	0	clear
1	1	toggle



• **BEEVT: External Event Effect on TIOB**

BEEVT		Effect
0	0	none
0	1	set
1	0	clear
1	1	toggle

• **BSWTRG: Software Trigger Effect on TIOB**

BSWTRG		Effect
0	0	none
0	1	set
1	0	clear
1	1	toggle



## 31.6.8 TC Counter Value Register

**Register Name:** TC\_CV

**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
CV							
7	6	5	4	3	2	1	0
CV							

- **CV: Counter Value**

CV contains the counter value in real time.



**31.6.9 TC Register A**

**Register Name:** TC\_RA

**Access Type:** Read-only if WAVE = 0, Read/Write if WAVE = 1

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

- RA: Register A**

RA contains the Register A value in real time.



## 31.6.10 TC Register B

**Register Name:** TC\_RB

**Access Type:** Read-only if WAVE = 0, Read/Write if WAVE = 1

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

- **RB: Register B**

RB contains the Register B value in real time.

## 31.6.11 TC Register C

**Register Name:** TC\_RC

**Access Type:** Read/Write

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

- **RC: Register C**

RC contains the Register C value in real time.

### 31.6.12 TC Status Register

**Register Name:** TC\_SR

**Access Type:** 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**

0 = No counter overflow has occurred since the last read of the Status Register.

1 = A counter overflow has occurred since the last read of the Status Register.

- **LOVRS: Load Overrun Status**

0 = Load overrun has not occurred since the last read of the Status Register or WAVE = 1.

1 = RA or RB have been loaded at least twice without any read of the corresponding register since the last read of the Status Register, if WAVE = 0.

- **CPAS: RA Compare Status**

0 = RA Compare has not occurred since the last read of the Status Register or WAVE = 0.

1 = RA Compare has occurred since the last read of the Status Register, if WAVE = 1.

- **CPBS: RB Compare Status**

0 = RB Compare has not occurred since the last read of the Status Register or WAVE = 0.

1 = RB Compare has occurred since the last read of the Status Register, if WAVE = 1.

- **CPCS: RC Compare Status**

0 = RC Compare has not occurred since the last read of the Status Register.

1 = RC Compare has occurred since the last read of the Status Register.

- **LDRAS: RA Loading Status**

0 = RA Load has not occurred since the last read of the Status Register or WAVE = 1.

1 = RA Load has occurred since the last read of the Status Register, if WAVE = 0.

- **LDRBS: RB Loading Status**

0 = RB Load has not occurred since the last read of the Status Register or WAVE = 1.

1 = RB Load has occurred since the last read of the Status Register, if WAVE = 0.

- **ETRGS: External Trigger Status**

0 = External trigger has not occurred since the last read of the Status Register.

1 = External trigger has occurred since the last read of the Status Register.

- **CLKSTA: Clock Enabling Status**

0 = Clock is disabled.

1 = Clock is enabled.

- **MTIOA: TIOA Mirror**

0 = TIOA is low. If WAVE = 0, this means that TIOA pin is low. If WAVE = 1, this means that TIOA is driven low.

1 = TIOA is high. If WAVE = 0, this means that TIOA pin is high. If WAVE = 1, this means that TIOA is driven high.

- **MTIOB: TIOB Mirror**

0 = TIOB is low. If WAVE = 0, this means that TIOB pin is low. If WAVE = 1, this means that TIOB is driven low.

1 = TIOB is high. If WAVE = 0, this means that TIOB pin is high. If WAVE = 1, this means that TIOB is driven high.

### 31.6.13 TC Interrupt Enable Register

**Register Name:** TC\_IER

**Access Type:** Write-only

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

## 31.6.14 TC Interrupt Disable Register

**Register Name:** TC\_IDR

**Access Type:** Write-only

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	-	-	-	-	-
7	6	5	4	3	2	1	0
ETRGS	LDRBS	LDRAS	CPCS	CPBS	CPAS	LOVRS	COVFS

- **COVFS: Counter Overflow**

0 = No effect.

1 = Disables the Counter Overflow Interrupt.

- **LOVRS: Load Overrun**

0 = No effect.

1 = Disables the Load Overrun Interrupt (if WAVE = 0).

- **CPAS: RA Compare**

0 = No effect.

1 = Disables the RA Compare Interrupt (if WAVE = 1).

- **CPBS: RB Compare**

0 = No effect.

1 = Disables the RB Compare Interrupt (if WAVE = 1).

- **CPCS: RC Compare**

0 = No effect.

1 = Disables the RC Compare Interrupt.

- **LDRAS: RA Loading**

0 = No effect.

1 = Disables the RA Load Interrupt (if WAVE = 0).

- **LDRBS: RB Loading**

0 = No effect.

1 = Disables the RB Load Interrupt (if WAVE = 0).

- **ETRGS: External Trigger**

0 = No effect.

1 = Disables the External Trigger Interrupt.

### 31.6.15 TC Interrupt Mask Register

Register Name: TC\_IMR

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
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. Pulse Width Modulation Controller (PWM)

### 32.1 Overview

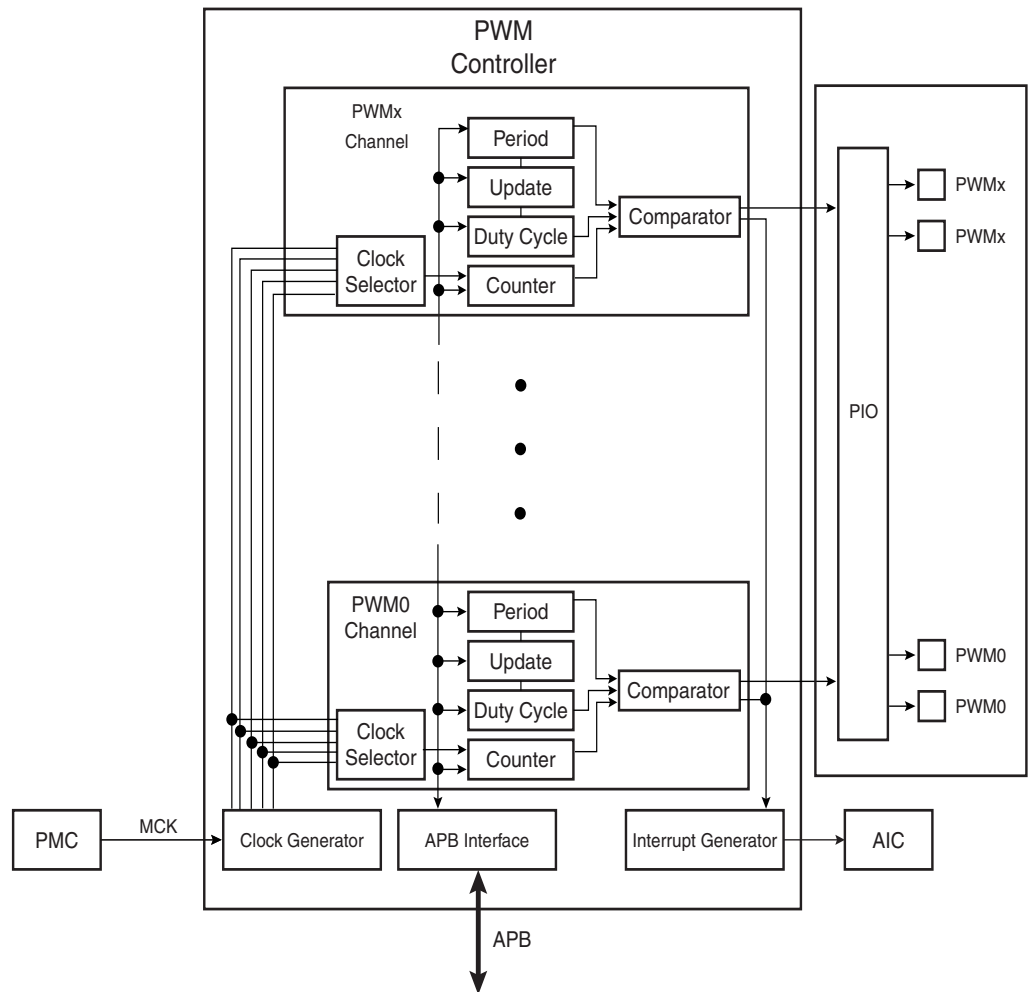
The PWM macrocell controls several channels independently. Each channel controls one square output waveform. Characteristics of the output waveform such as period, duty-cycle and polarity are configurable through the user interface. Each channel selects and uses one of the clocks provided by the clock 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.

### 32.2 Block Diagram

Figure 32-1. Pulse Width Modulation Controller Block Diagram



### 32.3 I/O Lines Description

Each channel outputs one waveform on one external I/O line.

**Table 32-1.** I/O Line Description

Name	Description	Type
PWMx	PWM Waveform Output for channel x	Output

### 32.4 Product Dependencies

#### 32.4.1 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.

#### 32.4.2 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.

#### 32.4.3 Interrupt Sources

The PWM interrupt line is connected on one of the internal sources of the Advanced Interrupt Controller. Using the PWM interrupt requires the AIC to be programmed first. Note that it is not recommended to use the PWM interrupt line in edge sensitive mode.

### 32.5 Functional Description

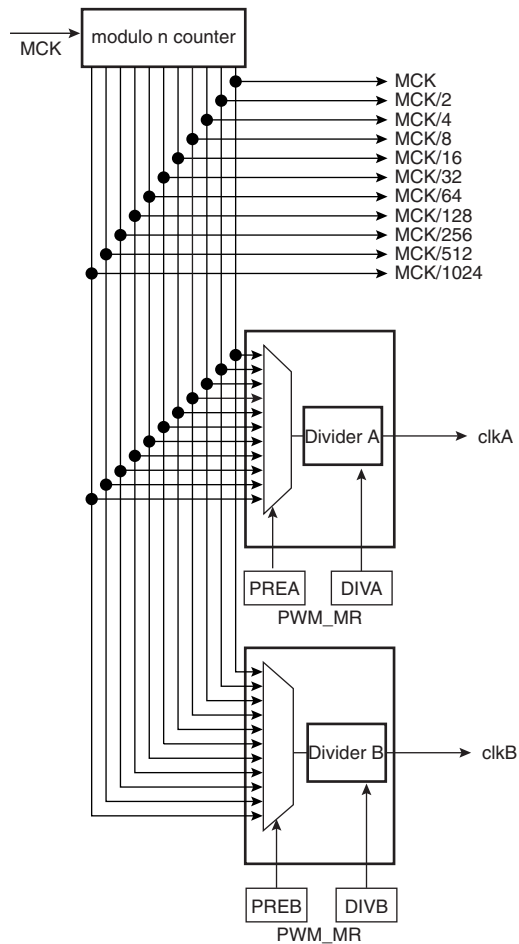
The PWM macrocell is primarily composed of a clock generator module and 8 channels.

- 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.



## 32.5.1 PWM Clock Generator

Figure 32-2. 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).

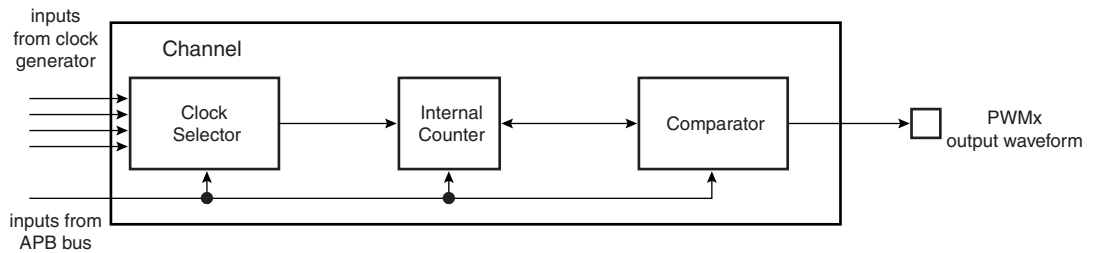
After a reset of the PWM controller, DIVA (DIVB) and PREA (PREB) in the PWM Mode register are set to 0. This implies that after reset clkA (clkB) are turned off.

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.

## 32.5.2 PWM Channel

### 32.5.2.1 Block Diagram

**Figure 32-3.** Functional View of the Channel Block Diagram



Each of the 8 channels is composed of three blocks:

- A clock selector which selects one of the clocks provided by the clock generator described in [Section 32.5.1 "PWM Clock Generator" on page 393](#).
- 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 20 bits.
- A comparator used to generate events according to the internal counter value. It also computes the PWMx output waveform according to the configuration.

### 32.5.2.2 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:

$$\frac{(CPRD \times DIVA)}{MCK} \text{ or } \frac{(CPRD \times DIVAB)}{MCK}$$

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:

$$\frac{(2 \times CPRD \times DIVA)}{MCK} \text{ or } \frac{(2 \times CPRD \times DIVB)}{MCK}$$

- 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:

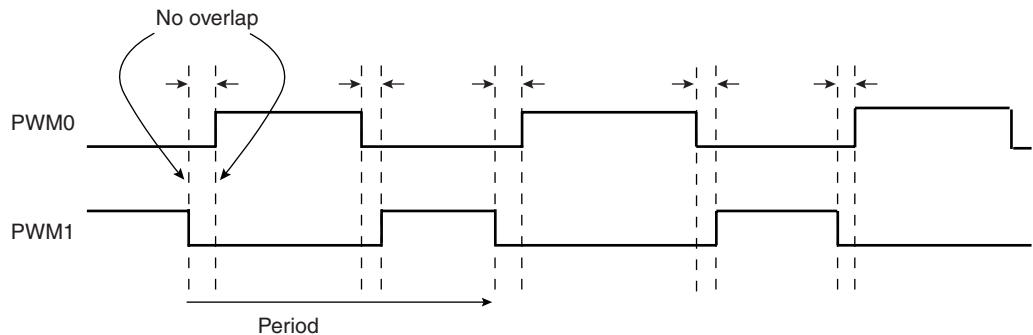
$$\text{duty cycle} = \frac{(\text{period} - 1 / \text{fchannel\_x\_clock} \times CDTY)}{\text{period}}$$

If the waveform is center aligned, then:

$$\text{duty cycle} = \frac{((\text{period}/2) - 1 / \text{fchannel\_x\_clock} \times CDTY)}{(\text{period}/2)}$$

- 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 32-4.** Non Overlapped Center Aligned Waveforms



Note: 1. See [Figure 32-5 on page 397](#) 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:

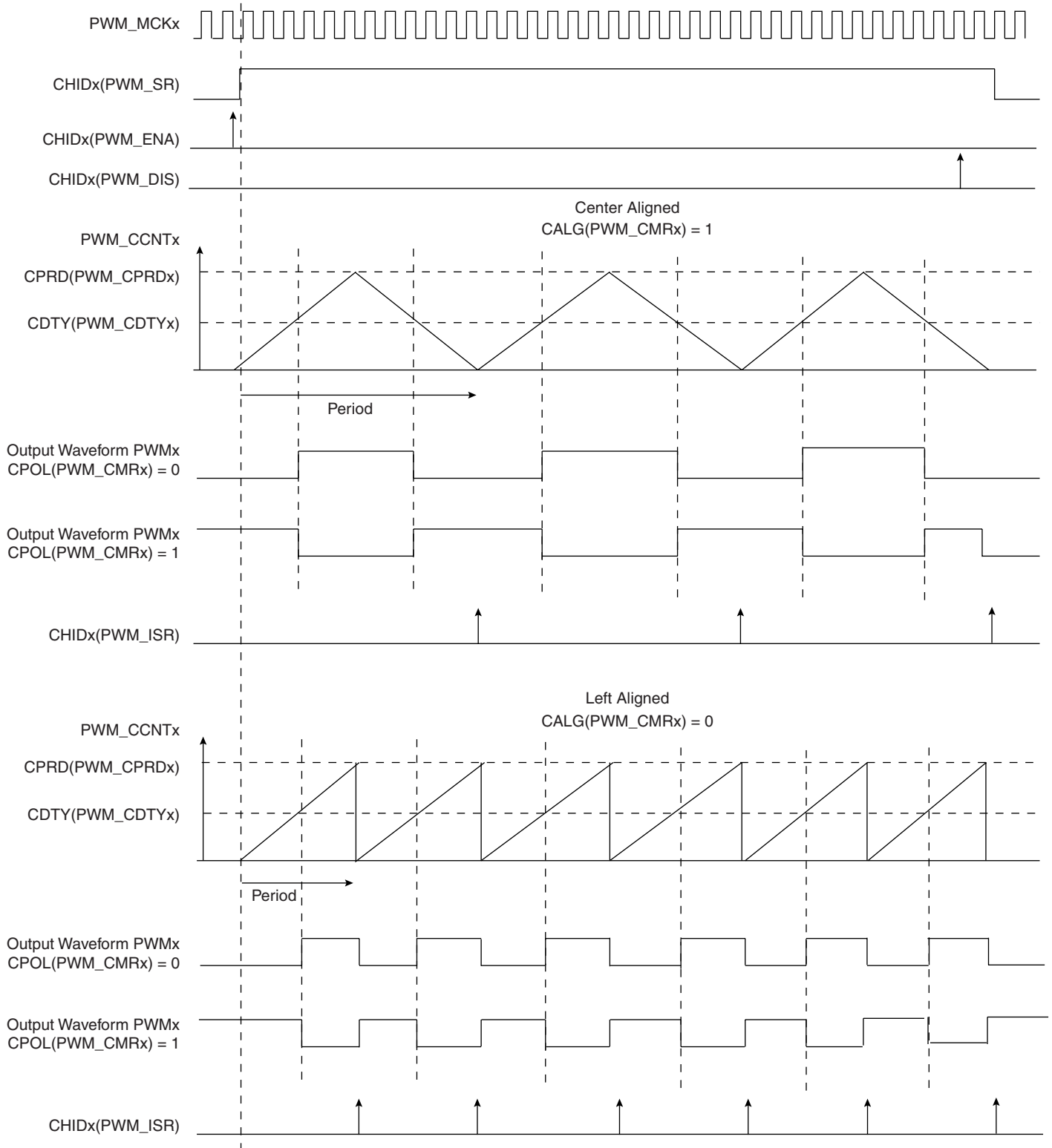
- CDTY = CPRD and CPOL = 0
- CDTY = 0 and CPOL = 1

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 32-5. Waveform Properties**



### 32.5.3 PWM Controller Operations

#### 32.5.3.1 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.

#### 32.5.3.2 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.

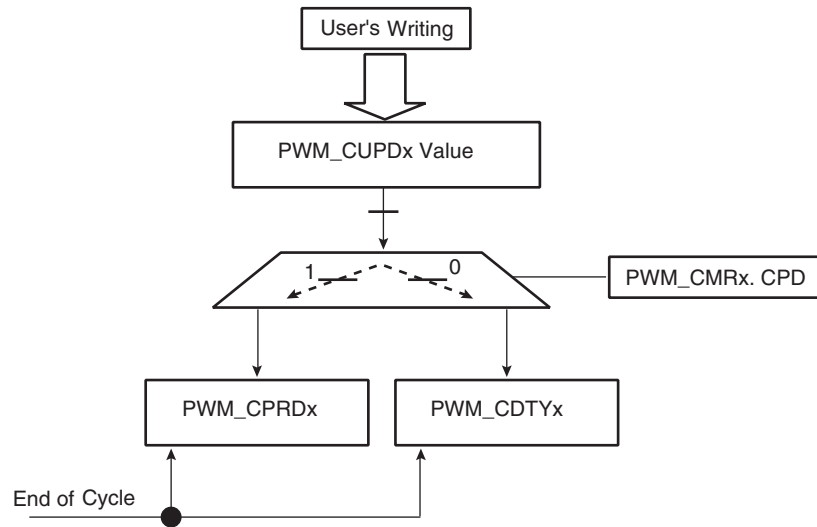
For example, if the user sets 15 (in decimal) in PWM\_CPRDx, the user is able to set a value between 1 up to 14 in PWM\_CDTYx Register. The resulting duty cycle quantum cannot be lower than 1/15 of the PWM period.

#### 32.5.3.3 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 32-6.** Synchronized Period or Duty Cycle Update



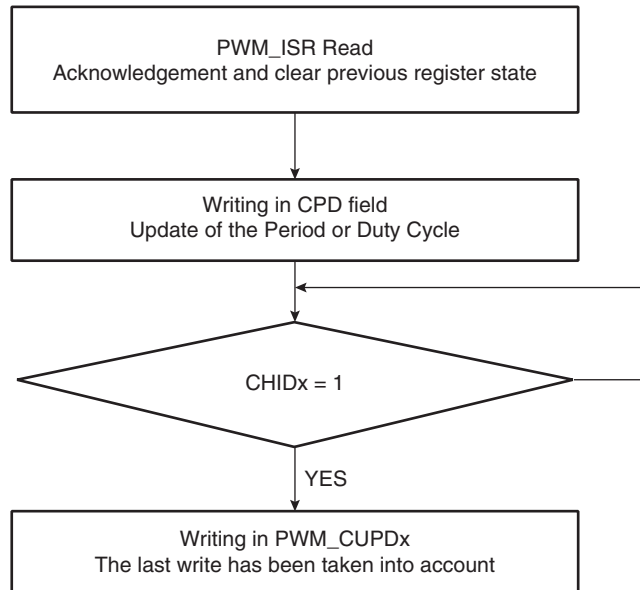
To prevent overwriting the PWM\_CUPDx by software, the user can use status events in order to synchronize his software. Two methods are possible. In both, the user must enable the dedicated interrupt in PWM\_IER at PWM Controller level.

The first method (polling method) consists of reading the relevant status bit in PWM\_ISR Register according to the enabled channel(s). See [Figure 32-7](#).

The second method uses an Interrupt Service Routine associated with the PWM channel.

Note: Reading the PWM\_ISR register automatically clears CHIDx flags.

**Figure 32-7.** Polling Method



Note: Polarity and alignment can be modified only when the channel is disabled.

#### 32.5.3.4 *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.



## 32.6 Pulse Width Modulation (PWM) Controller User Interface

**Table 32-2.** PWM Controller Registers

Offset	Register	Name	Access	Peripheral Reset Value
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
0x4C - 0xFC	Reserved	-	-	-
0x100 - 0x1FC	Reserved			
0x200	Channel 0 Mode Register	PWM_CMR0	Read/Write	0x0
0x204	Channel 0 Duty Cycle Register	PWM_CDTY0	Read/Write	0x0
0x208	Channel 0 Period Register	PWM_CPRD0	Read/Write	0x0
0x20C	Channel 0 Counter Register	PWM_CCNT0	Read-only	0x0
0x210	Channel 0 Update Register	PWM_CUPD0	Write-only	-
...	Reserved			
0x220	Channel 1 Mode Register	PWM_CMR1	Read/Write	0x0
0x224	Channel 1 Duty Cycle Register	PWM_CDTY1	Read/Write	0x0
0x228	Channel 1 Period Register	PWM_CPRD1	Read/Write	0x0
0x22C	Channel 1 Counter Register	PWM_CCNT1	Read-only	0x0
0x230	Channel 1 Update Register	PWM_CUPD1	Write-only	-
...	...	...	...	...



### 32.6.1 PWM Mode Register

Register Name: PWM\_MR  
Access Type: 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

DIVA, DIVB	CLKA, CLKB
0	CLKA, CLKB clock is turned off
1	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

PREA, PREB				Divider Input Clock
0	0	0	0	MCK.
0	0	0	1	MCK/2
0	0	1	0	MCK/4
0	0	1	1	MCK/8
0	1	0	0	MCK/16
0	1	0	1	MCK/32
0	1	1	0	MCK/64
0	1	1	1	MCK/128
1	0	0	0	MCK/256
1	0	0	1	MCK/512
1	0	1	0	MCK/1024
Other				Reserved

## 32.6.2 PWM Enable Register

**Register Name:** PWM\_ENA

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
CHID7	CHID6	CHID5	CHID4	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Channel ID**

0 = No effect.

1 = Enable PWM output for channel x.

## 32.6.3 PWM Disable Register

**Register Name:** PWM\_DIS

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
CHID7	CHID6	CHID5	CHID4	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Channel ID**

0 = No effect.

1 = Disable PWM output for channel x.

### 32.6.4 PWM Status Register

**Register Name:** PWM\_SR

**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
CHID7	CHID6	CHID5	CHID4	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Channel ID**

0 = PWM output for channel x is disabled.

1 = PWM output for channel x is enabled.

## 32.6.5 PWM Interrupt Enable Register

**Register Name:** PWM\_IER

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
CHID7	CHID6	CHID5	CHID4	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Channel ID.**

0 = No effect.

1 = Enable interrupt for PWM channel x.

## 32.6.6 PWM Interrupt Disable Register

**Register Name:** PWM\_IDR

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
CHID7	CHID6	CHID5	CHID4	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Channel ID.**

0 = No effect.

1 = Disable interrupt for PWM channel x.

### 32.6.7 PWM Interrupt Mask Register

**Register Name:** PWM\_IMR

**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
CHID7	CHID6	CHID5	CHID4	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Channel ID.**

0 = Interrupt for PWM channel x is disabled.

1 = Interrupt for PWM channel x is enabled.

### 32.6.8 PWM Interrupt Status Register

**Register Name:** PWM\_ISR

**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
CHID7	CHID6	CHID5	CHID4	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.

## 32.6.9 PWM Channel Mode Register

**Register Name:** PWM\_CMRx

**Access Type:** Read/Write

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

- **CPRE: Channel Pre-scaler**

CPRE				Channel Pre-scaler
0	0	0	0	MCK
0	0	0	1	MCK/2
0	0	1	0	MCK/4
0	0	1	1	MCK/8
0	1	0	0	MCK/16
0	1	0	1	MCK/32
0	1	1	0	MCK/64
0	1	1	1	MCK/128
1	0	0	0	MCK/256
1	0	0	1	MCK/512
1	0	1	0	MCK/1024
1	0	1	1	CLKA
1	1	0	0	CLKB
Other				Reserved

- **CALG: Channel Alignment**

0 = The period is left aligned.

1 = The period is center aligned.

- **CPOL: Channel Polarity**

0 = The output waveform starts at a low level.

1 = The output waveform starts at a high level.

- **CPD: Channel Update Period**

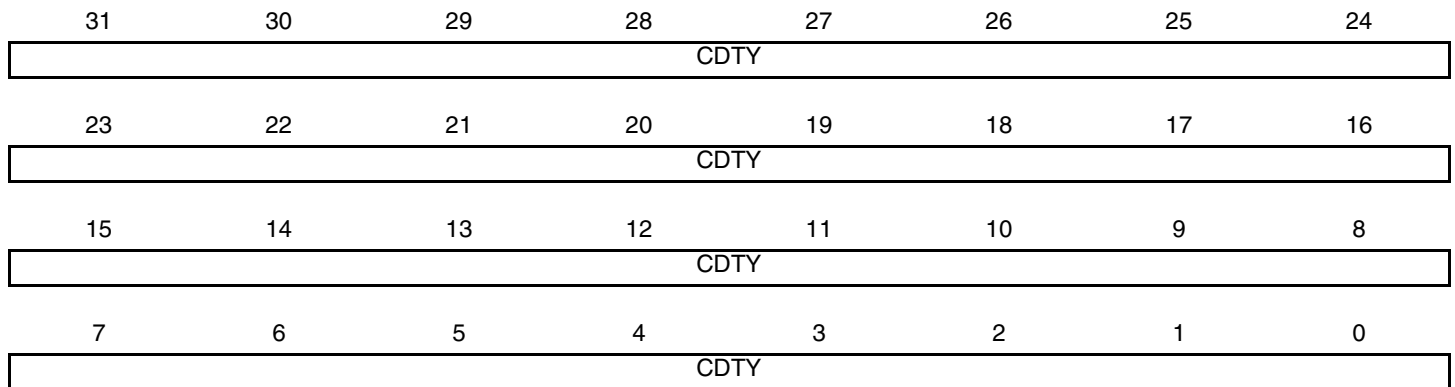
0 = Writing to the PWM\_CUPDx will modify the duty cycle at the next period start event.

1 = Writing to the PWM\_CUPDx will modify the period at the next period start event.

### 32.6.10 PWM Channel Duty Cycle Register

**Register Name:** PWM\_CDTYx

**Access Type:** Read/Write



Only the first 20 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).



## 32.6.11 PWM Channel Period Register

**Register Name:** PWM\_CPRDx

**Access Type:** 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 20 bits (internal channel counter size) are significant.

- **CPRD: Channel Period**

If the waveform is left-aligned, then the output waveform period depends on the counter source clock and can be calculated:

- By using the Master Clock (MCK) divided by an X given prescaler value (with X being 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024). The resulting period formula will be:

$$\frac{(X \times CPRD)}{MCK}$$

- By using a Master Clock divided by one of both DIVA or DIVB divider, the formula becomes, respectively:

$$\frac{(CPRD \times DIVA)}{MCK} \text{ or } \frac{(CPRD \times DIVB)}{MCK}$$

If the waveform is center-aligned, then the output waveform period depends on the counter source clock and can be calculated:

- By using the Master Clock (MCK) divided by an X given prescaler value (with X being 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, or 1024). The resulting period formula will be:

$$\frac{(2 \times X \times CPRD)}{MCK}$$

- By using a Master Clock divided by one of both DIVA or DIVB divider, the formula becomes, respectively:

$$\frac{(2 \times CPRD \times DIVA)}{MCK} \text{ or } \frac{(2 \times CPRD \times DIVB)}{MCK}$$

### 32.6.12 PWM Channel Counter Register

**Register Name:** PWM\_CCNTx

**Access Type:** 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.

### 32.6.13 PWM Channel Update Register

**Register Name:** PWM\_CUPDx

**Access Type:** 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							

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 20 bits (internal channel counter size) are significant.

CPD (PWM_CMRx Register)	
0	The duty-cycle (CDTC in the PWM_CDRx register) is updated with the CUPD value at the beginning of the next period.
1	The period (CPRD in the PWM_CPRx register) is updated with the CUPD value at the beginning of the next period.

## 33. USB Device Port (UDP)

### 33.1 Overview

The USB Device Port (UDP) is compliant with the Universal Serial Bus (USB) V2.0 full-speed device specification.

Each endpoint can be configured in one of several USB transfer types. It can be associated with one or two banks of a dual-port RAM used to store the current data payload. If two banks are used, one DPR bank is read or written by the processor, while the other is read or written by the USB device peripheral. This feature is mandatory for isochronous endpoints. Thus the device maintains the maximum bandwidth (1M bytes/s) by working with endpoints with two banks of DPR.

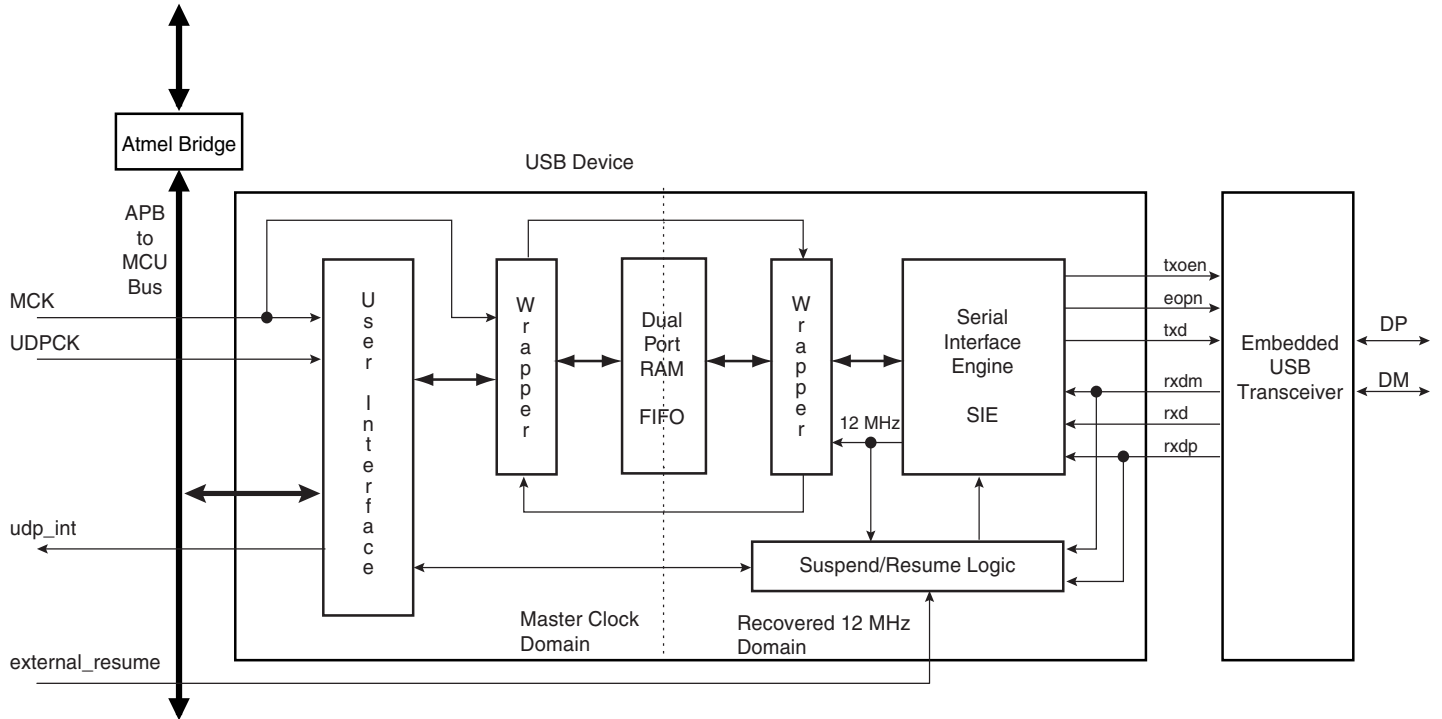
**Table 33-1.** USB Endpoint Description

Endpoint Number	Mnemonic	Dual-Bank	Max. Endpoint Size	Endpoint Type
0	EP0	No	8	Control/Bulk/Interrupt
1	EP1	Yes	64	Bulk/Iso/Interrupt
2	EP2	Yes	64	Bulk/Iso/Interrupt
3	EP3	No	64	Control/Bulk/Interrupt
4	EP4	Yes	512	Bulk/Iso/Interrupt
5	EP5	Yes	512	Bulk/Iso/Interrupt

Suspend and resume are automatically detected by the USB device, which notifies the processor by raising an interrupt. Depending on the product, an external signal can be used to send a wake up to the USB host controller.

### 33.2 Block Diagram

Figure 33-1. Block Diagram



Access to the UDP is via the APB bus interface. Read and write to the data FIFO are done by reading and writing 8-bit values to APB registers.

The UDP peripheral requires two clocks: one peripheral clock used by the MCK domain and a 48 MHz clock used by the 12 MHz domain.

A USB 2.0 full-speed pad is embedded and controlled by the Serial Interface Engine (SIE).

The signal external\_resume is optional. It allows the UDP peripheral to wake up once in system mode. The host is then notified that the device asks for a resume. This optional feature must be also negotiated with the host during the enumeration.

## 33.3 Product Dependencies

For further details on the USB Device hardware implementation, see the specific Product Properties document.

The USB physical transceiver is integrated into the product. The bidirectional differential signals DP and DM are available from the product boundary.

Two I/O lines may be used by the application:

- One to check that VBUS is still available from the host. Self-powered devices may use this entry to be notified that the host has been powered off. In this case, the board pullup on DP must be disabled in order to prevent feeding current to the host.
- One to control the board pullup on DP. Thus, when the device is ready to communicate with the host, it activates its DP pullup through this control line.

### 33.3.1 I/O Lines

DP and DM are not controlled by any PIO controllers. The embedded USB physical transceiver is controlled by the USB device peripheral.

To reserve an I/O line to check VBUS, the programmer must first program the PIO controller to assign this I/O in input PIO mode.

To reserve an I/O line to control the board pullup, the programmer must first program the PIO controller to assign this I/O in output PIO mode.

### 33.3.2 Power Management

The USB device peripheral requires a 48 MHz clock. This clock must be generated by a PLL with an accuracy of  $\pm 0.25\%$ .

Thus, the USB device receives two clocks from the Power Management Controller (PMC): the master clock, MCK, used to drive the peripheral user interface, and the UDPCK, used to interface with the bus USB signals (recovered 12 MHz domain).

**WARNING:** The UDP peripheral clock in the Power Management Controller (PMC) must be enabled before any read/write operations to the UDP registers including the UDP\_TXCV register.

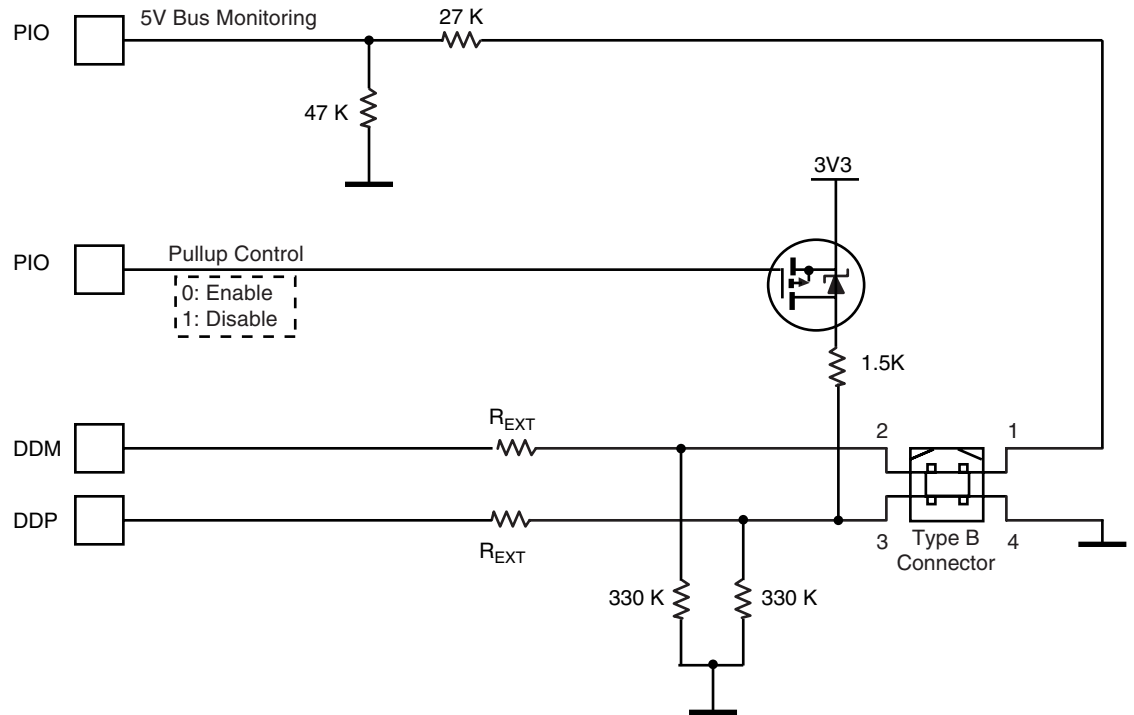
### 33.3.3 Interrupt

The USB device interface has an interrupt line connected to the Advanced Interrupt Controller (AIC).

Handling the USB device interrupt requires programming the AIC before configuring the UDP.

### 33.4 Typical Connection

**Figure 33-2.** Board Schematic to Interface USB Device Peripheral



#### 33.4.1 USB Device Transceiver

The USB device transceiver is embedded in the product. A few discrete components are required as follows:

- the application detects all device states as defined in chapter 9 of the USB specification;
  - pullup enable/disable
  - VBUS monitoring
- to reduce power consumption the host is disconnected
- for line termination.

Pullup enable/disable is done through a MOSFET controlled by a PIO. The pullup is enabled when the PIO drives a 0. Thus PIO default state to 1 corresponds to a pullup disable. Once the pullup is enabled, the host will force a device reset 100 ms later. Bus powered devices must connect the pullup within 100 ms.

#### 33.4.2 VBUS Monitoring

VBUS monitoring is required to detect host connection. VBUS monitoring is done using a standard PIO with internal pullup disabled. When the host is switched off, it should be considered as a disconnect, the pullup must be disabled in order to prevent powering the host through the pull-up resistor.

When the host is disconnected and the transceiver is enabled, then DDP and DDM are floating. This may lead to over consumption. A solution is to connect 330 K $\Omega$  pulldowns on DP and DM. These pulldowns do not alter DDP and DDM signal integrity.

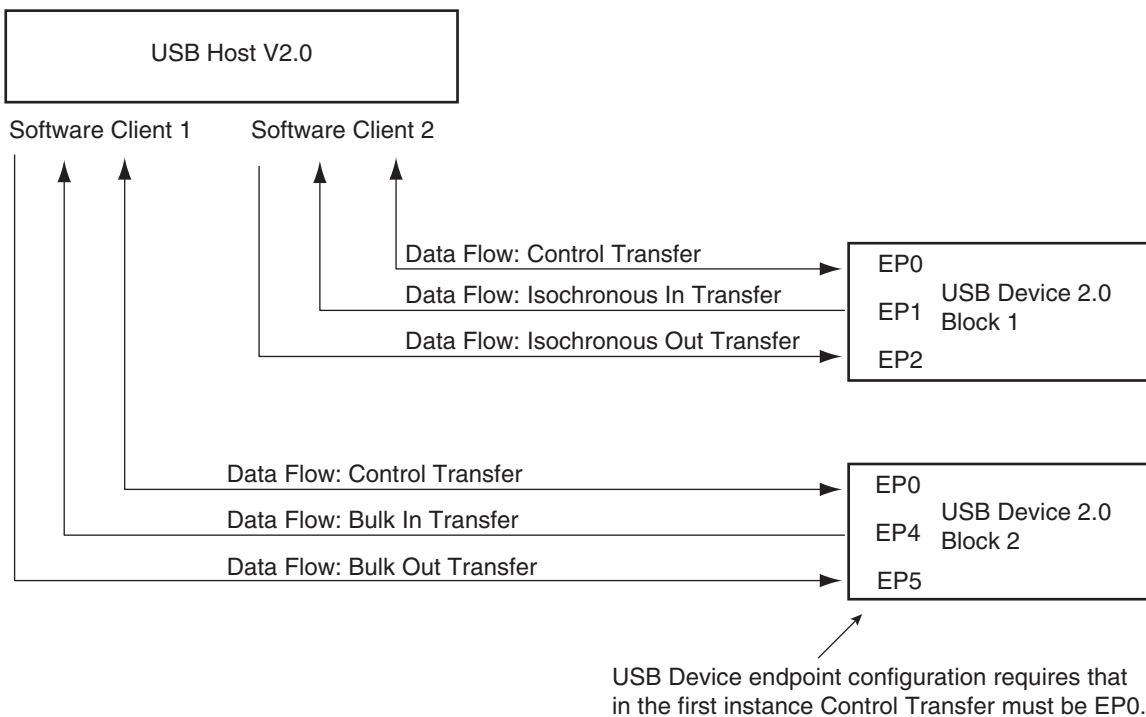
A termination serial resistor must be connected to DP and DM. The resistor value is defined in the electrical specification of the product ( $R_{EXT}$ ).

## 33.5 Functional Description

### 33.5.1 USB V2.0 Full-speed Introduction

The USB V2.0 full-speed provides communication services between host and attached USB devices. Each device is offered with a collection of communication flows (pipes) associated with each endpoint. Software on the host communicates with a USB device through a set of communication flows.

**Figure 33-3.** Example of USB V2.0 Full-speed Communication Control



The Control Transfer endpoint EP0 is always used when a USB device is first configured (USB v. 2.0 specifications).

#### 33.5.1.1 USB V2.0 Full-speed Transfer Types

A communication flow is carried over one of four transfer types defined by the USB device.

**Table 33-2.** USB Communication Flow

Transfer	Direction	Bandwidth	Supported Endpoint Size	Error Detection	Retrying
Control	Bidirectional	Not guaranteed	8, 16, 32, 64	Yes	Automatic
Isochronous	Unidirectional	Guaranteed	512	Yes	No
Interrupt	Unidirectional	Not guaranteed	8, 16, 32, 64	Yes	Yes
Bulk	Unidirectional	Not guaranteed	8, 16, 32, 64	Yes	Yes

### 33.5.1.2 USB Bus Transactions

Each transfer results in one or more transactions over the USB bus. There are three kinds of transactions flowing across the bus in packets:

1. Setup Transaction
2. Data IN Transaction
3. Data OUT Transaction

### 33.5.1.3 USB Transfer Event Definitions

As indicated below, transfers are sequential events carried out on the USB bus.

**Table 33-3.** USB Transfer Events

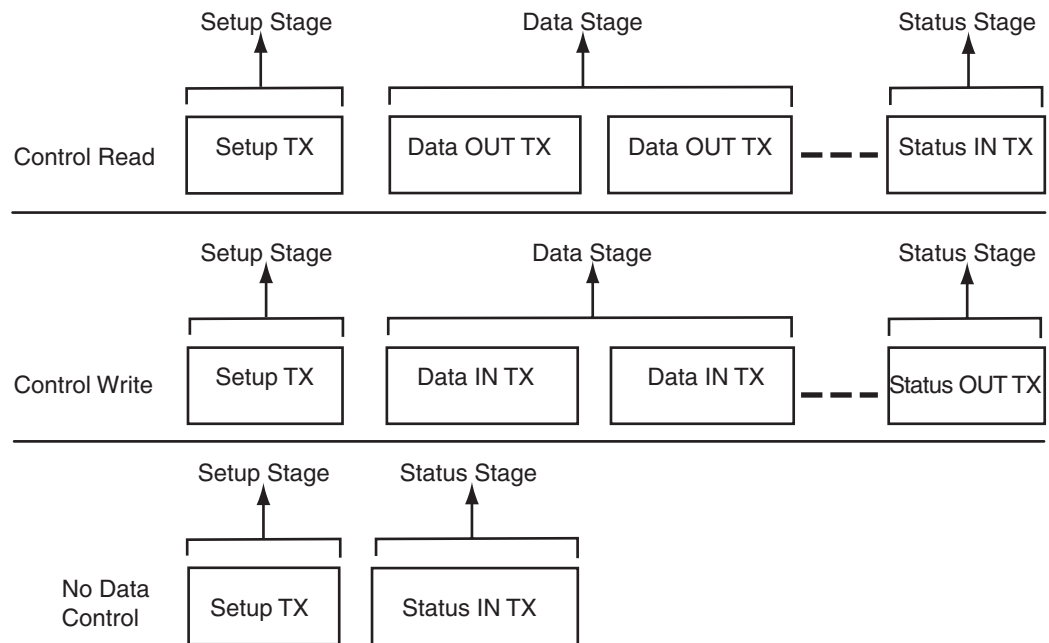
Control Transfers <sup>(1) (3)</sup>	<ul style="list-style-type: none"> <li>• Setup transaction &gt; Data IN transactions &gt; Status OUT transaction</li> <li>• Setup transaction &gt; Data OUT transactions &gt; Status IN transaction</li> <li>• Setup transaction &gt; Status IN transaction</li> </ul>
Interrupt IN Transfer (device toward host)	<ul style="list-style-type: none"> <li>• Data IN transaction &gt; Data IN transaction</li> </ul>
Interrupt OUT Transfer (host toward device)	<ul style="list-style-type: none"> <li>• Data OUT transaction &gt; Data OUT transaction</li> </ul>
Isochronous IN Transfer <sup>(2)</sup> (device toward host)	<ul style="list-style-type: none"> <li>• Data IN transaction &gt; Data IN transaction</li> </ul>
Isochronous OUT Transfer <sup>(2)</sup> (host toward device)	<ul style="list-style-type: none"> <li>• Data OUT transaction &gt; Data OUT transaction</li> </ul>
Bulk IN Transfer (device toward host)	<ul style="list-style-type: none"> <li>• Data IN transaction &gt; Data IN transaction</li> </ul>
Bulk OUT Transfer (host toward device)	<ul style="list-style-type: none"> <li>• Data OUT transaction &gt; Data OUT transaction</li> </ul>

Notes: 1. Control transfer must use endpoints with no ping-pong attributes.  
 2. Isochronous transfers must use endpoints with ping-pong attributes.  
 3. Control transfers can be aborted using a stall handshake.

A status transaction is a special type of host-to-device transaction used only in a control transfer. The control transfer must be performed using endpoints with no ping-pong attributes. According to the control sequence (read or write), the USB device sends or receives a status transaction.



**Figure 33-4.** Control Read and Write Sequences



- Notes:
1. During the Status IN stage, the host waits for a zero length packet (Data IN transaction with no data) from the device using DATA1 PID. Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev. 2.0*, for more information on the protocol layer.
  2. During the Status OUT stage, the host emits a zero length packet to the device (Data OUT transaction with no data).

## 33.5.2 Handling Transactions with USB V2.0 Device Peripheral

### 33.5.2.1 Setup Transaction

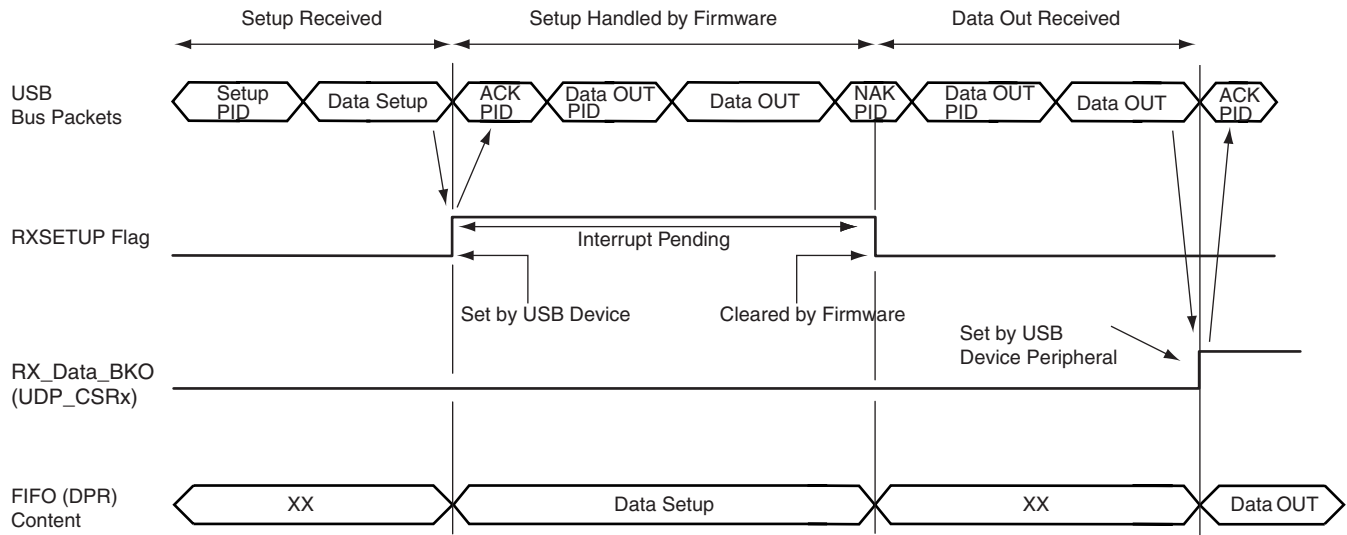
Setup is a special type of host-to-device transaction used during control transfers. Control transfers must be performed using endpoints with no ping-pong attributes. A setup transaction needs to be handled as soon as possible by the firmware. It is used to transmit requests from the host to the device. These requests are then handled by the USB device and may require more arguments. The arguments are sent to the device by a Data OUT transaction which follows the setup transaction. These requests may also return data. The data is carried out to the host by the next Data IN transaction which follows the setup transaction. A status transaction ends the control transfer.

When a setup transfer is received by the USB endpoint:

- The USB device automatically acknowledges the setup packet □
- RXSETUP is set in the UDP\_CSRx register □
- An endpoint interrupt is generated while the RXSETUP is not cleared. This interrupt is carried out to the microcontroller if interrupts are enabled for this endpoint.

Thus, firmware must detect the RXSETUP polling the UDP\_CSRx or catching an interrupt, read the setup packet in the FIFO, then clear the RXSETUP. RXSETUP cannot be cleared before the setup packet has been read in the FIFO. Otherwise, the USB device would accept the next Data OUT transfer and overwrite the setup packet in the FIFO.

**Figure 33-5.** Setup Transaction Followed by a Data OUT Transaction



### 33.5.2.2 Data IN Transaction

Data IN transactions are used in control, isochronous, bulk and interrupt transfers and conduct the transfer of data from the device to the host. Data IN transactions in isochronous transfer must be done using endpoints with ping-pong attributes.

### 33.5.2.3 Using Endpoints Without Ping-pong Attributes

To perform a Data IN transaction using a non ping-pong endpoint:

1. The application checks if it is possible to write in the FIFO by polling TXPKTRDY in the endpoint's UDP\_CSRx register (TXPKTRDY must be cleared).
2. The application writes the first packet of data to be sent in the endpoint's FIFO, writing zero or more byte values in the endpoint's UDP\_FDRx register,
3. The application notifies the USB peripheral it has finished by setting the TXPKTRDY in the endpoint's UDP\_CSRx register.
4. The application is notified that the endpoint's FIFO has been released by the USB device when TXCOMP in the endpoint's UDP\_CSRx register has been set. Then an interrupt for the corresponding endpoint is pending while TXCOMP is set.
5. The microcontroller writes the second packet of data to be sent in the endpoint's FIFO, writing zero or more byte values in the endpoint's UDP\_FDRx register,
6. The microcontroller notifies the USB peripheral it has finished by setting the TXPKTRDY in the endpoint's UDP\_CSRx register.
7. The application clears the TXCOMP in the endpoint's UDP\_CSRx.

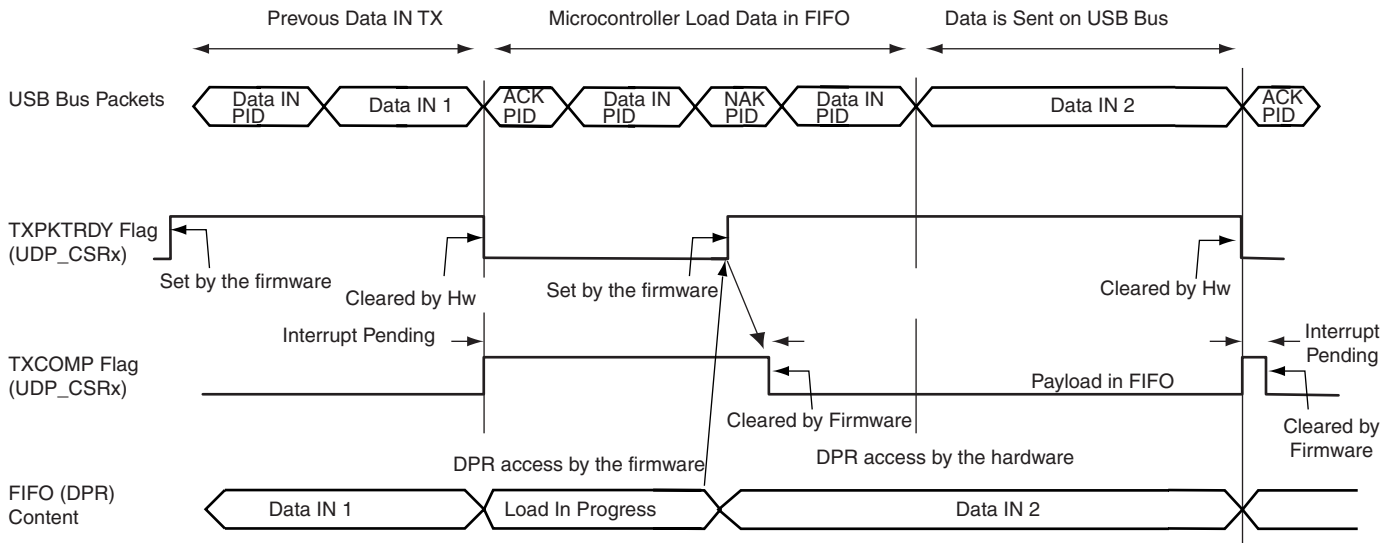
After the last packet has been sent, the application must clear TXCOMP once this has been set.

TXCOMP is set by the USB device when it has received an ACK PID signal for the Data IN packet. An interrupt is pending while TXCOMP is set.

**Warning:** TX\_COMP must be cleared after TX\_PKTRDY has been set.

Note: Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev 2.0*, for more information on the Data IN protocol layer.

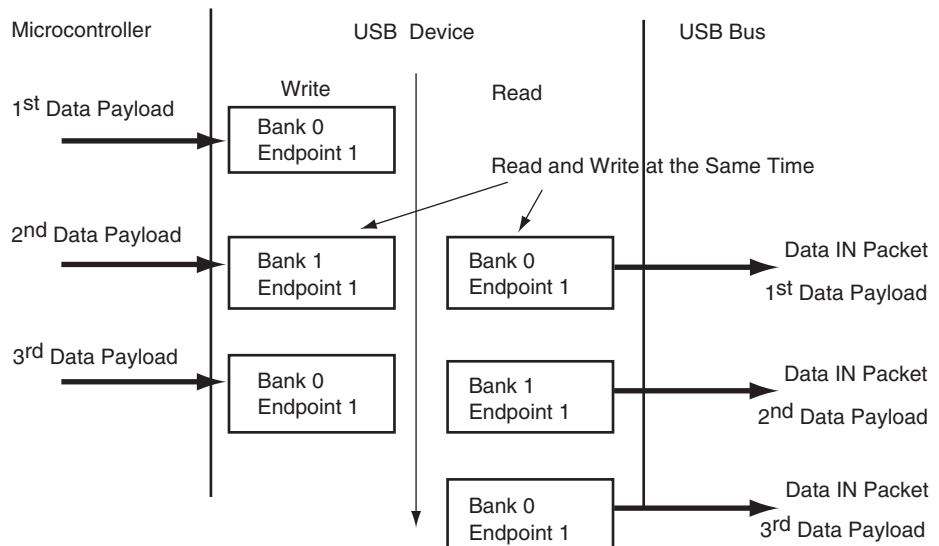
**Figure 33-6.** Data IN Transfer for Non Ping-pong Endpoint



### 33.5.2.4 Using Endpoints With Ping-pong Attribute

The use of an endpoint with ping-pong attributes is necessary during isochronous transfer. This also allows handling the maximum bandwidth defined in the USB specification during bulk transfer. To be able to guarantee a constant or the maximum bandwidth, the microcontroller must prepare the next data payload to be sent while the current one is being sent by the USB device. Thus two banks of memory are used. While one is available for the microcontroller, the other one is locked by the USB device.

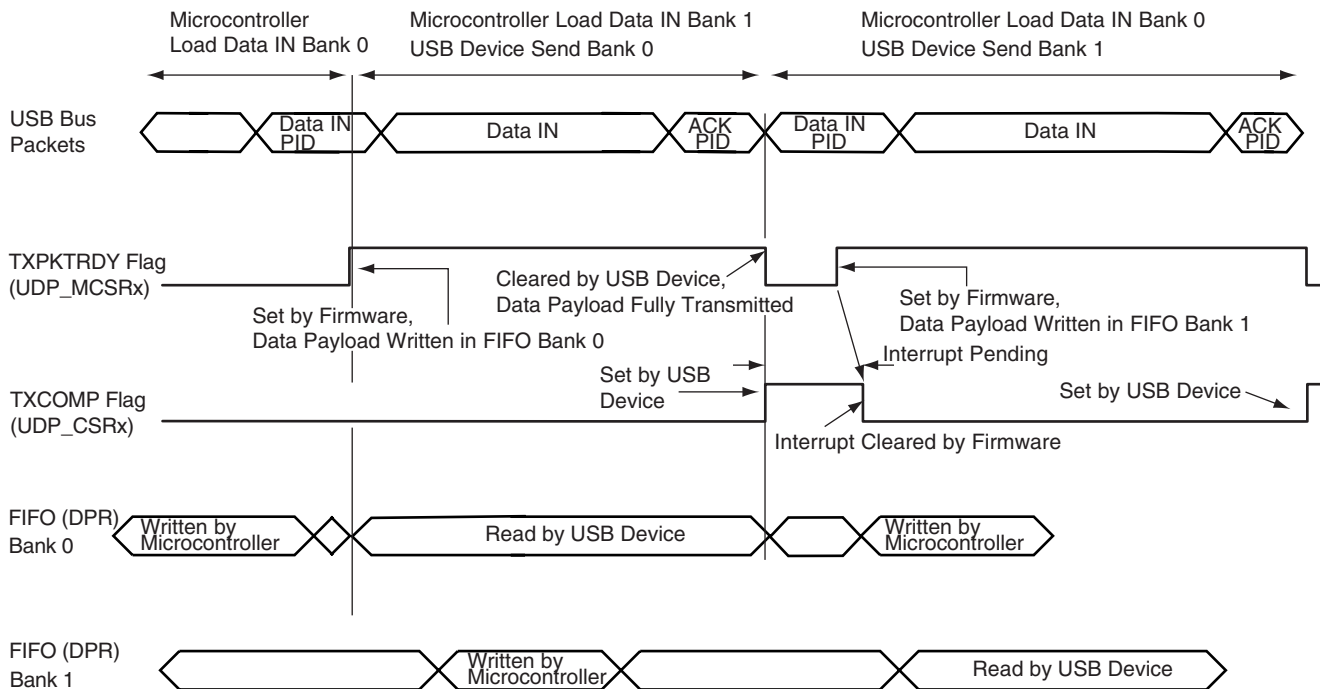
**Figure 33-7.** Bank Swapping Data IN Transfer for Ping-pong Endpoints



When using a ping-pong endpoint, the following procedures are required to perform Data IN transactions:

1. The microcontroller checks if it is possible to write in the FIFO by polling TXPKTRDY to be cleared in the endpoint's UDP\_CSRx register.
2. The microcontroller writes the first data payload to be sent in the FIFO (Bank 0), writing zero or more byte values in the endpoint's UDP\_FDRx register.
3. The microcontroller notifies the USB peripheral it has finished writing in Bank 0 of the FIFO by setting the TXPKTRDY in the endpoint's UDP\_CSRx register.
4. Without waiting for TXPKTRDY to be cleared, the microcontroller writes the second data payload to be sent in the FIFO (Bank 1), writing zero or more byte values in the endpoint's UDP\_FDRx register.
5. The microcontroller is notified that the first Bank has been released by the USB device when TXCOMP in the endpoint's UDP\_CSRx register is set. An interrupt is pending while TXCOMP is being set.
6. Once the microcontroller has received TXCOMP for the first Bank, it notifies the USB device that it has prepared the second Bank to be sent rising TXPKTRDY in the endpoint's UDP\_CSRx register.
7. At this step, Bank 0 is available and the microcontroller can prepare a third data payload to be sent.

**Figure 33-8.** Data IN Transfer for Ping-pong Endpoint



**Warning:** There is software critical path due to the fact that once the second bank is filled, the driver has to wait for TX\_COMP to set TX\_PKTRDY. If the delay between receiving TX\_COMP is set and TX\_PKTRDY is set is too long, some Data IN packets may be NACKed, reducing the bandwidth.

**Warning:** TX\_COMP must be cleared after TX\_PKTRDY has been set.

## 33.5.2.5 Data OUT Transaction

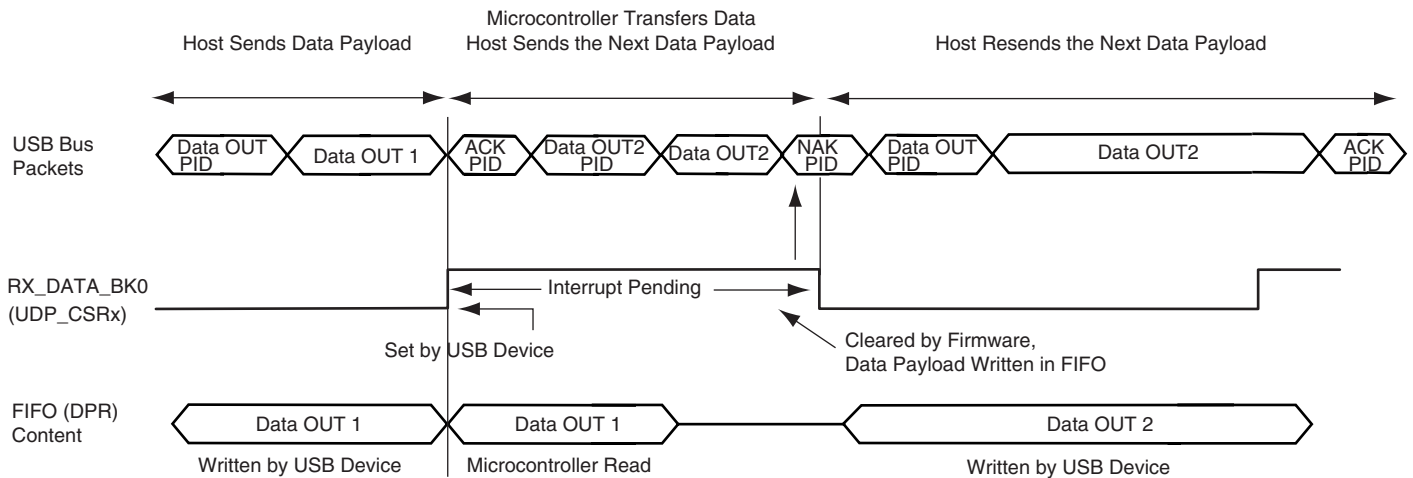
Data OUT transactions are used in control, isochronous, bulk and interrupt transfers and conduct the transfer of data from the host to the device. Data OUT transactions in isochronous transfers must be done using endpoints with ping-pong attributes.

## 33.5.2.6 Data OUT Transaction Without Ping-pong Attributes

To perform a Data OUT transaction, using a non ping-pong endpoint:

1. The host generates a Data OUT packet.
2. This packet is received by the USB device endpoint. While the FIFO associated to this endpoint is being used by the microcontroller, a NAK PID is returned to the host. Once the FIFO is available, data are written to the FIFO by the USB device and an ACK is automatically carried out to the host.
3. The microcontroller is notified that the USB device has received a data payload polling RX\_DATA\_BK0 in the endpoint's UDP\_CSRx register. An interrupt is pending for this endpoint while RX\_DATA\_BK0 is set.
4. The number of bytes available in the FIFO is made available by reading RXBYTECNT in the endpoint's UDP\_CSRx register.
5. The microcontroller carries out data received from the endpoint's memory to its memory. Data received is available by reading the endpoint's UDP\_FDRx register.
6. The microcontroller notifies the USB device that it has finished the transfer by clearing RX\_DATA\_BK0 in the endpoint's UDP\_CSRx register.
7. A new Data OUT packet can be accepted by the USB device.

**Figure 33-9.** Data OUT Transfer for Non Ping-pong Endpoints

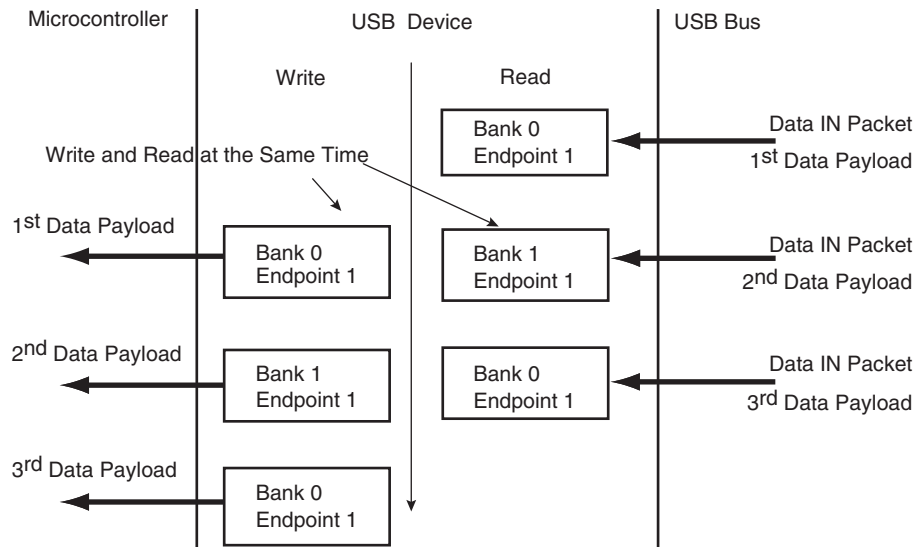


An interrupt is pending while the flag RX\_DATA\_BK0 is set. Memory transfer between the USB device, the FIFO and microcontroller memory can not be done after RX\_DATA\_BK0 has been cleared. Otherwise, the USB device would accept the next Data OUT transfer and overwrite the current Data OUT packet in the FIFO.

### 33.5.2.7 Using Endpoints With Ping-pong Attributes

During isochronous transfer, using an endpoint with ping-pong attributes is obligatory. To be able to guarantee a constant bandwidth, the microcontroller must read the previous data payload sent by the host, while the current data payload is received by the USB device. Thus two banks of memory are used. While one is available for the microcontroller, the other one is locked by the USB device.

**Figure 33-10.** Bank Swapping in Data OUT Transfers for Ping-pong Endpoints

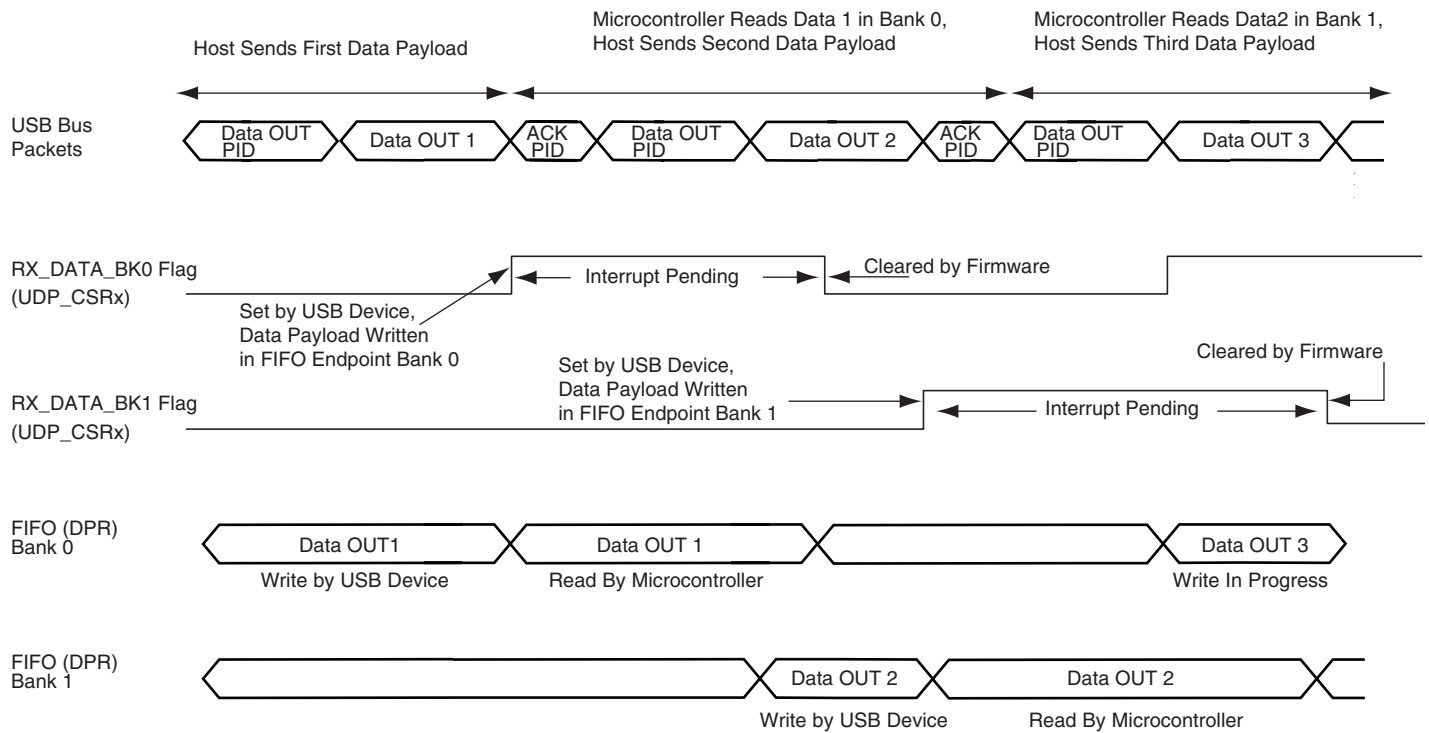


When using a ping-pong endpoint, the following procedures are required to perform Data OUT transactions:

1. The host generates a Data OUT packet.
2. This packet is received by the USB device endpoint. It is written in the endpoint's FIFO Bank 0.
3. The USB device sends an ACK PID packet to the host. The host can immediately send a second Data OUT packet. It is accepted by the device and copied to FIFO Bank 1.
4. The microcontroller is notified that the USB device has received a data payload, polling `RX_DATA_BK0` in the endpoint's `UDP_CSRx` register. An interrupt is pending for this endpoint while `RX_DATA_BK0` is set.
5. The number of bytes available in the FIFO is made available by reading `RXBYTECNT` in the endpoint's `UDP_CSRx` register.
6. The microcontroller transfers out data received from the endpoint's memory to the microcontroller's memory. Data received is made available by reading the endpoint's `UDP_FDRx` register.
7. The microcontroller notifies the USB peripheral device that it has finished the transfer by clearing `RX_DATA_BK0` in the endpoint's `UDP_CSRx` register.
8. A third Data OUT packet can be accepted by the USB peripheral device and copied in the FIFO Bank 0.
9. If a second Data OUT packet has been received, the microcontroller is notified by the flag `RX_DATA_BK1` set in the endpoint's `UDP_CSRx` register. An interrupt is pending for this endpoint while `RX_DATA_BK1` is set.

10. The microcontroller transfers out data received from the endpoint's memory to the microcontroller's memory. Data received is available by reading the endpoint's UDP\_FDRx register.
11. The microcontroller notifies the USB device it has finished the transfer by clearing RX\_DATA\_BK1 in the endpoint's UDP\_CSRx register.
12. A fourth Data OUT packet can be accepted by the USB device and copied in the FIFO Bank 0.

**Figure 33-11. Data OUT Transfer for Ping-pong Endpoint**



Note: An interrupt is pending while the RX\_DATA\_BK0 or RX\_DATA\_BK1 flag is set.

**Warning:** When RX\_DATA\_BK0 and RX\_DATA\_BK1 are both set, there is no way to determine which one to clear first. Thus the software must keep an internal counter to be sure to clear alternately RX\_DATA\_BK0 then RX\_DATA\_BK1. This situation may occur when the software application is busy elsewhere and the two banks are filled by the USB host. Once the application comes back to the USB driver, the two flags are set.

### 33.5.2.8 Stall Handshake

A stall handshake can be used in one of two distinct occasions. (For more information on the stall handshake, refer to Chapter 8 of the *Universal Serial Bus Specification, Rev 2.0*.)

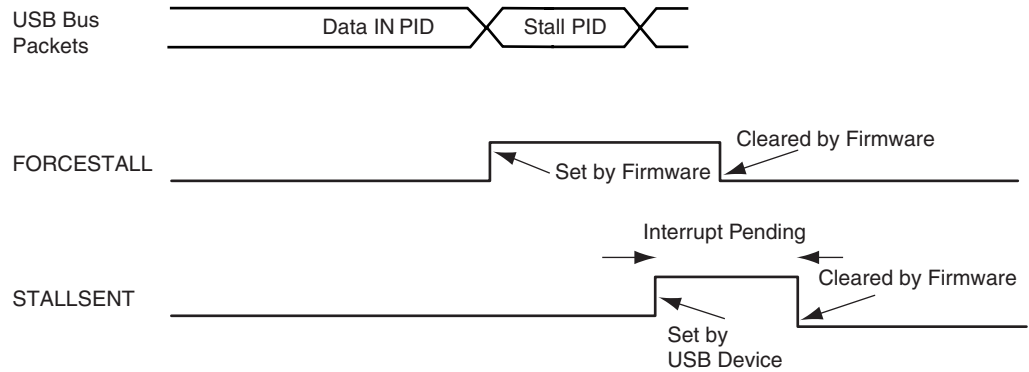
- A functional stall is used when the halt feature associated with the endpoint is set. (Refer to Chapter 9 of the *Universal Serial Bus Specification, Rev 2.0*, for more information on the halt feature.)
- To abort the current request, a protocol stall is used, but uniquely with control transfer.

The following procedure generates a stall packet:

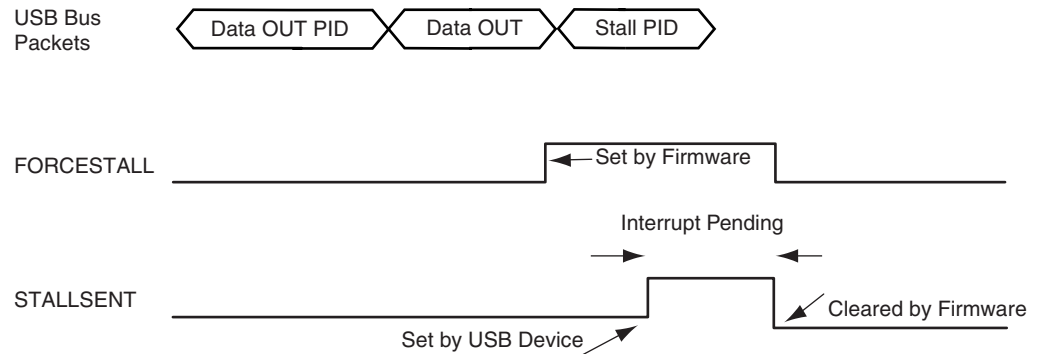
1. The microcontroller sets the FORCESTALL flag in the UDP\_ CSRx endpoint's register.
2. The host receives the stall packet.
3. The microcontroller is notified that the device has sent the stall by polling the STALLSENT to be set. An endpoint interrupt is pending while STALLSENT is set. The microcontroller must clear STALLSENT to clear the interrupt.

When a setup transaction is received after a stall handshake, STALLSENT must be cleared in order to prevent interrupts due to STALLSENT being set.

**Figure 33-12.** Stall Handshake (Data IN Transfer)



**Figure 33-13.** Stall Handshake (Data OUT Transfer)

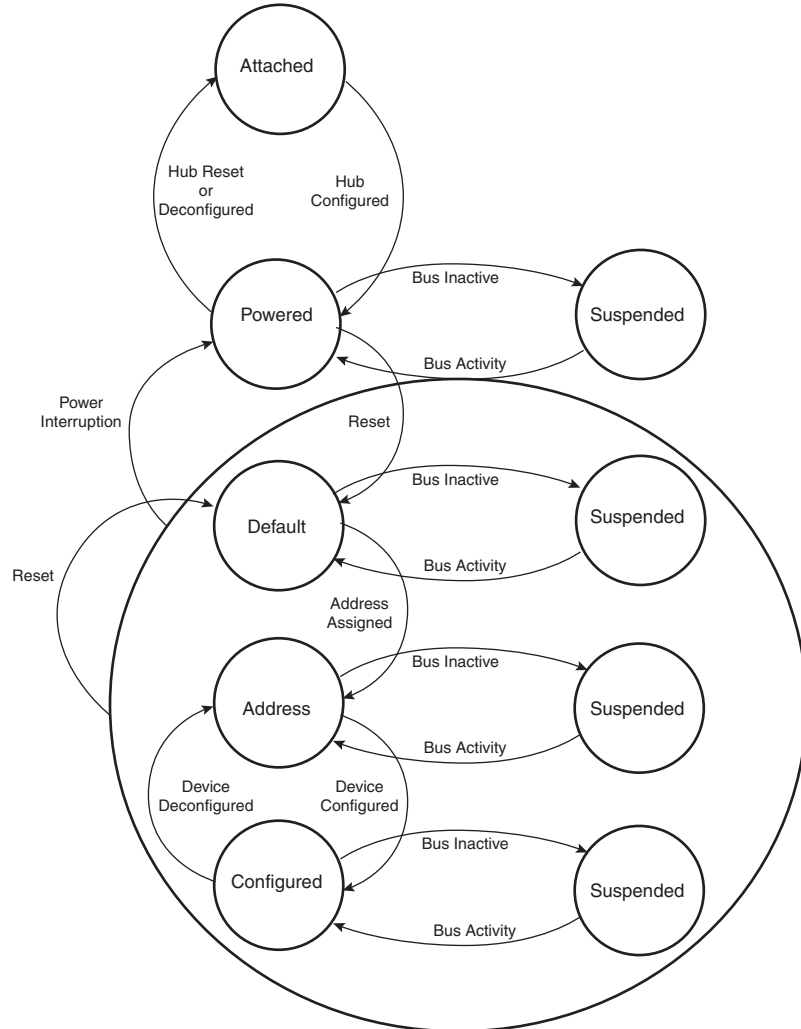




## 33.5.3 Controlling Device States

A USB device has several possible states. Refer to Chapter 9 of the *Universal Serial Bus Specification, Rev 2.0*.

**Figure 33-14.** USB Device State Diagram



Movement from one state to another depends on the USB bus state or on standard requests sent through control transactions via the default endpoint (endpoint 0).

After a period of bus inactivity, the USB device enters Suspend Mode. Accepting Suspend/Resume requests from the USB host is mandatory. Constraints in Suspend Mode are very strict for bus-powered applications; devices may not consume more than 500  $\mu\text{A}$  on the USB bus.

While in Suspend Mode, the host may wake up a device by sending a resume signal (bus activity) or a USB device may send a wake up request to the host, e.g., waking up a PC by moving a USB mouse.

The wake up feature is not mandatory for all devices and must be negotiated with the host.

### 33.5.3.1 *Not Powered State*

Self powered devices can detect 5V VBUS using a PIO as described in the typical connection section. When the device is not connected to a host, device power consumption can be reduced by disabling MCK for the UDP, disabling UDPCK and disabling the transceiver. DDP and DDM lines are pulled down by 330 K $\Omega$  resistors.

### 33.5.3.2 *Entering Attached State*

When no device is connected, the USB DP and DM signals are tied to GND by 15 K $\Omega$  pull-down resistors integrated in the hub downstream ports. When a device is attached to a hub downstream port, the device connects a 1.5 K $\Omega$  pull-up resistor on DP. The USB bus line goes into IDLE state, DP is pulled up by the device 1.5 K $\Omega$  resistor to 3.3V and DM is pulled down by the 15 K $\Omega$  resistor of the host.

After pullup connection, the device enters the powered state. In this state, the UDPCK and MCK must be enabled in the Power Management Controller. The transceiver can remain disabled.

### 33.5.3.3 *From Powered State to Default State*

After its connection to a USB host, the USB device waits for an end-of-bus reset. The unmaskable flag ENDBUSRES is set in the register UDP\_ISR and an interrupt is triggered.

Once the ENDBUSRES interrupt has been triggered, the device enters Default State. In this state, the UDP software must:

- Enable the default endpoint, setting the EPEDS flag in the UDP\_CSR[0] register and, optionally, enabling the interrupt for endpoint 0 by writing 1 to the UDP\_IER register. The enumeration then begins by a control transfer.
- Configure the interrupt mask register which has been reset by the USB reset detection
- Enable the transceiver clearing the TXVDIS flag in the UDP\_TXVC register.

In this state UDPCK and MCK must be enabled.

**Warning:** Each time an ENDBUSRES interrupt is triggered, the Interrupt Mask Register and UDP\_CSR registers have been reset.

### 33.5.3.4 *From Default State to Address State*

After a set address standard device request, the USB host peripheral enters the address state.

**Warning:** Before the device enters in address state, it must achieve the Status IN transaction of the control transfer, i.e., the UDP device sets its new address once the TXCOMP flag in the UDP\_CSR[0] register has been received and cleared.

To move to address state, the driver software sets the FADDEN flag in the UDP\_GLB\_STAT register, sets its new address, and sets the FEN bit in the UDP\_FADDR register.

### 33.5.3.5 *From Address State to Configured State*

Once a valid Set Configuration standard request has been received and acknowledged, the device enables endpoints corresponding to the current configuration. This is done by setting the EPEDS and EPTYPE fields in the UDP\_CSRx registers and, optionally, enabling corresponding interrupts in the UDP\_IER register.

### 33.5.3.6 *Entering in Suspend State*

When a Suspend (no bus activity on the USB bus) is detected, the RXSUSP signal in the UDP\_ISR register is set. This triggers an interrupt if the corresponding bit is set in the UDP\_IMR register. This flag is cleared by writing to the UDP\_ICR register. Then the device enters Suspend Mode.

In this state bus powered devices must drain less than 500uA from the 5V VBUS. As an example, the microcontroller switches to slow clock, disables the PLL and main oscillator, and goes into Idle Mode. It may also switch off other devices on the board.

The USB device peripheral clocks can be switched off. Resume event is asynchronously detected. MCK and UDPCK can be switched off in the Power Management controller and the USB transceiver can be disabled by setting the TXVDIS field in the UDP\_TXVC register.

**Warning:** Read, write operations to the UDP registers are allowed only if MCK is enabled for the UDP peripheral. Switching off MCK for the UDP peripheral must be one of the last operations after writing to the UDP\_TXVC and acknowledging the RXSUSP.

### 33.5.3.7 *Receiving a Host Resume*

In suspend mode, a resume event on the USB bus line is detected asynchronously, transceiver and clocks are disabled (however the pullup shall not be removed).

Once the resume is detected on the bus, the WAKEUP signal in the UDP\_ISR is set. It may generate an interrupt if the corresponding bit in the UDP\_IMR register is set. This interrupt may be used to wake up the core, enable PLL and main oscillators and configure clocks.

**Warning:** Read, write operations to the UDP registers are allowed only if MCK is enabled for the UDP peripheral. MCK for the UDP must be enabled before clearing the WAKEUP bit in the UDP\_ICR register and clearing TXVDIS in the UDP\_TXVC register.

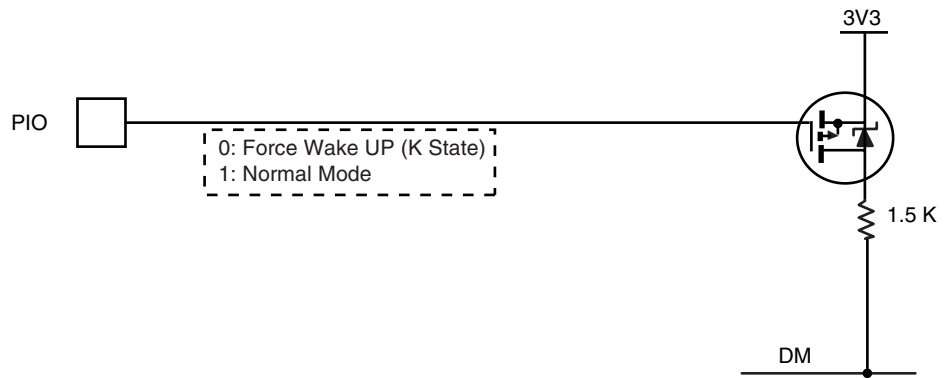
### 33.5.3.8 *Sending a Device Remote Wakeup*

In Suspend state it is possible to wake up the host sending an external resume.

- The device must wait at least 5 ms after being entered in suspend before sending an external resume.
- The device has 10 ms from the moment it starts to drain current and it forces a K state to resume the host.
- The device must force a K state from 1 to 15 ms to resume the host

To force a K state to the bus (DM at 3.3V and DP tied to GND), it is possible to use a transistor to connect a pullup on DM. The K state is obtained by disabling the pullup on DP and enabling the pullup on DM. This should be under the control of the application.

**Figure 33-15.** Board Schematic to Drive a K State



## 33.6 USB Device Port (UDP) User Interface

**WARNING:** The UDP peripheral clock in the Power Management Controller (PMC) must be enabled before any read/write operations to the UDP registers including the UDP\_TXCV register.

**Table 33-4.** UDP Memory Map

Offset	Register	Name	Access	Reset State
0x000	Frame Number Register	UDP_FRM_NUM	Read	0x0000_0000
0x004	Global State Register	UDP_GLB_STAT	Read/Write	0x0000_0000
0x008	Function Address Register	UDP_FADDR	Read/Write	0x0000_0100
0x00C	Reserved	–	–	–
0x010	Interrupt Enable Register	UDP_IER	Write	
0x014	Interrupt Disable Register	UDP_IDR	Write	
0x018	Interrupt Mask Register	UDP_IMR	Read	0x0000_1200
0x01C	Interrupt Status Register	UDP_ISR	Read	0x0000_XX00
0x020	Interrupt Clear Register	UDP_ICR	Write	
0x024	Reserved	–	–	–
0x028	Reset Endpoint Register	UDP_RST_EP	Read/Write	
0x02C	Reserved	–	–	–
0x030	Endpoint 0 Control and Status Register	UDP_CSR0	Read/Write	0x0000_0000
.	.			
.	.			
.	.			
See Note: <sup>(1)</sup>	Endpoint 5 Control and Status Register	UDP_CSR5	Read/Write	0x0000_0000
0x050	Endpoint 0 FIFO Data Register	UDP_FDR0	Read/Write	0x0000_0000
.	.			
.	.			
.	.			
See Note: <sup>(2)</sup>	Endpoint 5 FIFO Data Register	UDP_FDR5	Read/Write	0x0000_0000
0x070	Reserved	–	–	–
0x074	Transceiver Control Register	UDP_TXVC <sup>(3)</sup>	Read/Write	0x0000_0000
0x078 - 0xFC	Reserved	–	–	–

- Notes:
1. The addresses of the UDP\_CSRx registers are calculated as:  $0x030 + 4(\text{Endpoint Number} - 1)$ .
  2. The addresses of the UDP\_FDRx registers are calculated as:  $0x050 + 4(\text{Endpoint Number} - 1)$ .
  3. See Warning above the "UDP Memory Map" on this page.

### 33.6.1 UDP Frame Number Register

**Register Name:** UDP\_FRM\_NUM

**Access Type:** Read-only

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

- **FRM\_NUM[10:0]: Frame Number as Defined in the Packet Field Formats**

This 11-bit value is incremented by the host on a per frame basis. This value is updated at each start of frame. Value Updated at the SOF\_EOP (Start of Frame End of Packet).

- **FRM\_ERR: Frame Error**

This bit is set at SOF\_EOP when the SOF packet is received containing an error. This bit is reset upon receipt of SOF\_PID.

- **FRM\_OK: Frame OK**

This bit is set at SOF\_EOP when the SOF packet is received without any error. This bit is reset upon receipt of SOF\_PID (Packet Identification).

In the Interrupt Status Register, the SOF interrupt is updated upon receiving SOF\_PID. This bit is set without waiting for EOP.

Note: In the 8-bit Register Interface, FRM\_OK is bit 4 of FRM\_NUM\_H and FRM\_ERR is bit 3 of FRM\_NUM\_L.

## 33.6.2 UDP Global State Register

**Register Name:** UDP\_GLB\_STAT

**Access Type:** Read/Write

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

This register is used to get and set the device state as specified in Chapter 9 of the *USB Serial Bus Specification, Rev.2.0*.

- **FADDEN: Function Address Enable**

Read:

0 = Device is not in address state.

1 = Device is in address state.

Write:

0 = No effect, only a reset can bring back a device to the default state.

1 = Sets device in address state. This occurs after a successful Set Address request. Beforehand, the UDP\_FADDR register must have been initialized with Set Address parameters. Set Address must complete the Status Stage before setting FADDEN. Refer to chapter 9 of the *Universal Serial Bus Specification, Rev. 2.0* for more details.

- **CONFIG: Configured**

Read:

0 = Device is not in configured state.

1 = Device is in configured state.

Write:

0 = Sets device in a non configured state

1 = Sets device in configured state.

The device is set in configured state when it is in address state and receives a successful Set Configuration request. Refer to Chapter 9 of the *Universal Serial Bus Specification, Rev. 2.0* for more details.



### 33.6.3 UDP Function Address Register

**Register Name:** UDP\_FADDR

**Access Type:** Read/Write

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

- **FADD[6:0]: Function Address Value**

The Function Address Value must be programmed by firmware once the device receives a set address request from the host, and has achieved the status stage of the no-data control sequence. Refer to the *Universal Serial Bus Specification, Rev. 2.0* for more information. After power up or reset, the function address value is set to 0.

- **FEN: Function Enable**

Read:

0 = Function endpoint disabled.

1 = Function endpoint enabled.

Write:

0 = Disables function endpoint.

1 = Default value.

The Function Enable bit (FEN) allows the microcontroller to enable or disable the function endpoints. The microcontroller sets this bit after receipt of a reset from the host. Once this bit is set, the USB device is able to accept and transfer data packets from and to the host.



## 33.6.4 UDP Interrupt Enable Register

**Register Name:** UDP\_IER

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	WAKEUP	–	SOFINT	–	RXRSM	RXSUSP
7	6	5	4	3	2	1	0
		EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Enable Endpoint 0 Interrupt**

- **EP1INT: Enable Endpoint 1 Interrupt**

- **EP2INT: Enable Endpoint 2 Interrupt**

- **EP3INT: Enable Endpoint 3 Interrupt**

- **EP4INT: Enable Endpoint 4 Interrupt**

- **EP5INT: Enable Endpoint 5 Interrupt**

0 = No effect.

1 = Enables corresponding Endpoint Interrupt.

- **RXSUSP: Enable UDP Suspend Interrupt**

0 = No effect.

1 = Enables UDP Suspend Interrupt.

- **RXRSM: Enable UDP Resume Interrupt**

0 = No effect.

1 = Enables UDP Resume Interrupt.

- **SOFINT: Enable Start Of Frame Interrupt**

0 = No effect.

1 = Enables Start Of Frame Interrupt.

- **WAKEUP: Enable UDP bus Wakeup Interrupt**

0 = No effect.

1 = Enables USB bus Interrupt.

### 33.6.5 UDP Interrupt Disable Register

**Register Name:** UDP\_IDR

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	WAKEUP	–	SOFINT	–	RXRSM	RXSUSP
7	6	5	4	3	2	1	0
		EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Disable Endpoint 0 Interrupt**
  - **EP1INT: Disable Endpoint 1 Interrupt**
  - **EP2INT: Disable Endpoint 2 Interrupt**
  - **EP3INT: Disable Endpoint 3 Interrupt**
  - **EP4INT: Disable Endpoint 4 Interrupt**
  - **EP5INT: Disable Endpoint 5 Interrupt**
- 0 = No effect.  
1 = Disables corresponding Endpoint Interrupt.
- **RXSUSP: Disable UDP Suspend Interrupt**
- 0 = No effect.  
1 = Disables UDP Suspend Interrupt.
- **RXRSM: Disable UDP Resume Interrupt**
- 0 = No effect.  
1 = Disables UDP Resume Interrupt.
- **SOFINT: Disable Start Of Frame Interrupt**
- 0 = No effect.  
1 = Disables Start Of Frame Interrupt
- **WAKEUP: Disable USB Bus Interrupt**
- 0 = No effect.  
1 = Disables USB Bus Wakeup Interrupt.

## 33.6.6 UDP Interrupt Mask Register

**Register Name:** UDP\_IMR

**Access Type:** Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12 <sup>(1)</sup>	11	10	9	8
–	–	WAKEUP	–	SOFINT	–	RXRSM	RXSUSP
7	6	5	4	3	2	1	0
		EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

Note: 1. Bit 12 of UDP\_IMR cannot be masked and is always read at 1.

- **EP0INT: Mask Endpoint 0 Interrupt**
- **EP1INT: Mask Endpoint 1 Interrupt**
- **EP2INT: Mask Endpoint 2 Interrupt**
- **EP3INT: Mask Endpoint 3 Interrupt**
- **EP4INT: Mask Endpoint 4 Interrupt**
- **EP5INT: Mask Endpoint 5 Interrupt**

0 = Corresponding Endpoint Interrupt is disabled.

1 = Corresponding Endpoint Interrupt is enabled.

- **RXSUSP: Mask UDP Suspend Interrupt**

0 = UDP Suspend Interrupt is disabled.

1 = UDP Suspend Interrupt is enabled.

- **RXRSM: Mask UDP Resume Interrupt.**

0 = UDP Resume Interrupt is disabled.

1 = UDP Resume Interrupt is enabled.

- **SOFINT: Mask Start Of Frame Interrupt**

0 = Start of Frame Interrupt is disabled.

1 = Start of Frame Interrupt is enabled.



- **WAKEUP: USB Bus WAKEUP Interrupt**

0 = USB Bus Wakeup Interrupt is disabled.

1 = USB Bus Wakeup Interrupt is enabled.

Note: When the USB block is in suspend mode, the application may power down the USB logic. In this case, any USB HOST resume request that is made must be taken into account and, thus, the reset value of the RXRSM bit of the register UDP\_ IMR is enabled.

## 33.6.7 UDP Interrupt Status Register

**Register Name:** UDP\_ISR

**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
–	–	WAKEUP	ENDBUSRES	SOFINT	–	RXRSM	RXSUSP
7	6	5	4	3	2	1	0
		EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Endpoint 0 Interrupt Status**
- **EP1INT: Endpoint 1 Interrupt Status**
- **EP2INT: Endpoint 2 Interrupt Status**
- **EP3INT: Endpoint 3 Interrupt Status**
- **EP4INT: Endpoint 4 Interrupt Status**
- **EP5INT: Endpoint 5 Interrupt Status**

0 = No Endpoint0 Interrupt pending.

1 = Endpoint0 Interrupt has been raised.

Several signals can generate this interrupt. The reason can be found by reading UDP\_CSR0:

- RXSETUP set to 1
- RX\_DATA\_BK0 set to 1
- RX\_DATA\_BK1 set to 1
- TXCOMP set to 1
- STALLSENT set to 1

EP0INT is a sticky bit. Interrupt remains valid until EP0INT is cleared by writing in the corresponding UDP\_CSR0 bit.

- **RXSUSP: UDP Suspend Interrupt Status**

0 = No UDP Suspend Interrupt pending.

1 = UDP Suspend Interrupt has been raised.

The USB device sets this bit when it detects no activity for 3ms. The USB device enters Suspend mode.

- **RXRSM: UDP Resume Interrupt Status**

0 = No UDP Resume Interrupt pending.

1 =UDP Resume Interrupt has been raised.

The USB device sets this bit when a UDP resume signal is detected at its port.

After reset, the state of this bit is undefined, the application must clear this bit by setting the RXRSM flag in the UDP\_ICR register.

- **SOFINT: Start of Frame Interrupt Status**

0 = No Start of Frame Interrupt pending.

1 = Start of Frame Interrupt has been raised.

This interrupt is raised each time a SOF token has been detected. It can be used as a synchronization signal by using isochronous endpoints.

- **ENDBUSRES: End of BUS Reset Interrupt Status**

0 = No End of Bus Reset Interrupt pending.

1 = End of Bus Reset Interrupt has been raised.

This interrupt is raised at the end of a UDP reset sequence. The USB device must prepare to receive requests on the endpoint 0. The host starts the enumeration, then performs the configuration.

- **WAKEUP: UDP Resume Interrupt Status**

0 = No Wakeup Interrupt pending.

1 = A Wakeup Interrupt (USB Host Sent a RESUME or RESET) occurred since the last clear.

After reset the state of this bit is undefined, the application must clear this bit by setting the WAKEUP flag in the UDP\_ICR register.

## 33.6.8 UDP Interrupt Clear Register

**Register Name:** UDP\_ICR

**Access Type:** Write-only

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

- **RXSUSP: Clear UDP Suspend Interrupt**

0 = No effect.

1 = Clears UDP Suspend Interrupt.

- **RXRSM: Clear UDP Resume Interrupt**

0 = No effect.

1 = Clears UDP Resume Interrupt.

- **SOFINT: Clear Start Of Frame Interrupt**

0 = No effect.

1 = Clears Start Of Frame Interrupt.

- **ENDBUSRES: Clear End of Bus Reset Interrupt**

0 = No effect.

1 = Clears End of Bus Reset Interrupt.

- **WAKEUP: Clear Wakeup Interrupt**

0 = No effect.

1 = Clears Wakeup Interrupt.

### 33.6.9 UDP Reset Endpoint Register

**Register Name:** UDP\_RST\_EP

**Access Type:** Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
		EP5	EP4	EP3	EP2	EP1	EP0

- **EP0: Reset Endpoint 0**
- **EP1: Reset Endpoint 1**
- **EP2: Reset Endpoint 2**
- **EP3: Reset Endpoint 3**
- **EP4: Reset Endpoint 4**
- **EP5: Reset Endpoint 5**

This flag is used to reset the FIFO associated with the endpoint and the bit RXBYTECOUNT in the register UDP\_CSRx. It also resets the data toggle to DATA0. It is useful after removing a HALT condition on a BULK endpoint. Refer to Chapter 5.8.5 in the *USB Serial Bus Specification, Rev.2.0*.

**Warning:** This flag must be cleared at the end of the reset. It does not clear UDP\_CSRx flags.

0 = No reset.

1 = Forces the corresponding endpoint FIFO pointers to 0, therefore RXBYTECNT field is read at 0 in UDP\_CSRx register.



## 33.6.10 UDP Endpoint Control and Status Register

**Register Name:** UDP\_CSRx [x = 0..5]

**Access Type:** Read/Write

31	30	29	28	27	26	25	24
-	-	-	-	-	RXBYTECNT		
23	22	21	20	19	18	17	16
RXBYTECNT							
15	14	13	12	11	10	9	8
EPEDS	-	-	-	DTGLE	EPTYPE		
7	6	5	4	3	2	1	0
DIR	RX_DATA_BK1	FORCE STALL	TXPKTRDY	STALLSENT ISOERROR	RXSETUP	RX_DATA_BK0	TXCOMP

**WARNING:** Due to synchronization between MCK and UDPCK, the software application must wait for the end of the write operation before executing another write by polling the bits which must be set/cleared.

```

//! Clear flags of UDP UDP_CSR register and waits for synchronization
#define Udp_ep_clr_flag(pInterface, endpoint, flags) { \
    while (pInterface->UDP_CSR[endpoint] & (flags)) \
        pInterface->UDP_CSR[endpoint] &= ~(flags); \
}
//! Set flags of UDP UDP_CSR register and waits for synchronization
#define Udp_ep_set_flag(pInterface, endpoint, flags) { \
    while ( (pInterface->UDP_CSR[endpoint] & (flags)) != (flags) ) \
        pInterface->UDP_CSR[endpoint] |= (flags); \
}

```

- **TXCOMP: Generates an IN Packet with Data Previously Written in the DPR**

This flag generates an interrupt while it is set to one.

Write (Cleared by the firmware):

0 = Clear the flag, clear the interrupt.

1 = No effect.

Read (Set by the USB peripheral):

0 = Data IN transaction has not been acknowledged by the Host.

1 = Data IN transaction is achieved, acknowledged by the Host.

After having issued a Data IN transaction setting TXPKTRDY, the device firmware waits for TXCOMP to be sure that the host has acknowledged the transaction.

- **RX\_DATA\_BK0: Receive Data Bank 0**

This flag generates an interrupt while it is set to one.

Write (Cleared by the firmware):

0 = Notify USB peripheral device that data have been read in the FIFO's Bank 0.

1 = No effect.

Read (Set by the USB peripheral):

0 = No data packet has been received in the FIFO's Bank 0.

1 = A data packet has been received, it has been stored in the FIFO's Bank 0.

When the device firmware has polled this bit or has been interrupted by this signal, it must transfer data from the FIFO to the microcontroller memory. The number of bytes received is available in RXBYTCENT field. Bank 0 FIFO values are read through the UDP\_FDRx register. Once a transfer is done, the device firmware must release Bank 0 to the USB peripheral device by clearing RX\_DATA\_BK0.

- **RXSETUP: Received Setup**

This flag generates an interrupt while it is set to one.

Read:

0 = No setup packet available.

1 = A setup data packet has been sent by the host and is available in the FIFO.

Write:

0 = Device firmware notifies the USB peripheral device that it has read the setup data in the FIFO.

1 = No effect.

This flag is used to notify the USB device firmware that a valid Setup data packet has been sent by the host and successfully received by the USB device. The USB device firmware may transfer Setup data from the FIFO by reading the UDP\_FDRx register to the microcontroller memory. Once a transfer has been done, RXSETUP must be cleared by the device firmware.

Ensuing Data OUT transaction is not accepted while RXSETUP is set.

- **STALLSENT: Stall Sent (Control, Bulk Interrupt Endpoints) / ISOERROR (Isochronous Endpoints)**

This flag generates an interrupt while it is set to one.

STALLSENT: This ends a STALL handshake.

Read:

0 = The host has not acknowledged a STALL.

1 = Host has acknowledged the stall.

Write:

0 = Resets the STALLSENT flag, clears the interrupt.

1 = No effect.

This is mandatory for the device firmware to clear this flag. Otherwise the interrupt remains.

Refer to chapters 8.4.5 and 9.4.5 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the STALL handshake.

ISOERROR: A CRC error has been detected in an isochronous transfer.

Read:

0 = No error in the previous isochronous transfer.

1 = CRC error has been detected, data available in the FIFO are corrupted.

Write:

0 = Resets the ISOERROR flag, clears the interrupt.

1 = No effect.

- **TXPKTRDY: Transmit Packet Ready**

This flag is cleared by the USB device.

This flag is set by the USB device firmware.

Read:

0 = Can be set to one to send the FIFO data.

1 = The data is waiting to be sent upon reception of token IN.

Write:

0 = No effect.

1 = A new data payload is has been written in the FIFO by the firmware and is ready to be sent.

This flag is used to generate a Data IN transaction (device to host). Device firmware checks that it can write a data payload in the FIFO, checking that TXPKTRDY is cleared. Transfer to the FIFO is done by writing in the UDP\_FDRx register. Once the data payload has been transferred to the FIFO, the firmware notifies the USB device setting TXPKTRDY to one. USB bus transactions can start. TXCOMP is set once the data payload has been received by the host.

- **FORCESTALL: Force Stall (used by Control, Bulk and Isochronous Endpoints)**

Read:

0 = Normal state.

1 = Stall state.

Write:

0 = Return to normal state.

1 = Send STALL to the host.

Refer to chapters 8.4.5 and 9.4.5 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the STALL handshake.

Control endpoints: During the data stage and status stage, this bit indicates that the microcontroller cannot complete the request.

Bulk and interrupt endpoints: This bit notifies the host that the endpoint is halted.

The host acknowledges the STALL, device firmware is notified by the STALLSENT flag.

- **RX\_DATA\_BK1: Receive Data Bank 1 (only used by endpoints with ping-pong attributes)**

This flag generates an interrupt while it is set to one.

Write (Cleared by the firmware):

0 = Notifies USB device that data have been read in the FIFO's Bank 1.

1 = No effect.

Read (Set by the USB peripheral):

0 = No data packet has been received in the FIFO's Bank 1.

1 = A data packet has been received, it has been stored in FIFO's Bank 1.

When the device firmware has polled this bit or has been interrupted by this signal, it must transfer data from the FIFO to microcontroller memory. The number of bytes received is available in RXBYTECNT field. Bank 1 FIFO values are read through UDP\_FDRx register. Once a transfer is done, the device firmware must release Bank 1 to the USB device by clearing RX\_DATA\_BK1.

- **DIR: Transfer Direction (only available for control endpoints)**

Read/Write

0 = Allows Data OUT transactions in the control data stage.

1 = Enables Data IN transactions in the control data stage.

Refer to Chapter 8.5.3 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the control data stage.

This bit must be set before UDP\_CSRx/RXSETUP is cleared at the end of the setup stage. According to the request sent in the setup data packet, the data stage is either a device to host (DIR = 1) or host to device (DIR = 0) data transfer. It is not necessary to check this bit to reverse direction for the status stage.

- **EPTYPE[2:0]: Endpoint Type**

Read/Write

000	Control
001	Isochronous OUT
101	Isochronous IN
010	Bulk OUT
110	Bulk IN
011	Interrupt OUT
111	Interrupt IN

- **DTGLE: Data Toggle**

Read-only

0 = Identifies DATA0 packet.

1 = Identifies DATA1 packet.

Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on DATA0, DATA1 packet definitions.

- **EPEDS: Endpoint Enable Disable**

Read:

0 = Endpoint disabled.

1 = Endpoint enabled.

Write:

0 = Disables endpoint.

1 = Enables endpoint.

- **RXBYTECNT[10:0]: Number of Bytes Available in the FIFO**

Read-only

When the host sends a data packet to the device, the USB device stores the data in the FIFO and notifies the microcontroller. The microcontroller can load the data from the FIFO by reading RXBYTECENT bytes in the UDP\_FDRx register.

## 33.6.11 UDP FIFO Data Register

**Register Name:** UDP\_FDRx [x = 0..5]

**Access Type:** Read/Write

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

- **FIFO\_DATA[7:0]: FIFO Data Value**

The microcontroller can push or pop values in the FIFO through this register.

RXBYTECNT in the corresponding UDP\_CSRx register is the number of bytes to be read from the FIFO (sent by the host).

The maximum number of bytes to write is fixed by the Max Packet Size in the Standard Endpoint Descriptor. It can not be more than the physical memory size associated to the endpoint. Refer to the *Universal Serial Bus Specification, Rev. 2.0* for more information.



### 33.6.12 UDP Transceiver Control Register

**Register Name:** UDP\_TXVC

**Access Type:** Read/Write

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

**WARNING:** The UDP peripheral clock in the Power Management Controller (PMC) must be enabled before any read/write operations to the UDP registers including the UDP\_TXCV register.

- **TXVDIS: Transceiver Disable**

When UDP is disabled, power consumption can be reduced significantly by disabling the embedded transceiver. This can be done by setting TXVDIS field.

To enable the transceiver, TXVDIS must be cleared.

**NOTE:** If the USB pullup is not connected on DP, the user should not write in any UDP register other than the UDP\_TXVC register. This is because if DP and DM are floating at 0, or pulled down, then SE0 is received by the device with the consequence of a USB Reset.

## **34. MultiMedia Card Interface (MCI)**

### **34.1 Overview**

The MultiMedia Card Interface (MCI) supports the MultiMediaCard (MMC) Specification V2.2 and the SD Memory Card Specification V1.0.

The MCI includes a command register, response registers, data registers, timeout counters and error detection logic that automatically handle the transmission of commands and, when required, the reception of the associated responses and data with a limited processor overhead.

The MCI supports stream, block and multi-block data read and write, and is compatible with the Peripheral DMA Controller (PDC) channels, minimizing processor intervention for large buffer transfers.

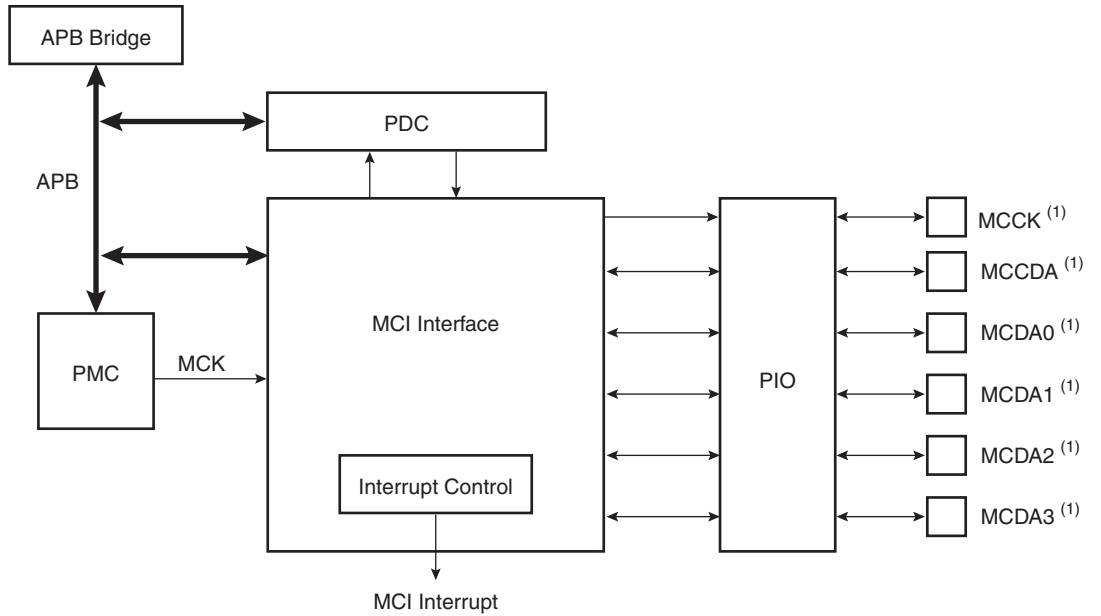
The MCI operates at a rate of up to Master Clock divided by 2 and supports the interfacing of 1 slot(s). Each slot may be used to interface with a MultiMediaCard bus (up to 30 Cards) or with a SD Memory Card. Only one slot can be selected at a time (slots are multiplexed). A bit field in the SD Card Register performs this selection.

The SD Memory Card communication is based on a 9-pin interface (clock, command, four data and three power lines) and the MultiMediaCard on a 7-pin interface (clock, command, one data, three power lines and one reserved for future use).

The SD Memory Card interface also supports MultiMediaCard operations. The main differences between SD and MultiMedia Cards are the initialization process and the bus topology.

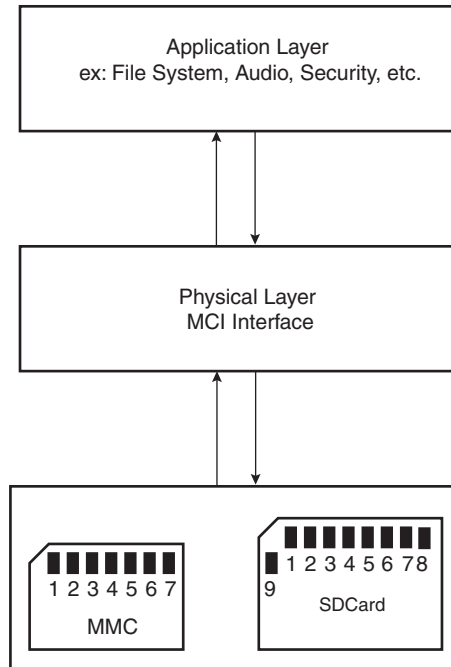
### 34.2 Block Diagram

Figure 34-1. Block Diagram



### 34.3 Application Block Diagram

Figure 34-2. Application Block Diagram





## 34.4 Pin Name List

**Table 34-1.** I/O Lines Description

Pin Name	Pin Description	Type <sup>(1)</sup>	Comments
MCCDA	Command/response	I/O/PP/OD	CMD of an MMC or SD Card
MCKK	Clock	I/O	CLK of an MMC or SD Card
MCDA0 - MCDA3	Data 0..3 of Slot A	I/O/PP	DAT0 of an MMC DAT[0..3] of an SD Card

Note: 1. I: Input, O: Output, PP: Push/Pull, OD: Open Drain.

## 34.5 Product Dependencies

### 34.5.1 I/O Lines

The pins used for interfacing the MultiMedia Cards or SD Cards may be multiplexed with PIO lines. The programmer must first program the PIO controllers to assign the peripheral functions to MCI pins.

### 34.5.2 Power Management

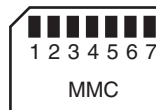
The MCI may be clocked through the Power Management Controller (PMC), so the programmer must first configure the PMC to enable the MCI clock.

### 34.5.3 Interrupt

The MCI interface has an interrupt line connected to the Advanced Interrupt Controller (AIC). Handling the MCI interrupt requires programming the AIC before configuring the MCI.

## 34.6 Bus Topology

**Figure 34-3.** Multimedia Memory Card Bus Topology



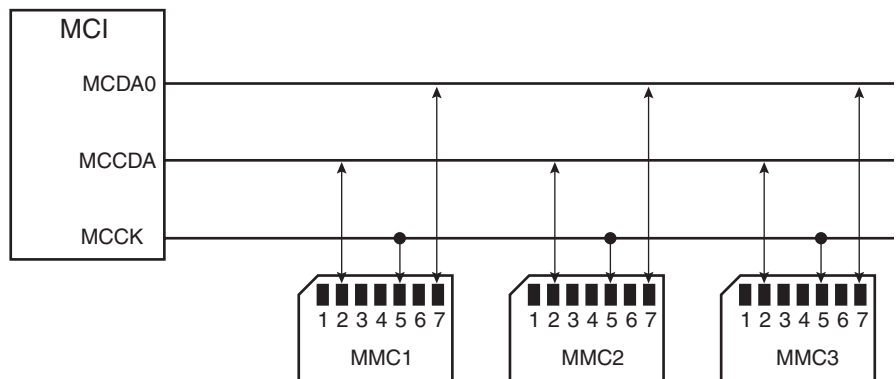
The MultiMedia Card communication is based on a 7-pin serial bus interface. It has three communication lines and four supply lines.

**Table 34-2.** Bus Topology

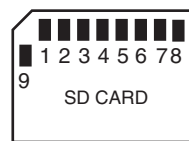
Pin Number	Name	Type <sup>(1)</sup>	Description	MCI Pin Name (Slot z)
1	RSV	NC	Not connected	-
2	CMD	I/O/PP/OD	Command/response	MCCDz
3	VSS1	S	Supply voltage ground	VSS
4	VDD	S	Supply voltage	VDD
5	CLK	I/O	Clock	MCKK
6	VSS2	S	Supply voltage ground	VSS
7	DAT[0]	I/O/PP	Data 0	MCDz0

Note: 1. I: Input, O: Output, PP: Push/Pull, OD: Open Drain.

**Figure 34-4.** MMC Bus Connections (One Slot)



**Figure 34-5.** SD Memory Card Bus Topology



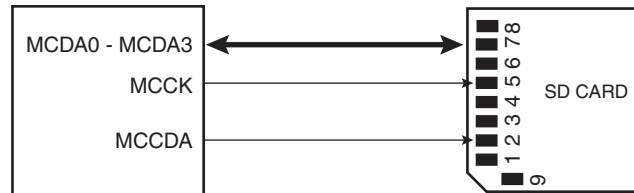
The SD Memory Card bus includes the signals listed in [Table 34-3](#).

**Table 34-3.** SD Memory Card Bus Signals

Pin Number	Name	Type <sup>(1)</sup>	Description	MCI Pin Name (Slot z)
1	CD/DAT[3]	I/O/PP	Card detect/ Data line Bit 3	MCDz3
2	CMD	PP	Command/response	MCCDz
3	VSS1	S	Supply voltage ground	VSS
4	VDD	S	Supply voltage	VDD
5	CLK	I/O	Clock	MCCK
6	VSS2	S	Supply voltage ground	VSS
7	DAT[0]	I/O/PP	Data line Bit 0	MCDz0
8	DAT[1]	I/O/PP	Data line Bit 1	MCDz1
9	DAT[2]	I/O/PP	Data line Bit 2	MCDz2

Note: 1. I: input, O: output, PP: Push Pull, OD: Open Drain.

**Figure 34-6.** SD Card Bus Connections with One Slot



When the MCI is configured to operate with SD memory cards, the width of the data bus can be selected in the MCI\_SDCR register. Clearing the SDCBUS bit in this register means that the width is one bit; setting it means that the width is four bits. In the case of multimedia cards, only the data line 0 is used. The other data lines can be used as independent PIOs.

## 34.7 MultiMedia Card Operations

After a power-on reset, the cards are initialized by a special message-based MultiMedia Card bus protocol. Each message is represented by one of the following tokens:

- **Command:** A command is a token that starts an operation. A command is sent from the host either to a single card (addressed command) or to all connected cards (broadcast command). A command is transferred serially on the CMD line.
- **Response:** A response is a token which is sent from an addressed card or (synchronously) from all connected cards to the host as an answer to a previously received command. A response is transferred serially on the CMD line.
- **Data:** Data can be transferred from the card to the host or vice versa. Data is transferred via the data line.

Card addressing is implemented using a session address assigned during the initialization phase by the bus controller to all currently connected cards. Their unique CID number identifies individual cards.

The structure of commands, responses and data blocks is described in the MultiMedia-Card System Specification. See also [Table 34-4 on page 452](#).

MultiMediaCard bus data transfers are composed of these tokens.

There are different types of operations. Addressed operations always contain a command and a response token. In addition, some operations have a data token; the others transfer their information directly within the command or response structure. In this case, no data token is present in an operation. The bits on the DAT and the CMD lines are transferred synchronous to the clock MCI Clock.

Two types of data transfer commands are defined:

- Sequential commands: These commands initiate a continuous data stream. They are terminated only when a stop command follows on the CMD line. This mode reduces the command overhead to an absolute minimum.
- Block-oriented commands: These commands send a data block succeeded by CRC bits.

Both read and write operations allow either single or multiple block transmission. A multiple block transmission is terminated when a stop command follows on the CMD line similarly to the sequential read.

The MCI provides a set of registers to perform the entire range of MultiMedia Card operations.

### 34.7.1 Command - Response Operation

After reset, the MCI is disabled and becomes valid after setting the MCIEN bit in the MCI\_CR Control Register.

The PWSEN bit saves power by dividing the MCI clock by  $2^{PWSDIV} + 1$  when the bus is inactive.

The command and the response of the card are clocked out with the rising edge of the MCI Clock.

All the timings for MultiMedia Card are defined in the MultiMediaCard System Specification.

The two bus modes (open drain and push/pull) needed to process all the operations are defined in the MCI command register. The MCI\_CMDR allows a command to be carried out.

For example, to perform an ALL\_SEND\_CID command:

Host Command					N <sub>ID</sub> Cycles			CID						
CMD	S	T	Content	CRC	E	Z	*****	Z	S	T	Content	Z	Z	Z

The command ALL\_SEND\_CID and the fields and values for the MCI\_CMDR Control Register are described in [Table 34-4](#) and [Table 34-5](#).

**Table 34-4.** ALL\_SEND\_CID Command Description

CMD Index	Type	Argument	Resp	Abbreviation	Command Description
CMD2	bcr <sup>(1)</sup>	[31:0] stuff bits	R2	ALL_SEND_CID	Asks all cards to send their CID numbers on the CMD line

Note: 1. bcr means broadcast command with response..

**Table 34-5.** Fields and Values for MCI\_CMDR Command Register

Field	Value
CMDNB (command number)	2 (CMD2)
RSPTYP (response type)	2 (R2: 136 bits response)
SPCMD (special command)	0 (not a special command)
OPCMD (open drain command)	1
MAXLAT (max latency for command to response)	0 (NID cycles ==> 5 cycles)
TRCMD (transfer command)	0 (No transfer)
TRDIR (transfer direction)	X (available only in transfer command)
TRTYP (transfer type)	X (available only in transfer command)

The MCI\_ARGR contains the argument field of the command.

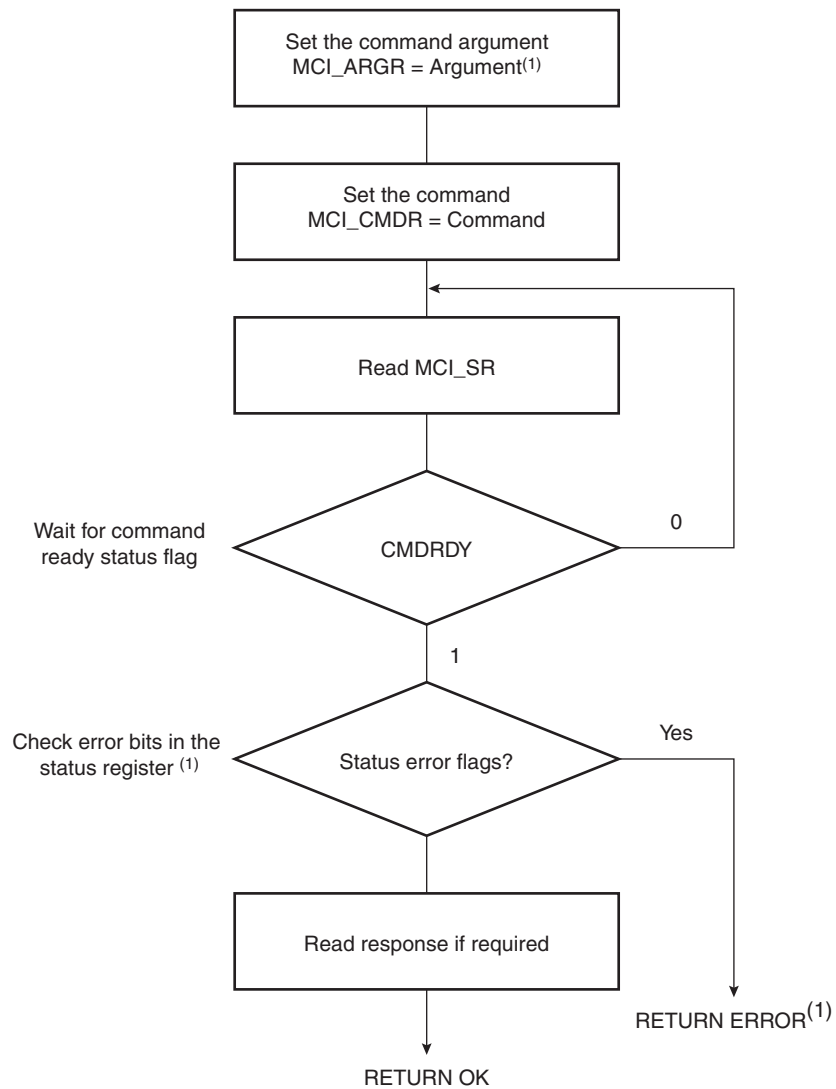
To send a command, the user must perform the following steps:

- Fill the argument register (MCI\_ARGR) with the command argument.
- Set the command register (MCI\_CMDR) (see [Table 34-5](#)).

The command is sent immediately after writing the command register. The status bit CMDRDY in the status register (MCI\_SR) is asserted when the command is completed. If the command requires a response, it can be read in the MCI response register (MCI\_RSPR). The response size can be from 48 bits up to 136 bits depending on the command. The MCI embeds an error detection to prevent any corrupted data during the transfer.

The following flowchart shows how to send a command to the card and read the response if needed. In this example, the status register bits are polled but setting the appropriate bits in the interrupt enable register (MCI\_IER) allows using an interrupt method.

**Figure 34-7.** Command/Response Functional Flow Diagram



Note: 1. If the command is SEND\_OP\_COND, the CRC error flag is always present (refer to R3 response in the MultiMedia Card specification).

### 34.7.2 Data Transfer Operation

The MultiMedia Card allows several read/write operations (single block, multiple blocks, stream, etc.). These kind of transfers can be selected setting the Transfer Type (TRTYP) field in the MCI Command Register (MCI\_CMDR).

These operations can be done using the features of the Peripheral DMA Controller (PDC). If the PDCMODE bit is set in MCI\_MR, then all reads and writes use the PDC facilities.

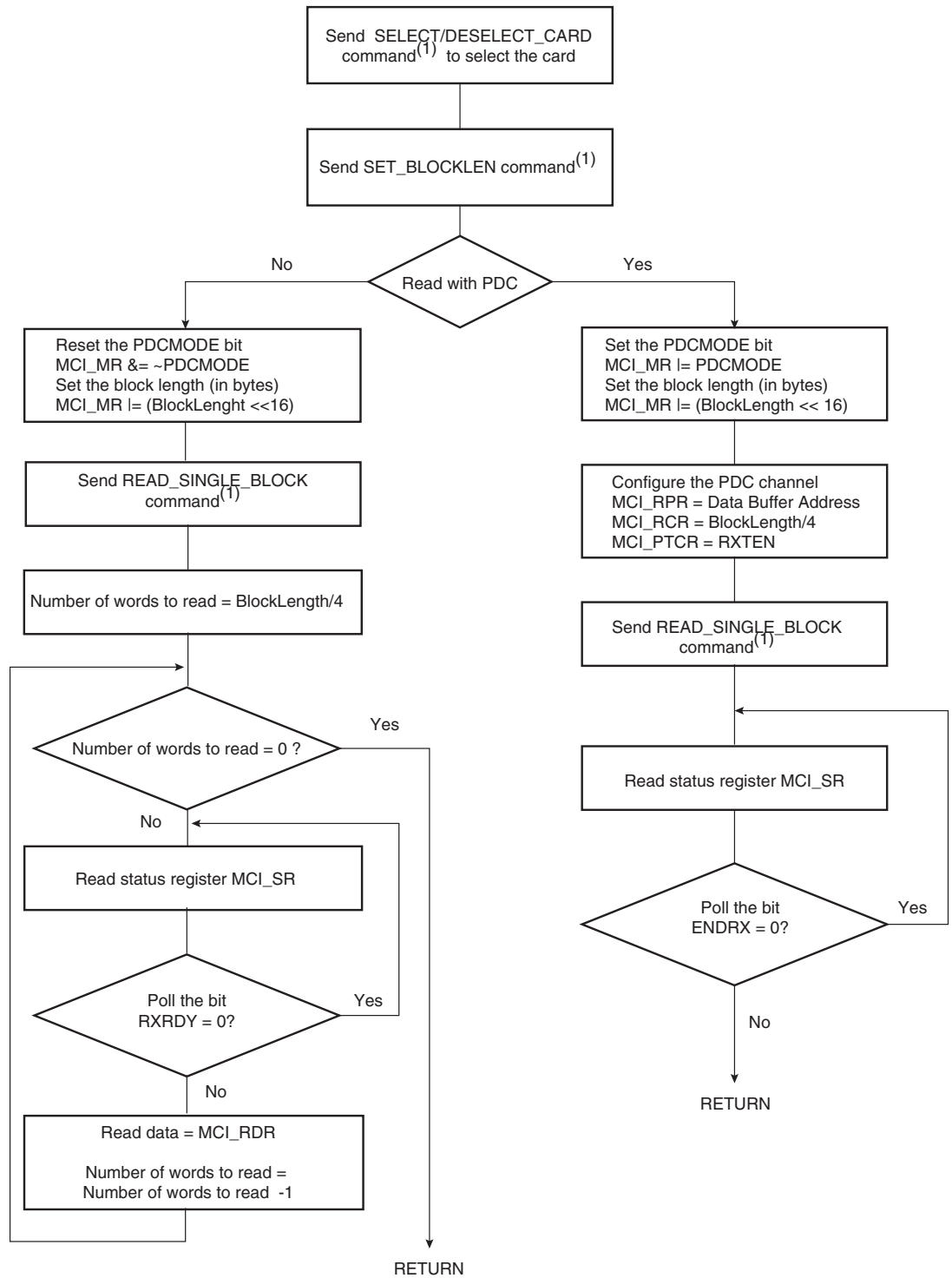
In all cases, the block length (BLKLEN field) must be defined in the mode register MCI\_MR. This field determines the size of the data block.

### 34.7.3 Read Operation

The following flowchart shows how to read a single block with or without use of PDC facilities. In this example (see [Figure 34-8](#)), a polling method is used to wait for the end of read. Simi-

larly, the user can configure the interrupt enable register (MCI\_IER) to trigger an interrupt at the end of read.

**Figure 34-8.** Read Functional Flow Diagram



Note: 1. It is assumed that this command has been correctly sent (see Figure 34-7).

## 34.7.4 Write Operation

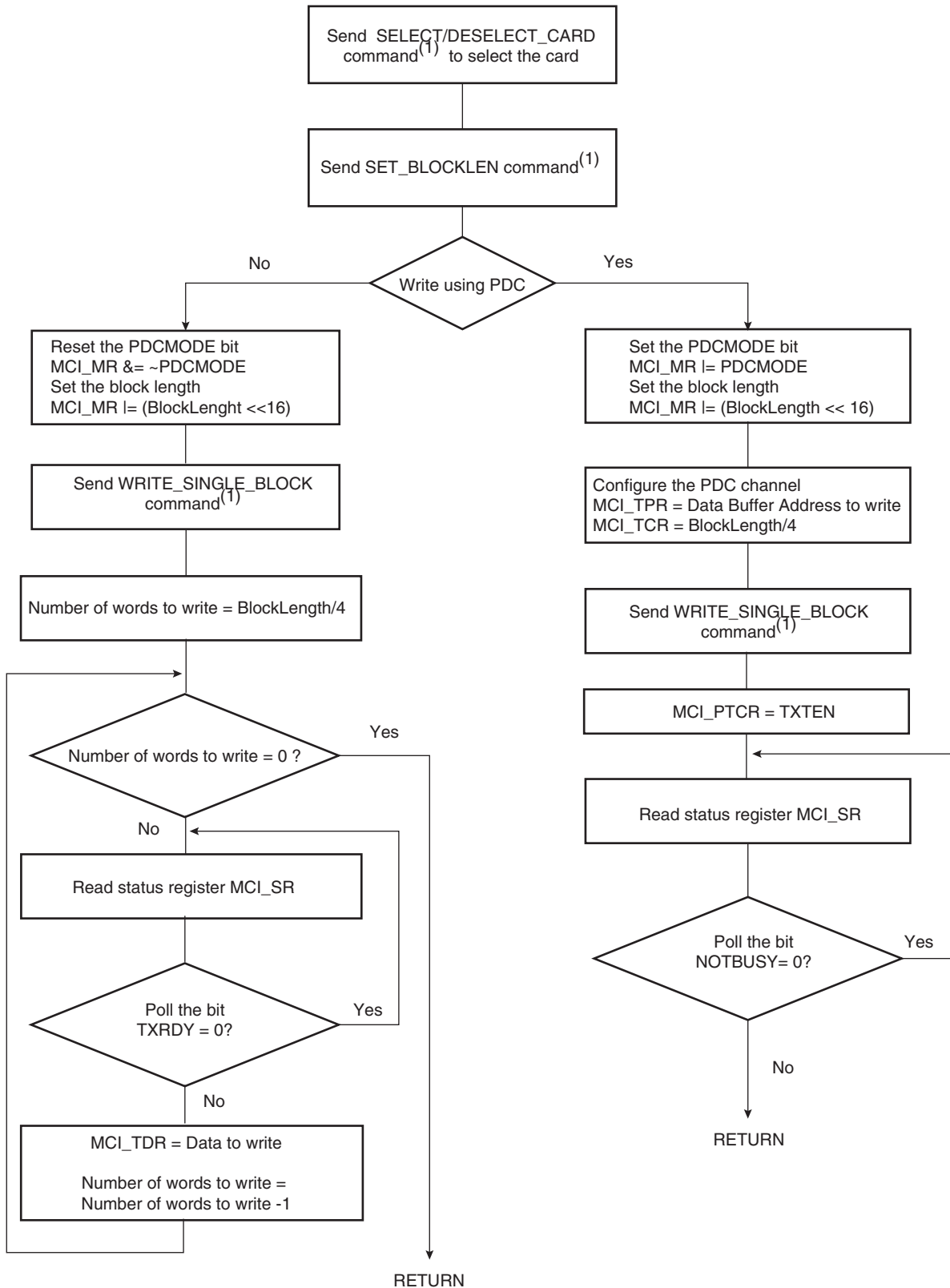
In write operation, the MCI Mode Register (MCI\_MR) is used to define the padding value when writing non-multiple block size. If the bit PDCPADV is 0, then 0x00 value is used when padding data, otherwise 0xFF is used.

If set, the bit PDCMODE enables PDC transfer.

The following flowchart shows how to write a single block with or without use of PDC facilities (see [Figure 34-9](#)). Polling or interrupt method can be used to wait for the end of write according to the contents of the Interrupt Mask Register (MCI\_IMR).



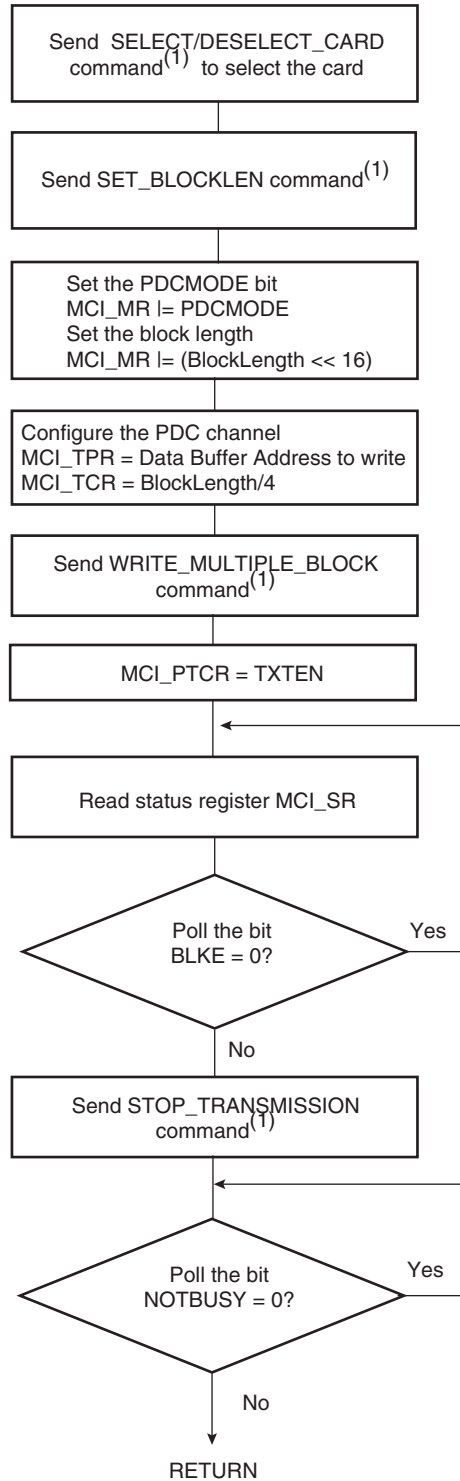
Figure 34-9. Write Functional Flow Diagram



Note: 1. It is assumed that this command has been correctly sent (see Figure 34-7).

The following flowchart shows how to manage a multiple write block transfer with the PDC (see [Figure 34-10](#)). Polling or interrupt method can be used to wait for the end of write according to the contents of the Interrupt Mask Register (MCI\_IMR).

**Figure 34-10.** Multiple Write Functional Flow Diagram



Note: 1. It is assumed that this command has been correctly sent (see [Figure 34-7](#)).

### **34.8 SD Card Operations**

The MultiMedia Card Interface allows processing of SD Memory (Secure Digital Memory Card) Card commands.

SD cards are based on the Multi Media Card (MMC) format, but are physically slightly thicker and feature higher data transfer rates, a lock switch on the side to prevent accidental overwriting and security features. The physical form factor, pin assignment and data transfer protocol are forward-compatible with the MultiMedia Card with some additions. SD is covered by numerous patents and trademarks, and licensing is only available through the Secure Digital Card Association.

The SD Card communication is based on a 9-pin interface (Clock, Command, 4 x Data and 3 x Power lines). The communication protocol is defined as a part of this specification. The main difference between the SD Card and the MultiMedia Card is the initialization process.

The SD Card Register (MCI\_SDCR) allows selection of the Card Slot and the data bus width.

The SD Card bus allows dynamic configuration of the number of data lines. After power up, by default, the SD Card uses only DAT0 for data transfer. After initialization, the host can change the bus width (number of active data lines).

## 34.9 MultiMedia Card Interface (MCI) User Interface

**Table 34-6.** Register Mapping

Offset	Register	Register Name	Read/Write	Reset
0x00	Control Register	MCI_CR	Write	–
0x04	Mode Register	MCI_MR	Read/write	0x0
0x08	Data Timeout Register	MCI_DTOR	Read/write	0x0
0x0C	SD Card Register	MCI_SDCR	Read/write	0x0
0x10	Argument Register	MCI_ARGR	Read/write	0x0
0x14	Command Register	MCI_CMDR	Write	–
0x18 - 0x1C	Reserved	–	–	–
0x20	Response Register <sup>(1)</sup>	MCI_RSPR	Read	0x0
0x24	Response Register <sup>(1)</sup>	MCI_RSPR	Read	0x0
0x28	Response Register <sup>(1)</sup>	MCI_RSPR	Read	0x0
0x2C	Response Register <sup>(1)</sup>	MCI_RSPR	Read	0x0
0x30	Receive Data Register	MCI_RDR	Read	0x0
0x34	Transmit Data Register	MCI_TDR	Write	–
0x38 - 0x3C	Reserved	–	–	–
0x40	Status Register	MCI_SR	Read	0xC0E5
0x44	Interrupt Enable Register	MCI_IER	Write	–
0x48	Interrupt Disable Register	MCI_IDR	Write	–
0x4C	Interrupt Mask Register	MCI_IMR	Read	0x0
0x50-0xFC	Reserved	–	–	–
0x100-0x124	Reserved for the PDC	–	–	–

Note: 1. The response register can be read by N accesses at the same MCI\_RSPR or at consecutive addresses (0x20 to 0x2C). N depends on the size of the response.

## 34.9.1 MCI Control Register

**Name:** MCI\_CR

**Access Type:** Write-only

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

- **MCIEN: Multi-Media Interface Enable**

0 = No effect.

1 = Enables the Multi-Media Interface if MCDIS is 0.

- **MCIDIS: Multi-Media Interface Disable**

0 = No effect.

1 = Disables the Multi-Media Interface.

- **PWSEN: Power Save Mode Enable**

0 = No effect.

1 = Enables the Power Saving Mode if PWSDIS is 0.

**Warning:** Before enabling this mode, the user must set a value different from 0 in the PWSDIV field (Mode Register MCI\_MR).

- **PWSDIS: Power Save Mode Disable**

0 = No effect.

1 = Disables the Power Saving Mode.

- **SWRST: Software Reset**

0 = No effect.

1 = Resets the MCI. A software triggered hardware reset of the MCI interface is performed.

## 34.9.2 MCI Mode Register

**Name:** MCI\_MR

**Access Type:** Read/write

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

- **CLKDIV: Clock Divider**

MultiMedia Card Interface clock (MCK or MCI\_CK) is Master Clock (MCK) divided by  $(2^{*(CLKDIV+1)})$ .

- **PWSDIV: Power Saving Divider**

MultiMedia Card Interface clock is divided by  $2^{(PWSDIV)} + 1$  when entering Power Saving Mode.

**Warning:** This value must be different from 0 before enabling the Power Save Mode in the MCI\_CR (MCI\_PWSEN bit).

- **PDCPADV: PDC Padding Value**

0 = 0x00 value is used when padding data in write transfer (not only PDC transfer).

1 = 0xFF value is used when padding data in write transfer (not only PDC transfer).

- **PDCMODE: PDC-oriented Mode**

0 = Disables PDC transfer

1 = Enables PDC transfer. In this case, UNRE and OVRE flags in the MCI Mode Register (MCI\_SR) are deactivated after the PDC transfer has been completed.

- **BLKLEN: Data Block Length**

This field determines the size of the data block.

Bits 16 and 17 must be set to 0

### 34.9.3 MCI Data Timeout Register

**Name:** MCI\_DTOR

**Access Type:** Read/write

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

- **DTOCYC: Data Timeout Cycle Number**
- **DTOMUL: Data Timeout Multiplier**

These fields determine the maximum number of Master Clock cycles that the MCI waits between two data block transfers. It equals (DTCYC x Multiplier).

Multiplier is defined by DTOMUL as shown in the following table:

DTOMUL			Multiplier
0	0	0	1
0	0	1	16
0	1	0	128
0	1	1	256
1	0	0	1024
1	0	1	4096
1	1	0	65536
1	1	1	1048576

If the data time-out set by DTOCYC and DTOMUL has been exceeded, the Data Time-out Error flag (DTCOE) in the MCI Status Register (MCI\_SR) raises.

## 34.9.4 MCI SDCard Register

**Name:** MCI\_SDCR

**Access Type:** Read/write

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

- **SDCSEL: SDCard Slot**

SDCSEL		SDCard Slot
0	0	Slot A is selected.
0	1	–
1	0	–
1	1	–

- **SDCBUS: SDCard Bus Width**

0 = 1-bit data bus

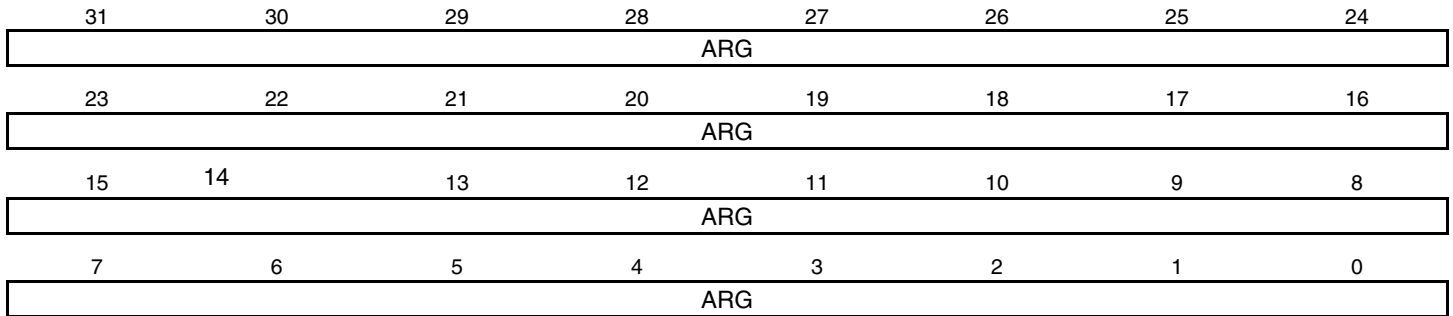
1 = 4-bit data bus



## 34.9.5 MCI Argument Register

Name: MCI\_ARGR

Access Type: Read/write



- ARG: Command Argument

## 34.9.6 MCI Command Register

**Name:** MCI\_CMDR

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	TRTYP		TRDIR	TRCMD	
15	14	13	12	11	10	9	8
–	–	–	MAXLAT	OPDCMD	SPCMD		
7	6	5	4	3	2	1	0
RSPTYP		CMDNB					

This register is write-protected while CMDRDY is 0 in MCI\_SR. If an Interrupt command is sent, this register is only write-able by an interrupt response (field SPCMD). This means that the current command execution cannot be interrupted or modified.

- **CMDNB: Command Number**
- **RSPTYP: Response Type**

RSP		Response Type
0	0	No response.
0	1	48-bit response.
1	0	136-bit response.
1	1	Reserved.

- **SPCMD: Special Command**

SPCMD			Command
0	0	0	Not a special CMD.
0	0	1	Initialization CMD: 74 clock cycles for initialization sequence.
0	1	0	Synchronized CMD: Wait for the end of the current data block transfer before sending the pending command.
0	1	1	Reserved.
1	0	0	Interrupt command: Corresponds to the Interrupt Mode (CMD40).
1	0	1	Interrupt response: Corresponds to the Interrupt Mode (CMD40).

- **OPDCMD: Open Drain Command**

0 = Push pull command

1 = Open drain command

- **MAXLAT: Max Latency for Command to Response**

0 = 5-cycle max latency

1 = 64-cycle max latency

- **TRCMD: Transfer Command**

TRCMD		Transfer Type
0	0	No data transfer
0	1	Start data transfer
1	0	Stop data transfer
1	1	Reserved

- **TRDIR: Transfer Direction**

0 = Write

1 = Read

- **TRTYP: Transfer Type**

TRTYP			Transfer Type
0	0	0	MMC/SDCard Single Block
0	0	1	MMC/SDCard Multiple Block
0	1	0	MMC Stream
0	1	1	Reserved

## 34.9.7 MCI Response Register

Name: MCI\_RSPR

Access Type: Read-only

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

- **RSP: Response**

Note: 1. The response register can be read by N accesses at the same MCI\_RSPR or at consecutive addresses (0x20 to 0x2C). N depends on the size of the response.

## 34.9.8 MCI Receive Data Register

Name: MCI\_RDR

Access Type: Read-only

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

- **DATA: Data to Read**

## 34.9.9 MCI Transmit Data Register

Name: MCI\_TDR

Access Type: Write-only

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

- **DATA: Data to Write**

## 34.9.10 MCI Status Register

**Name:** MCI\_SR

**Access Type:** Read-only

31	30	29	28	27	26	25	24
UNRE	OVRE	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	DTOE	DCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	–	–	–	–	–	–
7	6	5	4	3	2	1	0
ENDTX	ENDRX	NOTBUSY	DTIP	BLKE	TXRDY	RXRDY	CMDRDY

- **CMDRDY: Command Ready**

0 = A command is in progress.

1 = The last command has been sent. Cleared when writing in the MCI\_CMDR.

- **RXRDY: Receiver Ready**

0 = Data has not yet been received since the last read of MCI\_RDR.

1 = Data has been received since the last read of MCI\_RDR.

- **TXRDY: Transmit Ready**

0 = The last data written in MCI\_TDR has not yet been transferred in the Shift Register.

1 = The last data written in MCI\_TDR has been transferred in the Shift Register.

- **BLKE: Data Block Ended**

**This flag must be used only for Write Operations.**

0 = A data block transfer is not yet finished. Cleared when reading the MCI\_SR.

1 = A data block transfer has ended, including the CRC16 Status transmission.

In PDC mode (PDCMODE=1), the flag is set when the CRC Status of the last block has been transmitted (TXBUFE already set).

Otherwise (PDCMODE=0), the flag is set for each transmitted CRC Status.

Refer to the MMC or SD Specification for more details concerning the CRC Status.

- **DTIP: Data Transfer in Progress**

0 = No data transfer in progress.

1 = The current data transfer is still in progress, including CRC16 calculation. Cleared at the end of the CRC16 calculation.

- **NOTBUSY: MCI Not Busy**

**This flag must be used only for Write Operations.**

A block write operation uses a simple busy signalling of the write operation duration on the data (DAT0) line: during a data transfer block, if the card does not have a free data receive buffer, the card indicates this condition by pulling down the data line (DAT0) to LOW. The card stops pulling down the data line as soon as at least one receive buffer for the defined data transfer block length becomes free.

The NOTBUSY flag allows to deal with these different states.

0 = The MCI is not ready for new data transfer. Cleared at the end of the card response.

1 = The MCI is ready for new data transfer. Set when the busy state on the data line has ended. This corresponds to a free internal data receive buffer of the card.

Refer to the MMC or SD Specification for more details concerning the busy behavior.

- **ENDRX: End of RX Buffer**

0 = The Receive Counter Register has not reached 0 since the last write in MCI\_RCR or MCI\_RNCR.

1 = The Receive Counter Register has reached 0 since the last write in MCI\_RCR or MCI\_RNCR.

- **ENDTX: End of TX Buffer**

0 = The Transmit Counter Register has not reached 0 since the last write in MCI\_TCR or MCI\_TNCR.

1 = The Transmit Counter Register has reached 0 since the last write in MCI\_TCR or MCI\_TNCR.

Note: BLKE and NOTBUSY flags can be used to check that the data has been successfully transmitted on the data lines and not only transferred from the PDC to the MCI Controller.

- **RXBUFF: RX Buffer Full**

0 = MCI\_RCR or MCI\_RNCR has a value other than 0.

1 = Both MCI\_RCR and MCI\_RNCR have a value of 0.

- **TXBUFE: TX Buffer Empty**

0 = MCI\_TCR or MCI\_TNCR has a value other than 0.

1 = Both MCI\_TCR and MCI\_TNCR have a value of 0.

Note: BLKE and NOTBUSY flags can be used to check that the data has been successfully transmitted on the data lines and not only transferred from the PDC to the MCI Controller.

- **RINDE: Response Index Error**

0 = No error.

1 = A mismatch is detected between the command index sent and the response index received. Cleared when writing in the MCI\_CMDR.

- **RDIRE: Response Direction Error**

0 = No error.

1 = The direction bit from card to host in the response has not been detected.

- **RCRCE: Response CRC Error**

0 = No error.

1 = A CRC7 error has been detected in the response. Cleared when writing in the MCI\_CMDR.

- **RENDE: Response End Bit Error**

0 = No error.

1 = The end bit of the response has not been detected. Cleared when writing in the MCI\_CMDR.

- **RTOE: Response Time-out Error**

0 = No error.

1 = The response time-out set by MAXLAT in the MCI\_CMDR has been exceeded. Cleared when writing in the MCI\_CMDR.

- **DCRCE: Data CRC Error**

0 = No error.

1 = A CRC16 error has been detected in the last data block. Cleared by reading in the MCI\_SR register.

- **DTOE: Data Time-out Error**

0 = No error.

1 = The data time-out set by DTOCYC and DTOMUL in MCI\_DTOR has been exceeded. Cleared by reading in the MCI\_SR register.

- **OVRE: Overrun**

0 = No error.

1 = At least one 8-bit received data has been lost (not read). Cleared when sending a new data transfer command.

- **UNRE: Underrun**

0 = No error.

1 = At least one 8-bit data has been sent without valid information (not written). Cleared when sending a new data transfer command.

- **RXBUFF: RX Buffer Full**

0 = MCI\_RCR or MCI\_RNCR has a value other than 0.

1 = Both MCI\_RCR and MCI\_RNCR have a value of 0.

- **TXBUFE: TX Buffer Empty**

0 = MCI\_TCR or MCI\_TNCR has a value other than 0.

1 = Both MCI\_TCR and MCI\_TNCR have a value of 0.

## 34.9.11 MCI Interrupt Enable Register

Name: MCI\_IER

Access Type: Write-only

31	30	29	28	27	26	25	24
UNRE	OVRE	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	DTOE	DCRCE	RTOE	RENDE	RRCRCE	RDIRE	RINDE
15	14	13	12	11	10	9	8
TXBUFE	RXBUFFER	–	–	–	–	–	–
7	6	5	4	3	2	1	0
ENDTX	ENDRX	NOTBUSY	DTIP	BLKE	TXRDY	RXRDY	CMDRDY

- **CMDRDY: Command Ready Interrupt Enable**
- **RXRDY: Receiver Ready Interrupt Enable**
- **TXRDY: Transmit Ready Interrupt Enable**
- **BLKE: Data Block Ended Interrupt Enable**
- **DTIP: Data Transfer in Progress Interrupt Enable**
- **NOTBUSY: Data Not Busy Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable**
- **ENDTX: End of Transmit Buffer Interrupt Enable**
- **RXBUFFER: Receive Buffer Full Interrupt Enable**
- **TXBUFE: Transmit Buffer Empty Interrupt Enable**
- **RINDE: Response Index Error Interrupt Enable**
- **RDIRE: Response Direction Error Interrupt Enable**
- **RRCRCE: Response CRC Error Interrupt Enable**
- **RENDE: Response End Bit Error Interrupt Enable**
- **RTOE: Response Time-out Error Interrupt Enable**
- **DCRCE: Data CRC Error Interrupt Enable**
- **DTOE: Data Time-out Error Interrupt Enable**
- **OVRE: Overrun Interrupt Enable**
- **UNRE: UnderRun Interrupt Enable**

0 = No effect.

1 = Enables the corresponding interrupt.



## 34.9.12 MCI Interrupt Disable Register

**Name:** MCI\_IDR

**Access Type:** Write-only

31	30	29	28	27	26	25	24
UNRE	OVRE	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	DTOE	DCRCE	RTOE	RENDE	RRCRCE	RDIRE	RINDE
15	14	13	12	11	10	9	8
TXBUFE	RXBUFFER	–	–	–	–	–	–
7	6	5	4	3	2	1	0
ENDTX	ENDRX	NOTBUSY	DTIP	BLKE	TXRDY	RXRDY	CMDRDY

- **CMDRDY: Command Ready Interrupt Disable**
- **RXRDY: Receiver Ready Interrupt Disable**
- **TXRDY: Transmit Ready Interrupt Disable**
- **BLKE: Data Block Ended Interrupt Disable**
- **DTIP: Data Transfer in Progress Interrupt Disable**
- **NOTBUSY: Data Not Busy Interrupt Disable**
- **ENDRX: End of Receive Buffer Interrupt Disable**
- **ENDTX: End of Transmit Buffer Interrupt Disable**
- **RXBUFFER: Receive Buffer Full Interrupt Disable**
- **TXBUFE: Transmit Buffer Empty Interrupt Disable**
- **RINDE: Response Index Error Interrupt Disable**
- **RDIRE: Response Direction Error Interrupt Disable**
- **RRCRCE: Response CRC Error Interrupt Disable**
- **RENDE: Response End Bit Error Interrupt Disable**
- **RTOE: Response Time-out Error Interrupt Disable**
- **DCRCE: Data CRC Error Interrupt Disable**
- **DTOE: Data Time-out Error Interrupt Disable**
- **OVRE: Overrun Interrupt Disable**
- **UNRE: UnderRun Interrupt Disable**

0 = No effect.

1 = Disables the corresponding interrupt.

### 34.9.13 MCI Interrupt Mask Register

Name: MCI\_IMR

Access Type: Read-only

31	30	29	28	27	26	25	24
UNRE	OVRE	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	DTOE	DCRCE	RTOE	RENDE	RRCRCE	RDIRE	RINDE
15	14	13	12	11	10	9	8
TXBUFE	RXBUFFER	–	–	–	–	–	–
7	6	5	4	3	2	1	0
ENDTX	ENDRX	NOTBUSY	DTIP	BLKE	TXRDY	RXRDY	CMDRDY

- **CMDRDY: Command Ready Interrupt Mask**
- **RXRDY: Receiver Ready Interrupt Mask**
- **TXRDY: Transmit Ready Interrupt Mask**
- **BLKE: Data Block Ended Interrupt Mask**
- **DTIP: Data Transfer in Progress Interrupt Mask**
- **NOTBUSY: Data Not Busy Interrupt Mask**
- **ENDRX: End of Receive Buffer Interrupt Mask**
- **ENDTX: End of Transmit Buffer Interrupt Mask**
- **RXBUFFER: Receive Buffer Full Interrupt Mask**
- **TXBUFE: Transmit Buffer Empty Interrupt Mask**
- **RINDE: Response Index Error Interrupt Mask**
- **RDIRE: Response Direction Error Interrupt Mask**
- **RRCRCE: Response CRC Error Interrupt Mask**
- **RENDE: Response End Bit Error Interrupt Mask**
- **RTOE: Response Time-out Error Interrupt Mask**
- **DCRCE: Data CRC Error Interrupt Mask**
- **DTOE: Data Time-out Error Interrupt Mask**
- **OVRE: Overrun Interrupt Mask**
- **UNRE: UnderRun Interrupt Mask**

0 = The corresponding interrupt is not enabled.

1 = The corresponding interrupt is enabled.

## 35. Analog-to-Digital Converter (ADC)

### 35.1 Overview

The ADC is based on a Successive Approximation Register (SAR) 10-bit Analog-to-Digital Converter (ADC). It also integrates an 8-to-1 analog multiplexer, making possible the analog-to-digital conversions of 8 analog lines. The conversions extend from 0V to ADVREF.

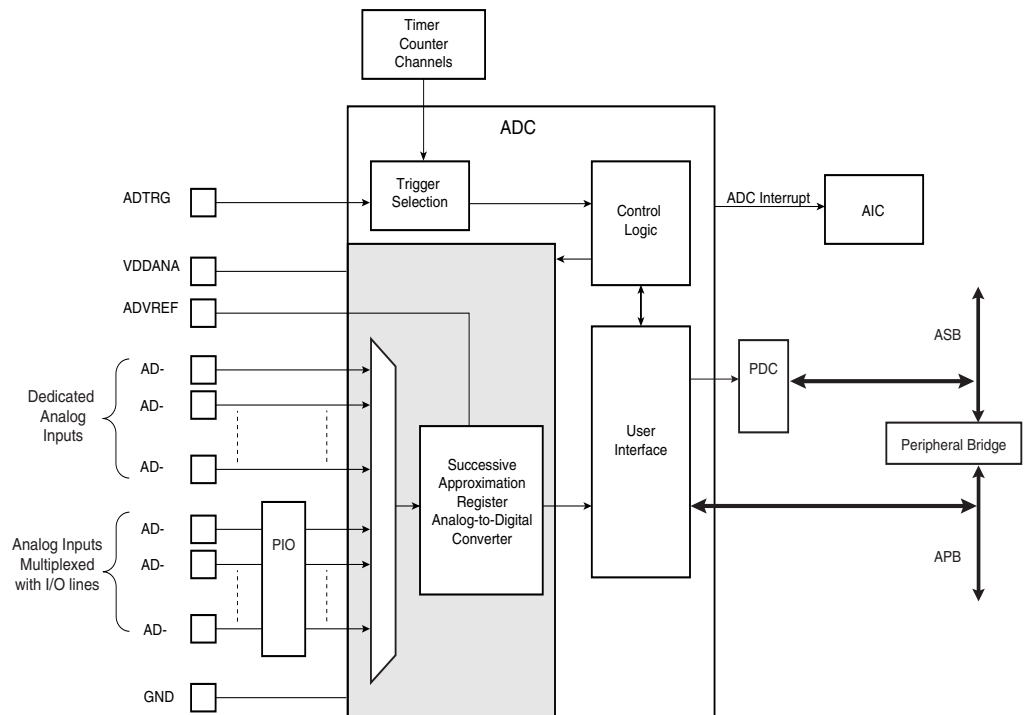
The ADC supports an 8-bit or 10-bit resolution mode, and conversion results are reported in a common register for all channels, as well as in a channel-dedicated register. Software trigger, external trigger on rising edge of the ADTRG pin or internal triggers from Timer Counter output(s) are configurable.

The ADC also integrates a Sleep Mode and a conversion sequencer and connects with a PDC channel. These features reduce both power consumption and processor intervention.

Finally, the user can configure ADC timings, such as Startup Time and Sample & Hold Time.

### 35.2 Block Diagram

Figure 35-1. Analog-to-Digital Converter Block Diagram



### 35.3 Signal Description

**Table 35-1.** ADC Pin Description

Pin Name	Description
VDDANA	Analog power supply
ADVREF	Reference voltage
AD0 - AD7	Analog input channels
ADTRG	External trigger

### 35.4 Product Dependencies

#### 35.4.1 Power Management

The ADC is automatically clocked after the first conversion in Normal Mode. In Sleep Mode, the ADC clock is automatically stopped after each conversion. As the logic is small and the ADC cell can be put into Sleep Mode, the Power Management Controller has no effect on the ADC behavior.

#### 35.4.2 Interrupt Sources

The ADC interrupt line is connected on one of the internal sources of the Advanced Interrupt Controller. Using the ADC interrupt requires the AIC to be programmed first.

#### 35.4.3 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.

#### 35.4.4 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.

#### 35.4.5 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 non-connected.

#### 35.4.6 Conversion Performances

For performance and electrical characteristics of the ADC, see the DC Characteristics section.

## 35.5 Functional Description

### 35.5.1 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 Sample and Hold Clock cycles as defined in the field SHTIM of the "ADC Mode Register" on page 484 and 10 ADC Clock cycles. The ADC Clock frequency is selected in the PRESCAL field of the Mode Register (ADC\_MR).

The ADC clock range is between  $MCK/2$ , if PRESCAL is 0, and  $MCK/128$ , if PRESCAL is set to 63 (0x3F). PRESCAL must be programmed in order to provide an ADC clock frequency according to the parameters given in the Product definition section.

### 35.5.2 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.

### 35.5.3 Conversion Resolution

The ADC supports 8-bit or 10-bit resolutions. The 8-bit selection is performed by setting the bit LOWRES 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 bit LOWRES, the ADC switches in the lowest resolution and the conversion results can be read in the eight 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.

Moreover, when a PDC channel is connected to the ADC, 10-bit resolution sets the transfer request sizes to 16-bit. Setting the bit LOWRES automatically switches to 8-bit data transfers. In this case, the destination buffers are optimized.

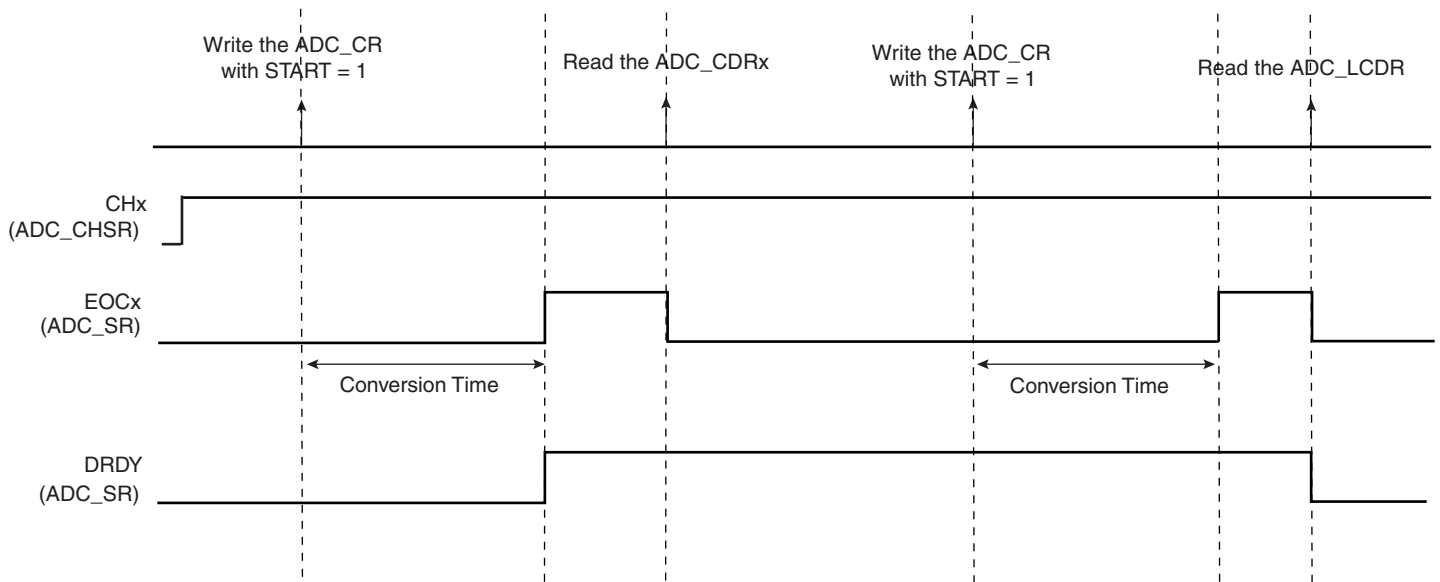
### 35.5.4 Conversion Results

When a conversion is completed, the resulting 10-bit digital value is stored in the Channel Data Register (ADC\_CDR) of the current channel and in the ADC Last Converted Data Register (ADC\_LCDDR).

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\_LCDDR clears the DRDY bit and the EOC bit corresponding to the last converted channel.

**Figure 35-2.** EOCx and DRDY Flag Behavior

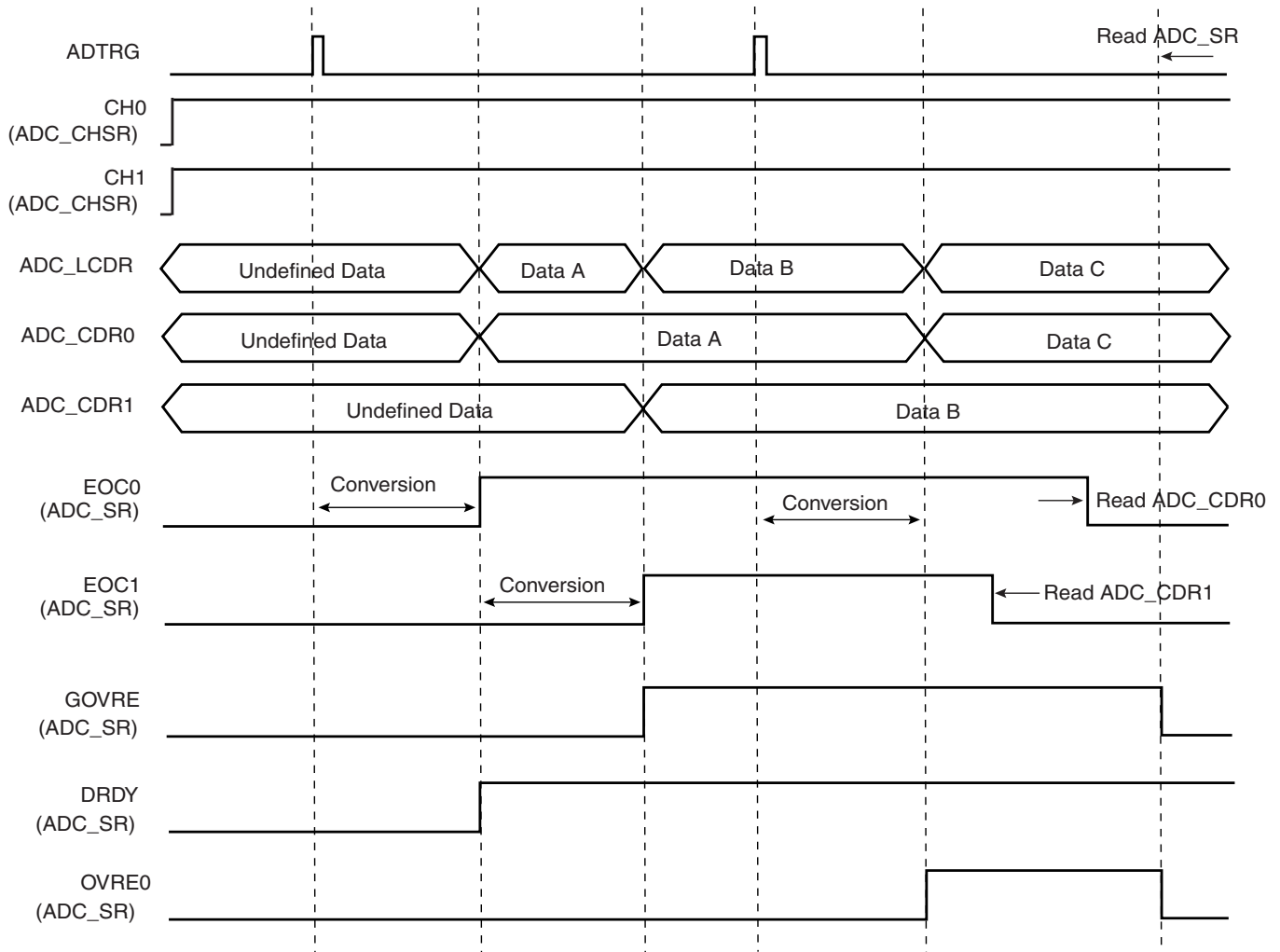


If the ADC\_CDR is not read before further incoming data is converted, the corresponding Overrun Error (OVRE) flag is set in the Status Register (ADC\_SR).

In the same way, new data converted when DRDY is high sets the bit GOVRE (General Overrun Error) in ADC\_SR.

The OVRE and GOVRE flags are automatically cleared when ADC\_SR is read.

**Figure 35-3.** 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.

### 35.5.5 Conversion Triggers

Conversions of the active analog channels are started with a software or a hardware trigger. The software trigger is provided by writing the Control Register (ADC\_CR) with the bit START 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 field TRGSEL in the Mode Register (ADC\_MR). The selected hardware trigger is enabled with the bit TRGEN in the Mode Register (ADC\_MR).

If a hardware trigger is selected, the start of a conversion is detected at each rising edge of the selected signal. If one of the TIOA outputs is selected, the corresponding Timer Counter channel must be programmed in Waveform Mode.

Only one start command is necessary to initiate a conversion sequence on all the channels. The ADC hardware logic automatically performs the conversions on the active channels, then waits for a new request. The Channel Enable (ADC\_CHER) and Channel Disable (ADC\_CHDR) Registers enable the analog channels to be enabled or disabled independently.

If the ADC is used with a PDC, only the transfers of converted data from enabled channels are performed and the resulting data buffers should be interpreted accordingly.

**Warning:** Enabling hardware triggers does not disable the software trigger functionality. Thus, if a hardware trigger is selected, the start of a conversion can be initiated either by the hardware or the software trigger.

### 35.5.6 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 bit SLEEP 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.

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.

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.

Note: The reference voltage pins always remain connected in normal mode as in sleep mode.



## 35.5.7 ADC Timings

Each ADC has its own minimal Startup Time that is programmed through the field STARTUP in the Mode Register ADC\_MR.

In the same way, a minimal Sample and Hold Time is necessary for the ADC to guarantee the best converted final value between two channels selection. This time has to be programmed through the bitfield SHTIM 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 SHTIM field. See the section ADC Characteristics in the product datasheet.

## 35.6 Analog-to-digital Converter (ADC) User Interface

**Table 35-2.** ADC Register Mapping

Offset	Register	Name	Access	Reset State
0x00	Control Register	ADC_CR	Write-only	–
0x04	Mode Register	ADC_MR	Read/Write	0x00000000
0x08	Reserved	–	–	–
0x0C	Reserved	–	–	–
0x10	Channel Enable Register	ADC_CHER	Write-only	–
0x14	Channel Disable Register	ADC_CHDR	Write-only	–
0x18	Channel Status Register	ADC_CHSR	Read-only	0x00000000
0x1C	Status Register	ADC_SR	Read-only	0x000C0000
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	Channel Data Register 0	ADC_CDR0	Read-only	0x00000000
0x34	Channel Data Register 1	ADC_CDR1	Read-only	0x00000000
...	...	...	...	...
0x4C	Channel Data Register 7	ADC_CDR7	Read-only	0x00000000
0x50 - 0xFC	Reserved	–	–	–

## 35.6.1 ADC Control Register

**Register Name:** ADC\_CR

**Access Type:** Write-only

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

### 35.6.2 ADC Mode Register

**Register Name:** ADC\_MR

**Access Type:** Read/Write

31	30	29	28	27	26	25	24	
–	–	–	–	SHTIM				
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	
–	–	SLEEP	LOWRES	TRGSEL			TRGEN	

- **TRGEN: Trigger Enable**

TRGEN	Selected TRGEN
0	Hardware triggers are disabled. Starting a conversion is only possible by software.
1	Hardware trigger selected by TRGSEL field is enabled.

- **TRGSEL: Trigger Selection**

TRGSEL			Selected TRGSEL
0	0	0	TIOA Ouput of the Timer Counter Channel 0
0	0	1	TIOA Ouput of the Timer Counter Channel 1
0	1	0	TIOA Ouput of the Timer Counter Channel 2
0	1	1	TIOA Ouput of the Timer Counter Channel 3
1	0	0	TIOA Ouput of the Timer Counter Channel 4
1	0	1	TIOA Ouput of the Timer Counter Channel 5
1	1	0	External trigger
1	1	1	Reserved

- **LOWRES: Resolution**

LOWRES	Selected Resolution
0	10-bit resolution
1	8-bit resolution

- **SLEEP: Sleep Mode**

SLEEP	Selected Mode
0	Normal Mode
1	Sleep Mode

- **PRESCAL: Prescaler Rate Selection**

$$\text{ADCClock} = \text{MCK} / ( (\text{PRESCAL} + 1) * 2 )$$

- **STARTUP: Start Up Time**

$$\text{Startup Time} = (\text{STARTUP} + 1) * 8 / \text{ADCClock}$$

- **SHTIM: Sample & Hold Time**

$$\text{Sample \& Hold Time} = (\text{SHTIM} + 1) / \text{ADCClock}$$

### 35.6.3 ADC Channel Enable Register

**Register Name:** ADC\_CHER

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0

- **CHx: Channel x Enable**

0 = No effect.

1 = Enables the corresponding channel.

### 35.6.4 ADC Channel Disable Register

**Register Name:** ADC\_CHDR

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0

- **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.

## 35.6.5 ADC Channel Status Register

**Register Name:** ADC\_CHSR

**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
CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0

- **CHx: Channel x Status**

0 = Corresponding channel is disabled.

1 = Corresponding channel is enabled.

## 35.6.6 ADC Status Register

**Register Name:** ADC\_SR

**Access Type:** Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	RXBUFF	ENDRX	GOVRE	DRDY
15	14	13	12	11	10	9	8
OVRE7	OVRE6	OVRE5	OVRE4	OVRE3	OVRE2	OVRE1	OVRE0
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.

1 = Corresponding analog channel is enabled and conversion is complete.

- **OVREx: Overrun Error x**

0 = No overrun error on the corresponding channel since the last read of ADC\_SR.

1 = There has been an overrun error on the corresponding channel since the last read of ADC\_SR.

- **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\_SR.

1 = At least one General Overrun Error has occurred since the last read of ADC\_SR.

- **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.



## 35.6.7 ADC Last Converted Data Register

**Register Name:** ADC\_LCDR

**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
–	–	–	–	–	–	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.

## 35.6.8 ADC Interrupt Enable Register

**Register Name:** ADC\_IER

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	RXBUFF	ENDRX	GOVRE	DRDY
15	14	13	12	11	10	9	8
OVRE7	OVRE6	OVRE5	OVRE4	OVRE3	OVRE2	OVRE1	OVRE0
7	6	5	4	3	2	1	0
EOC7	EOC6	EOC5	EOC4	EOC3	EOC2	EOC1	EOC0

- **EOCx: End of Conversion Interrupt Enable x**
- **OVREx: Overrun Error Interrupt Enable x**
- **DRDY: Data Ready Interrupt Enable**
- **GOVRE: General Overrun Error Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable**
- **RXBUFF: Receive Buffer Full Interrupt Enable**

0 = No effect.

1 = Enables the corresponding interrupt.

## 35.6.9 ADC Interrupt Disable Register

**Register Name:** ADC\_IDR

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	RXBUFF	ENDRX	GOVRE	DRDY
15	14	13	12	11	10	9	8
OVRE7	OVRE6	OVRE5	OVRE4	OVRE3	OVRE2	OVRE1	OVRE0
7	6	5	4	3	2	1	0
EOC7	EOC6	EOC5	EOC4	EOC3	EOC2	EOC1	EOC0

- **EOCx:** End of Conversion Interrupt Disable x
- **OVREx:** Overrun Error Interrupt Disable x
- **DRDY:** Data Ready Interrupt Disable
- **GOVRE:** General Overrun Error Interrupt Disable
- **ENDRX:** End of Receive Buffer Interrupt Disable
- **RXBUFF:** Receive Buffer Full Interrupt Disable

0 = No effect.

1 = Disables the corresponding interrupt.

## 35.6.10 ADC Interrupt Mask Register

**Register Name:** ADC\_IMR

**Access Type:** Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	RXBUFF	ENDRX	GOVRE	DRDY
15	14	13	12	11	10	9	8
OVRE7	OVRE6	OVRE5	OVRE4	OVRE3	OVRE2	OVRE1	OVRE0
7	6	5	4	3	2	1	0
EOC7	EOC6	EOC5	EOC4	EOC3	EOC2	EOC1	EOC0

- **EOCx:** End of Conversion Interrupt Mask x
- **OVREx:** Overrun Error Interrupt Mask x
- **DRDY:** Data Ready Interrupt Mask
- **GOVRE:** General Overrun Error 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.

## 35.6.11 ADC Channel Data Register

**Register Name:** ADC\_CDRx

**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
–	–	–	–	–	–	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.

## **36. Controller Area Network (CAN)**

### **36.1 Overview**

The CAN controller provides all the features required to implement the serial communication protocol CAN defined by Robert Bosch GmbH, the CAN specification as referred to by ISO/11898A (2.0 Part A and 2.0 Part B) for high speeds and ISO/11519-2 for low speeds. The CAN Controller is able to handle all types of frames (Data, Remote, Error and Overload) and achieves a bitrate of 1 Mbit/sec.

CAN controller accesses are made through configuration registers. 16 independent message objects (mailboxes) are implemented.

Any mailbox can be programmed as a reception buffer block (even non-consecutive buffers). For the reception of defined messages, one or several message objects can be masked without participating in the buffer feature. An interrupt is generated when the buffer is full. According to the mailbox configuration, the first message received can be locked in the CAN controller registers until the application acknowledges it, or this message can be discarded by new received messages.

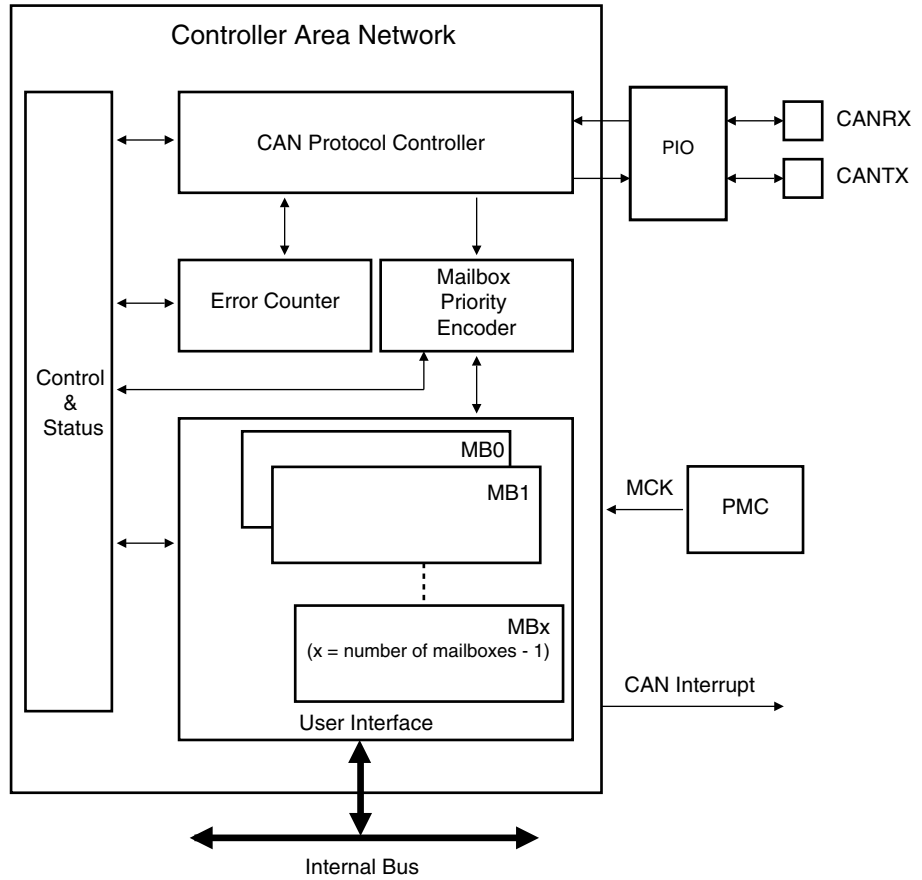
Any mailbox can be programmed for transmission. Several transmission mailboxes can be enabled in the same time. A priority can be defined for each mailbox independently.

An internal 16-bit timer is used to stamp each received and sent message. This timer starts counting as soon as the CAN controller is enabled. This counter can be reset by the application or automatically after a reception in the last mailbox in Time Triggered Mode.

The CAN controller offers optimized features to support the Time Triggered Communication (TTC) protocol.

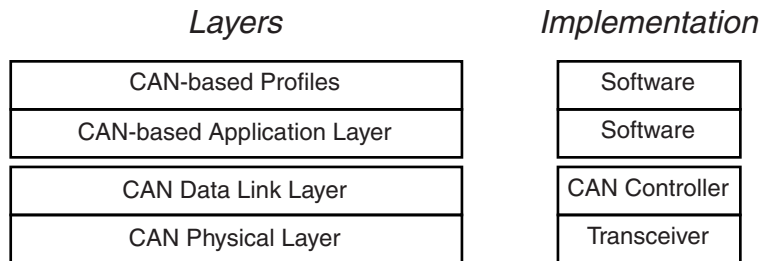
### 36.2 Block Diagram

Figure 36-1. CAN Block Diagram



### 36.3 Application Block Diagram

Figure 36-2. Application Block Diagram



## 36.4 I/O Lines Description

**Table 36-1.** I/O Lines Description

Name	Description	Type
CANRX	CAN Receive Serial Data	Input
CANTX	CAN Transmit Serial Data	Output

## 36.5 Product Dependencies

### 36.5.1 I/O Lines

The pins used for interfacing the CAN may be multiplexed with the PIO lines. The programmer must first program the PIO controller to assign the desired CAN pins to their peripheral function. If I/O lines of the CAN are not used by the application, they can be used for other purposes by the PIO Controller.

### 36.5.2 Power Management

The programmer must first enable the CAN clock in the Power Management Controller (PMC) before using the CAN.

A Low-power Mode is defined for the CAN controller: If the application does not require CAN operations, the CAN clock can be stopped when not needed and be restarted later. Before stopping the clock, the CAN Controller must be in Low-power Mode to complete the current transfer. After restarting the clock, the application must disable the Low-power Mode of the CAN controller.

### 36.5.3 Interrupt

The CAN interrupt line is connected on one of the internal sources of the Advanced Interrupt Controller. Using the CAN interrupt requires the AIC to be programmed first. Note that it is not recommended to use the CAN interrupt line in edge-sensitive mode.

## 36.6 CAN Controller Features

### 36.6.1 CAN Protocol Overview

The Controller Area Network (CAN) is a multi-master serial communication protocol that efficiently supports real-time control with a very high level of security with bit rates up to 1 Mbit/s.

The CAN protocol supports four different frame types:

- Data frames: They carry data from a transmitter node to the receiver nodes. The overall maximum data frame length is 108 bits for a standard frame and 128 bits for an extended frame.
- Remote frames: A destination node can request data from the source by sending a remote frame with an identifier that matches the identifier of the required data frame. The appropriate data source node then sends a data frame as a response to this node request.
- Error frames: An error frame is generated by any node that detects a bus error.
- Overload frames: They provide an extra delay between the preceding and the successive data frames or remote frames.

The Atmel CAN controller provides the CPU with full functionality of the CAN protocol V2.0 Part A and V2.0 Part B. It minimizes the CPU load in communication overhead. The Data Link Layer and part of the physical layer are automatically handled by the CAN controller itself.

The CPU reads or writes data or messages via the CAN controller mailboxes. An identifier is assigned to each mailbox. The CAN controller encapsulates or decodes data messages to build or to decode bus data frames. Remote frames, error frames and overload frames are automatically handled by the CAN controller under supervision of the software application.

### 36.6.2 Mailbox Organization

The CAN module has 16 buffers, also called channels or mailboxes. An identifier that corresponds to the CAN identifier is defined for each active mailbox. Message identifiers can match the standard frame identifier or the extended frame identifier. This identifier is defined for the first time during the CAN initialization, but can be dynamically reconfigured later so that the mailbox can handle a new message family. Several mailboxes can be configured with the same ID.

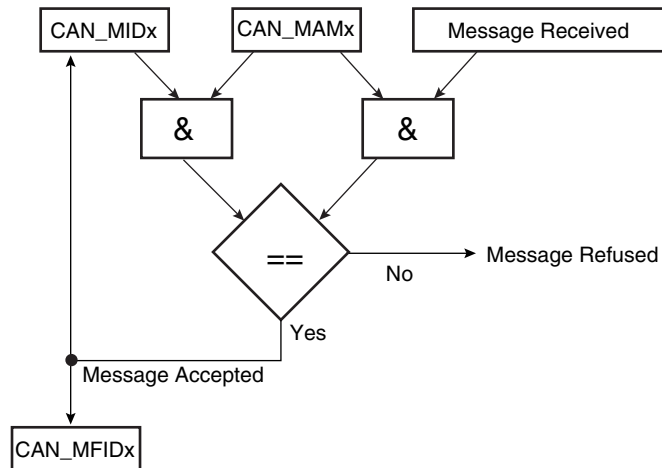
Each mailbox can be configured in receive or in transmit mode independently. The mailbox object type is defined in the MOT field of the CAN\_MMRx register.

#### 36.6.2.1 Message Acceptance Procedure

If the MIDE field in the CAN\_MIDx register is set, the mailbox can handle the extended format identifier; otherwise, the mailbox handles the standard format identifier. Once a new message is received, its ID is masked with the CAN\_MAMx value and compared with the CAN\_MIDx value. If accepted, the message ID is copied to the CAN\_MIDx register.



**Figure 36-3. Message Acceptance Procedure**



If a mailbox is dedicated to receiving several messages (a family of messages) with different IDs, the acceptance mask defined in the CAN\_MAMx register must mask the variable part of the ID family. Once a message is received, the application must decode the masked bits in the CAN\_MIDx. To speed up the decoding, masked bits are grouped in the family ID register (CAN\_MFIDx).

For example, if the following message IDs are handled by the same mailbox:

```

ID0 101000100100010010000100 0 11 00b
ID1 101000100100010010000100 0 11 01b
ID2 101000100100010010000100 0 11 10b
ID3 101000100100010010000100 0 11 11b
ID4 101000100100010010000100 1 11 00b
ID5 101000100100010010000100 1 11 01b
ID6 101000100100010010000100 1 11 10b
ID7 101000100100010010000100 1 11 11b
  
```

The CAN\_MIDx and CAN\_MAMx of Mailbox x must be initialized to the corresponding values:

```

CAN_MIDx = 001 101000100100010010000100 x 11 xxb
CAN_MAMx = 001 111111111111111111111111 0 11 00b
  
```

If Mailbox x receives a message with ID6, then CAN\_MIDx and CAN\_MFIDx are set:

```

CAN_MIDx = 001 101000100100010010000100 1 11 10b
CAN_MFIDx = 00000000000000000000000000000110b
  
```

If the application associates a handler for each message ID, it may define an array of pointers to functions:

```
void (*pHandler[8])(void);
```

When a message is received, the corresponding handler can be invoked using CAN\_MFIDx register and there is no need to check masked bits:

```

unsigned int MFID0_register;
MFID0_register = Get_CAN_MFID0_Register();
// Get_CAN_MFID0_Register() returns the value of the CAN_MFID0 register
pHandler[MFID0_register] ();
  
```

## 36.6.2.2 Receive Mailbox

When the CAN module receives a message, it looks for the first available mailbox with the lowest number and compares the received message ID with the mailbox ID. If such a mailbox is found, then the message is stored in its data registers. Depending on the configuration, the mailbox is disabled as long as the message has not been acknowledged by the application (Receive only), or, if new messages with the same ID are received, then they overwrite the previous ones (Receive with overwrite).

It is also possible to configure a mailbox in Consumer Mode. In this mode, after each transfer request, a remote frame is automatically sent. The first answer received is stored in the corresponding mailbox data registers.

Several mailboxes can be chained to receive a buffer. They must be configured with the same ID in Receive Mode, except for the last one, which can be configured in Receive with Overwrite Mode. The last mailbox can be used to detect a buffer overflow.

**Table 36-2.**

Mailbox Object Type	Description
Receive	The first message received is stored in mailbox data registers. Data remain available until the next transfer request.
Receive with overwrite	The last message received is stored in mailbox data register. The next message always overwrites the previous one. The application has to check whether a new message has not overwritten the current one while reading the data registers.
Consumer	A remote frame is sent by the mailbox. The answer received is stored in mailbox data register. This extends Receive mailbox features. Data remain available until the next transfer request.

## 36.6.2.3 Transmit Mailbox

When transmitting a message, the message length and data are written to the transmit mailbox with the correct identifier. For each transmit mailbox, a priority is assigned. The controller automatically sends the message with the highest priority first (set with the field PRIOR in CAN\_MMRx register).

It is also possible to configure a mailbox in Producer Mode. In this mode, when a remote frame is received, the mailbox data are sent automatically. By enabling this mode, a producer can be done using only one mailbox instead of two: one to detect the remote frame and one to send the answer.

**Table 36-3.**

Mailbox Object Type	Description
Transmit	The message stored in the mailbox data registers will try to win the bus arbitration immediately or later according to or not the Time Management Unit configuration (see <a href="#">Section 36.6.3</a> ). The application is notified that the message has been sent or aborted.
Producer	The message prepared in the mailbox data registers will be sent after receiving the next remote frame. This extends transmit mailbox features.

## 36.6.3 Time Management Unit

The CAN Controller integrates a free-running 16-bit internal timer. The counter is driven by the bit clock of the CAN bus line. It is enabled when the CAN controller is enabled (CANEN set in the CAN\_MR register). It is automatically cleared in the following cases:

- after a reset
- when the CAN controller is in Low-power Mode is enabled (LPM bit set in the CAN\_MR and SLEEP bit set in the CAN\_SR)
- after a reset of the CAN controller (CANEN bit in the CAN\_MR register)
- in Time-triggered Mode, when a message is accepted by the last mailbox (rising edge of the MRDY signal in the CAN\_MSR<sub>last\_mailbox\_number</sub> register).

The application can also reset the internal timer by setting TIMRST in the CAN\_TCR register. The current value of the internal timer is always accessible by reading the CAN\_TIM register.

When the timer rolls-over from FFFFh to 0000h, TOVF (Timer Overflow) signal in the CAN\_SR register is set. TOVF bit in the CAN\_SR register is cleared by reading the CAN\_SR register. Depending on the corresponding interrupt mask in the CAN\_IMR register, an interrupt is generated while TOVF is set.

In a CAN network, some CAN devices may have a larger counter. In this case, the application can also decide to freeze the internal counter when the timer reaches FFFFh and to wait for a restart condition from another device. This feature is enabled by setting TIMFRZ in the CAN\_MR register. The CAN\_TIM register is frozen to the FFFFh value. A clear condition described above restarts the timer. A timer overflow (TOVF) interrupt is triggered.

To monitor the CAN bus activity, the CAN\_TIM register is copied to the CAN\_TIMESTP register after each start of frame or end of frame and a TSTP interrupt is triggered. If TEOF bit in the CAN\_MR register is set, the value is captured at each End Of Frame, else it is captured at each Start Of Frame. Depending on the corresponding mask in the CAN\_IMR register, an interrupt is generated while TSTP is set in the CAN\_SR. TSTP bit is cleared by reading the CAN\_SR register.

The time management unit can operate in one of the two following modes:

- Timestamping mode: The value of the internal timer is captured at each Start Of Frame or each End Of Frame
- Time Triggered mode: A mailbox transfer operation is triggered when the internal timer reaches the mailbox trigger.

Timestamping Mode is enabled by clearing TTM field in the CAN\_MR register. Time Triggered Mode is enabled by setting TTM field in the CAN\_MR register.

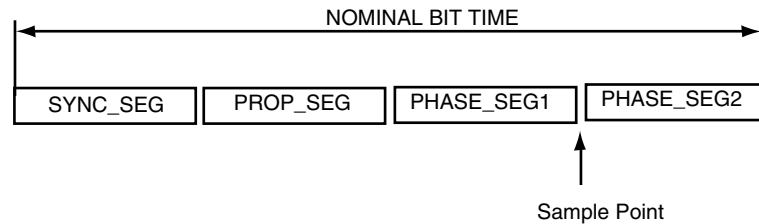
## 36.6.4 CAN 2.0 Standard Features

### 36.6.4.1 CAN Bit Timing Configuration

All controllers on a CAN bus must have the same bit rate and bit length. At different clock frequencies of the individual controllers, the bit rate has to be adjusted by the time segments.

The CAN protocol specification partitions the nominal bit time into four different segments:

**Figure 36-4.** Partition of the CAN Bit Time



#### TIME QUANTUM:

The TIME QUANTUM (TQ) is a fixed unit of time derived from the MCK period. The total number of TIME QUANTA in a bit time is programmable from 8 to 25.

**SYNC SEG:** SYNChronization Segment.

This part of the bit time is used to synchronize the various nodes on the bus. An edge is expected to lie within this segment. It is 1 TQ long.

**PROP SEG:** PROPagation Segment.

This part of the bit time is used to compensate for the physical delay times within the network. It is twice the sum of the signal's propagation time on the bus line, the input comparator delay, and the output driver delay. It is programmable to be 1,2,..., 8 TQ long.

This parameter is defined in the PROPAG field of the "[CAN Baudrate Register](#)".

**PHASE SEG1, PHASE SEG2:** PHASE Segment 1 and 2.

The Phase-Buffer-Segments are used to compensate for edge phase errors. These segments can be lengthened (PHASE SEG1) or shortened (PHASE SEG2) by resynchronization.

Phase Segment 1 is programmable to be 1,2,..., 8 TQ long.

Phase Segment 2 length has to be at least as long as the Information Processing Time (IPT) and may not be more than the length of Phase Segment 1.

These parameters are defined in the PHASE1 and PHASE2 fields of the "[CAN Baudrate Register](#)".

#### INFORMATION PROCESSING TIME:

The Information Processing Time (IPT) is the time required for the logic to determine the bit level of a sampled bit. The IPT begins at the sample point, is measured in TQ and **is fixed at 2 TQ for the Atmel CAN**. Since Phase Segment 2 also begins at the sample point and is the last segment in the bit time, PHASE SEG2 shall not be less than the IPT.

## SAMPLE POINT:

The SAMPLE POINT is the point in time at which the bus level is read and interpreted as the value of that respective bit. Its location is at the end of PHASE\_SEG1.

SJW: ReSynchronization Jump Width.

The ReSynchronization Jump Width defines the limit to the amount of lengthening or shortening of the Phase Segments.

SJW is programmable to be the minimum of PHASE\_SEG1 and 4 TQ.

If the SMP field in the CAN\_BR register is set, then the incoming bit stream is sampled three times with a period of half a CAN clock period, centered on sample point.

In the CAN controller, the length of a bit on the CAN bus is determined by the parameters (BRP, PROPAG, PHASE1 and PHASE2).

$$t_{\text{BIT}} = t_{\text{CSC}} + t_{\text{PRS}} + t_{\text{PHS1}} + t_{\text{PHS2}}$$

The time quantum is calculated as follows:

$$t_{\text{CSC}} = (\text{BRP} + 1) / \text{MCK}$$

**Note:** The BRP field must be within the range [1, 0x7F], i.e., BRP = 0 is not authorized.

$$t_{\text{PRS}} = t_{\text{CSC}} \times (\text{PROPAG} + 1)$$

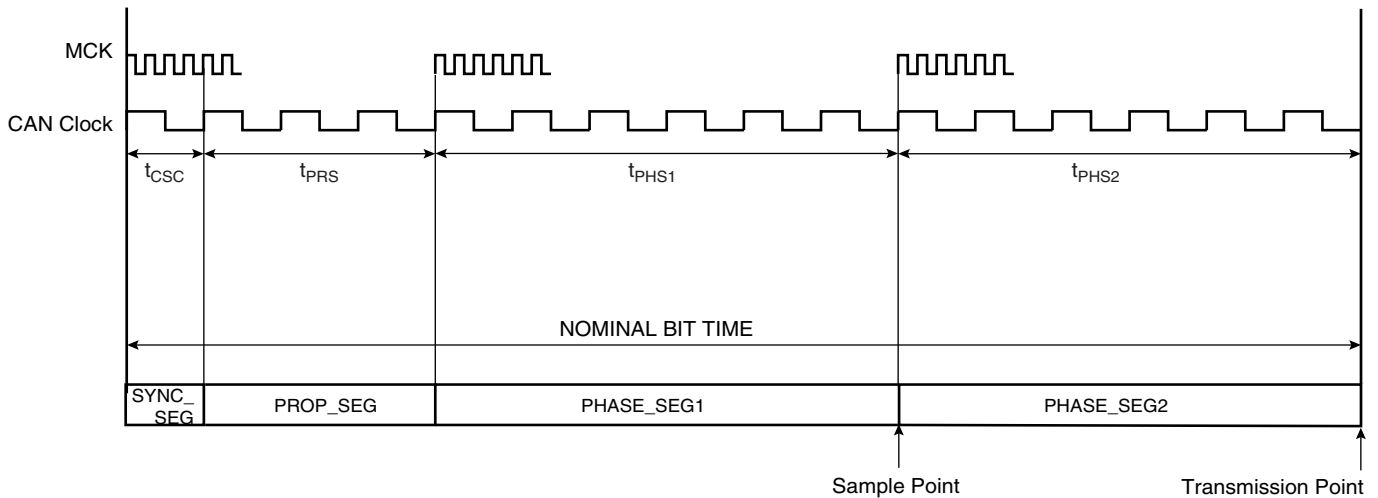
$$t_{\text{PHS1}} = t_{\text{CSC}} \times (\text{PHASE1} + 1)$$

$$t_{\text{PHS2}} = t_{\text{CSC}} \times (\text{PHASE2} + 1)$$

To compensate for phase shifts between clock oscillators of different controllers on the bus, the CAN controller must resynchronize on any relevant signal edge of the current transmission. The resynchronization shortens or lengthens the bit time so that the position of the sample point is shifted with regard to the detected edge. The resynchronization jump width (SJW) defines the maximum of time by which a bit period may be shortened or lengthened by resynchronization.

$$t_{\text{SJW}} = t_{\text{CSC}} \times (\text{SJW} + 1)$$

Figure 36-5. CAN Bit Timing



Example of bit timing determination for CAN baudrate of 500 Kbit/s:

```
MCK = 48MHz
CAN baudrate= 500kbit/s => bit time= 2us
Delay of the bus driver: 50 ns
Delay of the receiver: 30ns
Delay of the bus line (20m): 110ns
```

The total number of time quanta in a bit time must be comprised between 8 and 25. If we fix the bit time to 16 time quanta:

```
Tcsc = 1 time quanta = bit time / 16 = 125 ns
=> BRP = (Tcsc x MCK) - 1 = 5
```

The propagation segment time is equal to twice the sum of the signal's propagation time on the bus line, the receiver delay and the output driver delay:

```
Tprs = 2 * (50+30+110) ns = 380 ns = 3 Tcsc
=> PROPAG = Tprs/Tcsc - 1 = 2
```

The remaining time for the two phase segments is:

```
Tphs1 + Tphs2 = bit time - Tcsc - Tprs = (16 - 1 - 3)Tcsc
Tphs1 + Tphs2 = 12 Tcsc
```

Because this number is even, we choose  $T_{phs2} = T_{phs1}$  (else we would choose  $T_{phs2} = T_{phs1} + T_{csc}$ )

```
Tphs1 = Tphs2 = (12/2) Tcsc = 6 Tcsc
=> PHASE1 = PHASE2 = Tphs1/Tcsc - 1 = 5
```

The resynchronization jump width must be comprised between 1  $T_{csc}$  and the minimum of 4  $T_{csc}$  and  $T_{phs1}$ . We choose its maximum value:

```
Tsjw = Min(4 Tcsc, Tphs1) = 4 Tcsc
=> SJW = Tsjw/Tcsc - 1 = 3
```

Finally: CAN\_BR = 0x00053255

## CAN Bus Synchronization

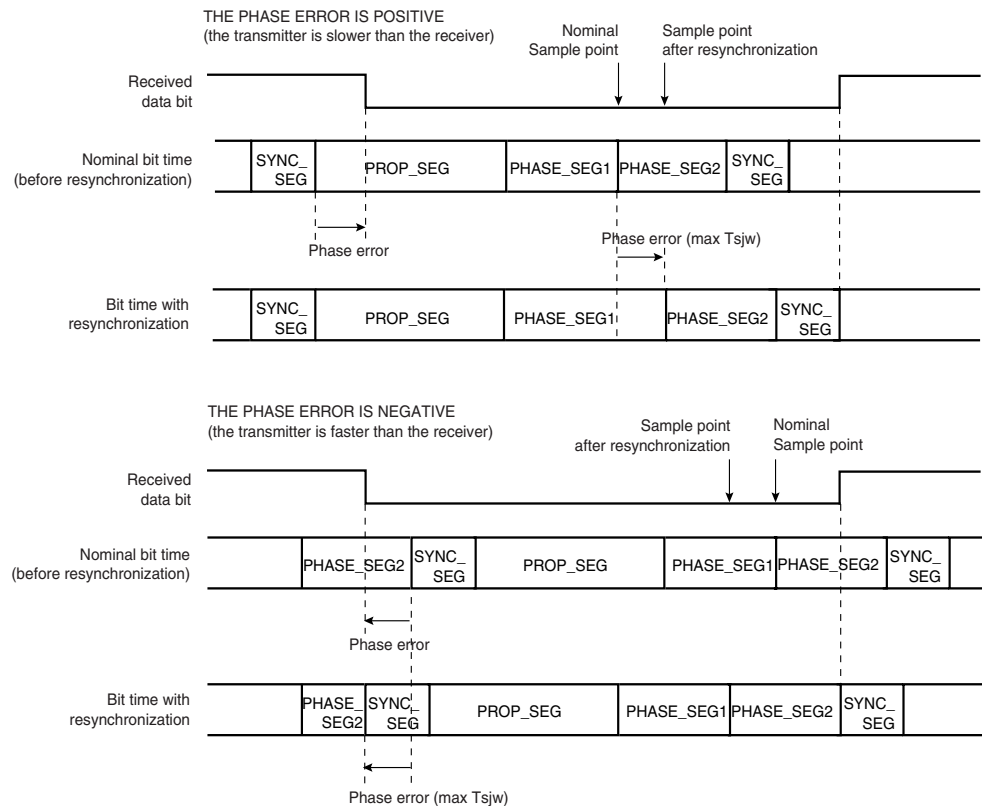
Two types of synchronization are distinguished: “hard synchronization” at the start of a frame and “resynchronization” inside a frame. After a hard synchronization, the bit time is restarted with the end of the SYNC\_SEG segment, regardless of the phase error. Resynchronization causes a reduction or increase in the bit time so that the position of the sample point is shifted with respect to the detected edge.

The effect of resynchronization is the same as that of hard synchronization when the magnitude of the phase error of the edge causing the resynchronization is less than or equal to the programmed value of the resynchronization jump width ( $t_{sjw}$ ).

When the magnitude of the phase error is larger than the resynchronization jump width and

- the phase error is positive, then PHASE\_SEG1 is lengthened by an amount equal to the resynchronization jump width.
- the phase error is negative, then PHASE\_SEG2 is shortened by an amount equal to the resynchronization jump width.

**Figure 36-6. CAN Resynchronization**



## Autobaud Mode

The autobaud feature is enabled by setting the ABM field in the CAN\_MR register. In this mode, the CAN controller is only listening to the line without acknowledging the received messages. It can not send any message. The errors flags are updated. The bit timing can be adjusted until no error occurs (good configuration found). In this mode, the error counters are frozen. To go back to the standard mode, the ABM bit must be cleared in the CAN\_MR register.

## 36.6.4.2 Error Detection

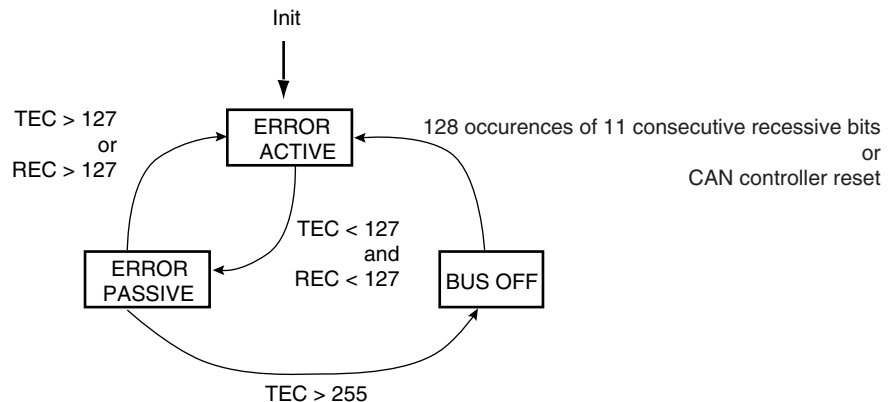
There are five different error types that are not mutually exclusive. Each error concerns only specific fields of the CAN data frame (refer to the Bosch CAN specification for their correspondence):

- CRC error (CERR bit in the CAN\_SR register): With the CRC, the transmitter calculates a checksum for the CRC bit sequence from the Start of Frame bit until the end of the Data Field. This CRC sequence is transmitted in the CRC field of the Data or Remote Frame.
- Bit-stuffing error (SERR bit in the CAN\_SR register): If a node detects a sixth consecutive equal bit level during the bit-stuffing area of a frame, it generates an Error Frame starting with the next bit-time.
- Bit error (BERR bit in CAN\_SR register): A bit error occurs if a transmitter sends a dominant bit but detects a recessive bit on the bus line, or if it sends a recessive bit but detects a dominant bit on the bus line. An error frame is generated and starts with the next bit time.
- Form Error (FERR bit in the CAN\_SR register): If a transmitter detects a dominant bit in one of the fix-formatted segments CRC Delimiter, ACK Delimiter or End of Frame, a form error has occurred and an error frame is generated.
- Acknowledgment error (AERR bit in the CAN\_SR register): The transmitter checks the Acknowledge Slot, which is transmitted by the transmitting node as a recessive bit, contains a dominant bit. If this is the case, at least one other node has received the frame correctly. If not, an Acknowledge Error has occurred and the transmitter will start in the next bit-time an Error Frame transmission.

## Fault Confinement

To distinguish between temporary and permanent failures, every CAN controller has two error counters: REC (Receive Error Counter) and TEC (Transmit Error Counter). The counters are incremented upon detected errors and respectively are decremented upon correct transmissions or receptions. Depending on the counter values, the state of the node changes: the initial state of the CAN controller is Error Active, meaning that the controller can send Error Active flags. The controller changes to the Error Passive state if there is an accumulation of errors. If the CAN controller fails or if there is an extreme accumulation of errors, there is a state transition to Bus Off.

**Figure 36-7.** Line Error Mode





An error active unit takes part in bus communication and sends an active error frame when the CAN controller detects an error.

An error passive unit cannot send an active error frame. It takes part in bus communication, but when an error is detected, a passive error frame is sent. Also, after a transmission, an error passive unit waits before initiating further transmission.

A bus off unit is not allowed to have any influence on the bus.

For fault confinement, two errors counters (TEC and REC) are implemented. These counters are accessible via the CAN\_ECR register. The state of the CAN controller is automatically updated according to these counter values. If the CAN controller is in Error Active state, then the ERRA bit is set in the CAN\_SR register. The corresponding interrupt is pending while the interrupt is not masked in the CAN\_IMR register. If the CAN controller is in Error Passive Mode, then the ERRP bit is set in the CAN\_SR register and an interrupt remains pending while the ERRP bit is set in the CAN\_IMR register. If the CAN is in Bus-off Mode, then the BOFF bit is set in the CAN\_SR register. As for ERRP and ERRA, an interrupt is pending while the BOFF bit is set in the CAN\_IMR register.

When one of the error counters values exceeds 96, an increased error rate is indicated to the controller through the WARN bit in CAN\_SR register, but the node remains error active. The corresponding interrupt is pending while the interrupt is set in the CAN\_IMR register.

Refer to the Bosch CAN specification v2.0 for details on fault confinement.

### 36.6.4.3 *Overload*

The overload frame is provided to request a delay of the next data or remote frame by the receiver node (“Request overload frame”) or to signal certain error conditions (“Reactive overload frame”) related to the intermission field respectively.

Reactive overload frames are transmitted after detection of the following error conditions:

- Detection of a dominant bit during the first two bits of the intermission field
- Detection of a dominant bit in the last bit of EOF by a receiver, or detection of a dominant bit by a receiver or a transmitter at the last bit of an error or overload frame delimiter

The CAN controller can generate a request overload frame automatically after each message sent to one of the CAN controller mailboxes. This feature is enabled by setting the OVL bit in the CAN\_MR register.

Reactive overload frames are automatically handled by the CAN controller even if the OVL bit in the CAN\_MR register is not set. An overload flag is generated in the same way as an error flag, but error counters do not increment.

### 36.6.5 **Low-power Mode**

In Low-power Mode, the CAN controller cannot send or receive messages. All mailboxes are inactive.

In Low-power Mode, the SLEEP signal in the CAN\_SR register is set; otherwise, the WAKEUP signal in the CAN\_SR register is set. These two fields are exclusive except after a CAN controller reset (WAKEUP and SLEEP are stuck at 0 after a reset). After power-up reset, the Low-power Mode is disabled and the WAKEUP bit is set in the CAN\_SR register only after detection of 11 consecutive recessive bits on the bus.

## 36.6.5.1 Enabling Low-power Mode

A software application can enable Low-power Mode by setting the LPM bit in the CAN\_MR global register. The CAN controller enters Low-power Mode once all pending transmit messages are sent.

When the CAN controller enters Low-power Mode, the SLEEP signal in the CAN\_SR register is set. Depending on the corresponding mask in the CAN\_IMR register, an interrupt is generated while SLEEP is set.

The SLEEP signal in the CAN\_SR register is automatically cleared once WAKEUP is set. The WAKEUP signal is automatically cleared once SLEEP is set.

Reception is disabled while the SLEEP signal is set to one in the CAN\_SR register. It is important to note that those messages with higher priority than the last message transmitted can be received between the LPM command and entry in Low-power Mode.

Once in Low-power Mode, the CAN controller clock can be switched off by programming the chip's Power Management Controller (PMC). The CAN controller drains only the static current.

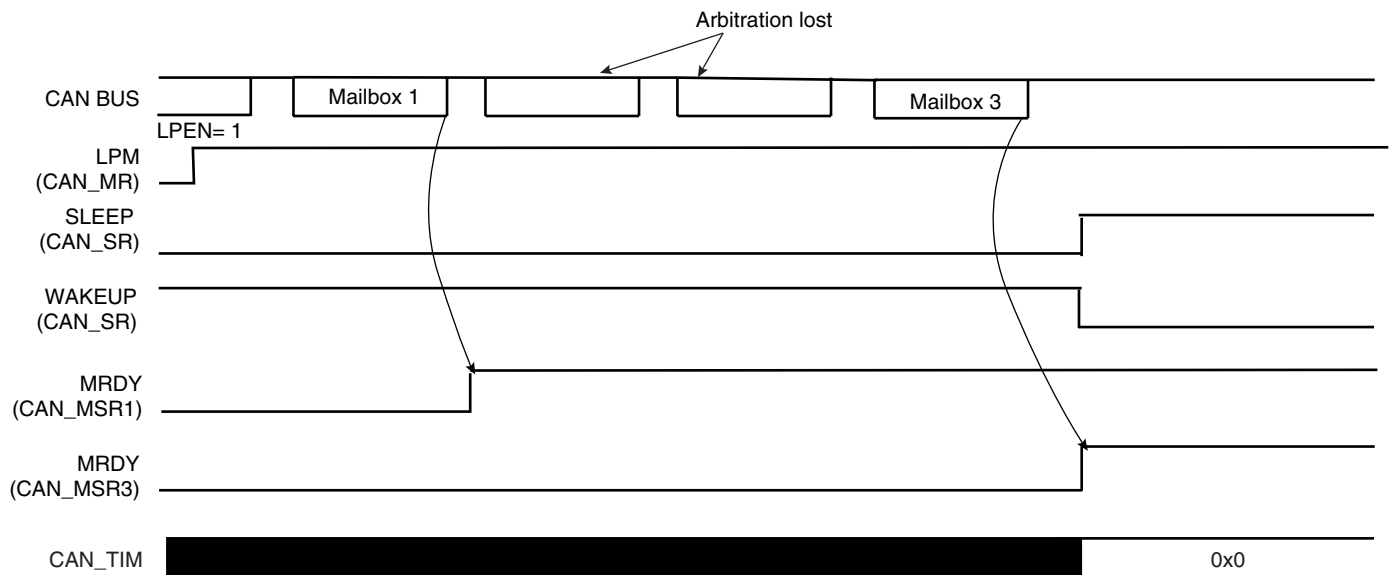
Error counters are disabled while the SLEEP signal is set to one.

Thus, to enter Low-power Mode, the software application must:

- Set LPM field in the CAN\_MR register
- Wait for SLEEP signal rising

Now the CAN Controller clock can be disabled. This is done by programming the Power Management Controller (PMC).

**Figure 36-8.** Enabling Low-power Mode



## 36.6.5.2 Disabling Low-power Mode

The CAN controller can be awake after detecting a CAN bus activity. Bus activity detection is done by an external module that may be embedded in the chip. When it is notified of a CAN bus activity, the software application disables Low-power Mode by programming the CAN controller.

To disable Low-power Mode, the software application must:

- Enable the CAN Controller clock. This is done by programming the Power Management Controller (PMC).
- Clear the LPM field in the CAN\_MR register

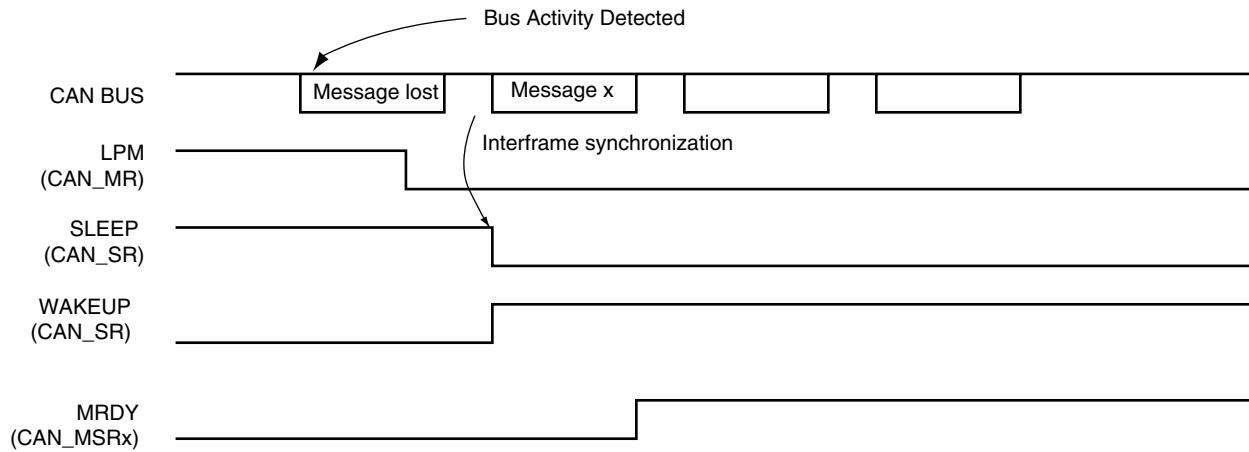
The CAN controller synchronizes itself with the bus activity by checking for eleven consecutive “recessive” bits. Once synchronized, the WAKEUP signal in the CAN\_SR register is set.

Depending on the corresponding mask in the CAN\_IMR register, an interrupt is generated while WAKEUP is set. The SLEEP signal in the CAN\_SR register is automatically cleared once WAKEUP is set. WAKEUP signal is automatically cleared once SLEEP is set.

If no message is being sent on the bus, then the CAN controller is able to send a message eleven bit times after disabling Low-power Mode.

If there is bus activity when Low-power mode is disabled, the CAN controller is synchronized with the bus activity in the next interframe. The previous message is lost (see [Figure 36-9](#)).

**Figure 36-9.** Disabling Low-power Mode



## 36.7 Functional Description

### 36.7.1 CAN Controller Initialization

After power-up reset, the CAN controller is disabled. The CAN controller clock must be activated by the Power Management Controller (PMC) and the CAN controller interrupt line must be enabled by the interrupt controller (AIC).

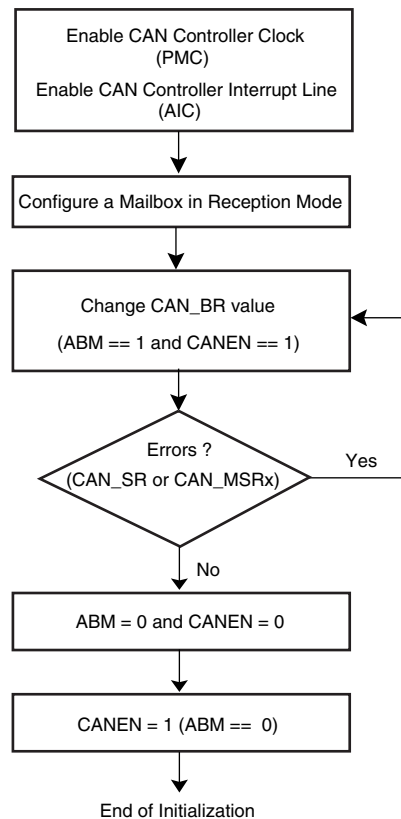
The CAN controller must be initialized with the CAN network parameters. The CAN\_BR register defines the sampling point in the bit time period. CAN\_BR must be set before the CAN controller is enabled by setting the CANEN field in the CAN\_MR register.

The CAN controller is enabled by setting the CANEN flag in the CAN\_MR register. At this stage, the internal CAN controller state machine is reset, error counters are reset to 0, error flags are reset to 0.

Once the CAN controller is enabled, bus synchronization is done automatically by scanning eleven recessive bits. The WAKEUP bit in the CAN\_SR register is automatically set to 1 when the CAN controller is synchronized (WAKEUP and SLEEP are stuck at 0 after a reset).

The CAN controller can start listening to the network in Autobaud Mode. In this case, the error counters are locked and a mailbox may be configured in Receive Mode. By scanning error flags, the CAN\_BR register values synchronized with the network. Once no error has been detected, the application disables the Autobaud Mode, clearing the ABM field in the CAN\_MR register.

**Figure 36-10.** Possible Initialization Procedure



## 36.7.2 CAN Controller Interrupt Handling

There are two different types of interrupts. One type of interrupt is a message-object related interrupt, the other is a system interrupt that handles errors or system-related interrupt sources.

All interrupt sources can be masked by writing the corresponding field in the CAN\_IDR register. They can be unmasked by writing to the CAN\_IER register. After a power-up reset, all interrupt sources are disabled (masked). The current mask status can be checked by reading the CAN\_IMR register.

The CAN\_SR register gives all interrupt source states.

The following events may initiate one of the two interrupts:

- Message object interrupt
  - Data registers in the mailbox object are available to the application. In Receive Mode, a new message was received. In Transmit Mode, a message was transmitted successfully.
  - A sent transmission was aborted.
- System interrupts
  - Bus-off interrupt: The CAN module enters the bus-off state.
  - Error-passive interrupt: The CAN module enters Error Passive Mode.
  - Error-active Mode: The CAN module is neither in Error Passive Mode nor in Bus-off mode.
  - Warn Limit interrupt: The CAN module is in Error-active Mode, but at least one of its error counter value exceeds 96.
  - Wake-up interrupt: This interrupt is generated after a wake-up and a bus synchronization.
  - Sleep interrupt: This interrupt is generated after a Low-power Mode enable once all pending messages in transmission have been sent.
  - Internal timer counter overflow interrupt: This interrupt is generated when the internal timer rolls over.
  - Timestamp interrupt: This interrupt is generated after the reception or the transmission of a start of frame or an end of frame. The value of the internal counter is copied in the CAN\_TIMESTP register.

All interrupts are cleared by clearing the interrupt source except for the internal timer counter overflow interrupt and the timestamp interrupt. These interrupts are cleared by reading the CAN\_SR register.

## 36.7.3 CAN Controller Message Handling

### 36.7.3.1 Receive Handling

Two modes are available to configure a mailbox to receive messages. In **Receive Mode**, the first message received is stored in the mailbox data register. In **Receive with Overwrite Mode**, the last message received is stored in the mailbox.

#### Simple Receive Mailbox

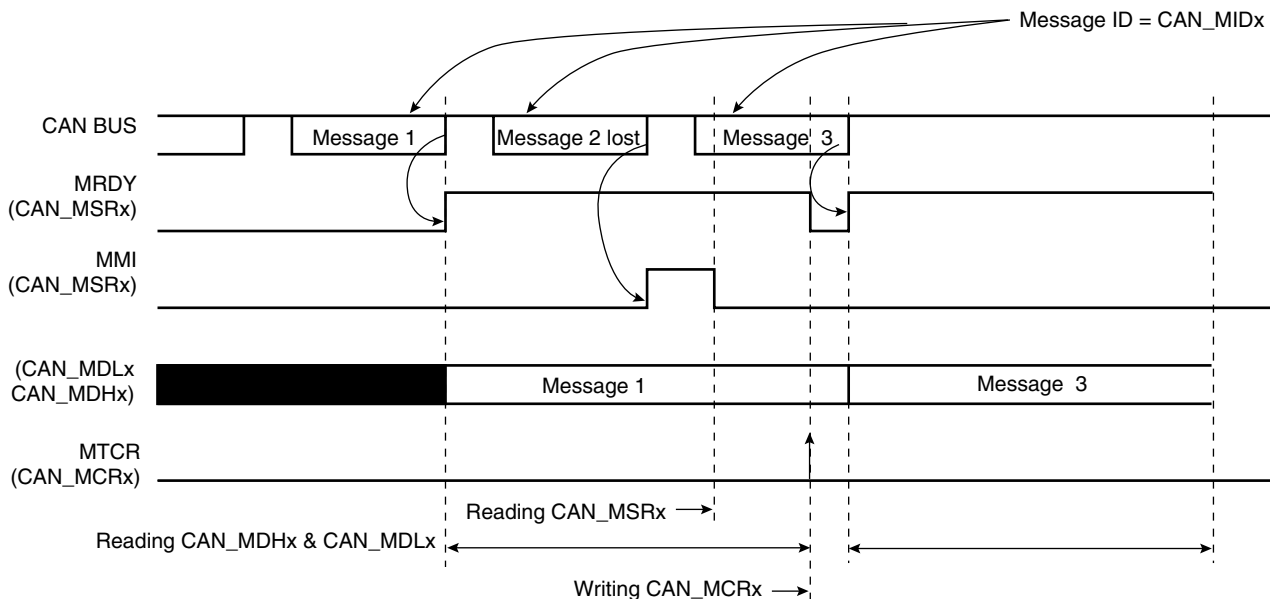
A mailbox is in Receive Mode once the MOT field in the CAN\_MMRx register has been configured. Message ID and Message Acceptance Mask must be set before the Receive Mode is enabled.

After Receive Mode is enabled, the MRDY flag in the CAN\_MSR register is automatically cleared until the first message is received. When the first message has been accepted by the mailbox, the MRDY flag is set. An interrupt is pending for the mailbox while the MRDY flag is set. This interrupt can be masked depending on the mailbox flag in the CAN\_IMR global register.

Message data are stored in the mailbox data register until the software application notifies that data processing has ended. This is done by asking for a new transfer command, setting the MTCR flag in the CAN\_MCRx register. This automatically clears the MRDY signal.

The MMI flag in the CAN\_MSRx register notifies the software that a message has been lost by the mailbox. This flag is set when messages are received while MRDY is set in the CAN\_MSRx register. This flag is cleared by reading the CAN\_MSRs register. A receive mailbox prevents from overwriting the first message by new ones while MRDY flag is set in the CAN\_MSRx register. See [Figure 36-11](#).

**Figure 36-11.** Receive Mailbox



**Note:** In the case of ARM architecture, CAN\_MSRx, CAN\_MDLx, CAN\_MDHx can be read using an optimized Idm assembler instruction.

## Receive with Overwrite Mailbox

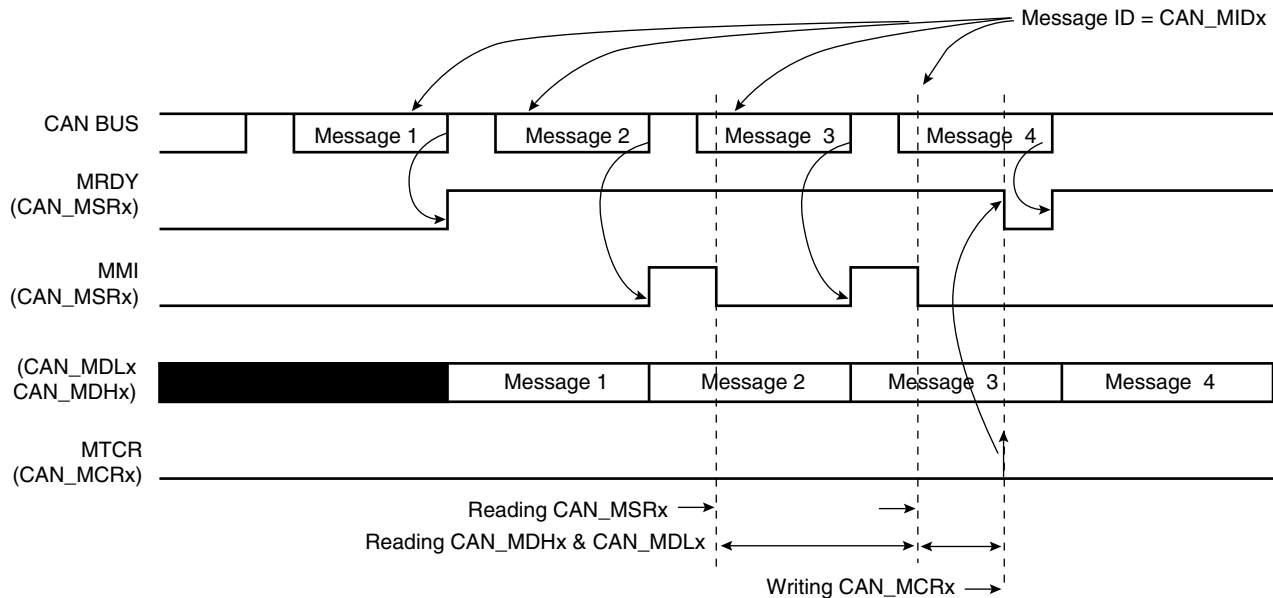
A mailbox is in Receive with Overwrite Mode once the MOT field in the CAN\_MMRx register has been configured. Message ID and Message Acceptance masks must be set before Receive Mode is enabled.

After Receive Mode is enabled, the MRDY flag in the CAN\_MSR register is automatically cleared until the first message is received. When the first message has been accepted by the mailbox, the MRDY flag is set. An interrupt is pending for the mailbox while the MRDY flag is set. This interrupt is masked depending on the mailbox flag in the CAN\_IMR global register.

If a new message is received while the MRDY flag is set, this new message is stored in the mailbox data register, overwriting the previous message. The MMI flag in the CAN\_MSRx register notifies the software that a message has been dropped by the mailbox. This flag is cleared when reading the CAN\_MSRx register.

The CAN controller may store a new message in the CAN data registers while the application reads them. To check that CAN\_MDHx and CAN\_MDLx do not belong to different messages, the application must check the MMI field in the CAN\_MSRx register before and after reading CAN\_MDHx and CAN\_MDLx. If the MMI flag is set again after the data registers have been read, the software application has to re-read CAN\_MDHx and CAN\_MDLx (see Figure 36-12).

**Figure 36-12.** Receive with Overwrite Mailbox

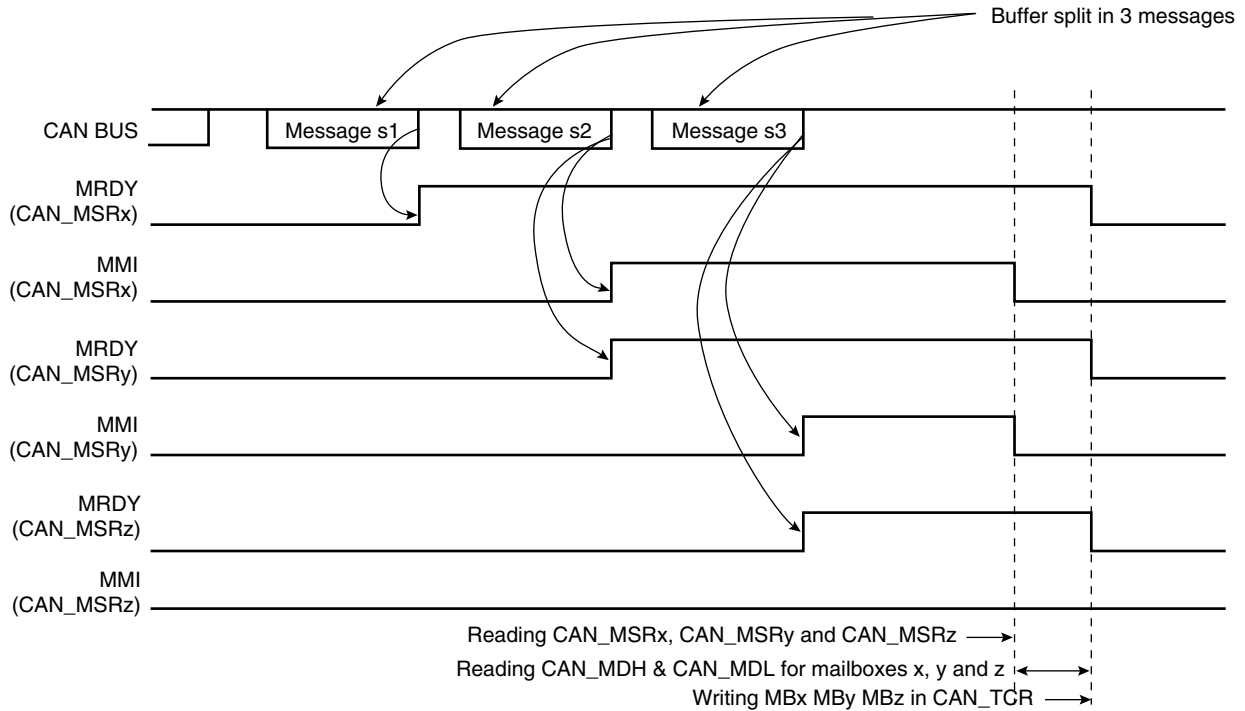


## Chaining Mailboxes

Several mailboxes may be used to receive a buffer split into several messages with the same ID. In this case, the mailbox with the lowest number is serviced first. In the receive and receive with overwrite modes, the field PRIOR in the CAN\_MMRx register has no effect. If Mailbox 0 and Mailbox 5 accept messages with the same ID, the first message is received by Mailbox 0 and the second message is received by Mailbox 5. Mailbox 0 must be configured in Receive Mode (i.e., the first message received is considered) and Mailbox 5 must be configured in Receive with Overwrite Mode. Mailbox 0 cannot be configured in Receive with Overwrite Mode; otherwise, all messages are accepted by this mailbox and Mailbox 5 is never serviced.

If several mailboxes are chained to receive a buffer split into several messages, all mailboxes except the last one (with the highest number) must be configured in Receive Mode. The first message received is handled by the first mailbox, the second one is refused by the first mailbox and accepted by the second mailbox, the last message is accepted by the last mailbox and refused by previous ones (see [Figure 36-13](#)).

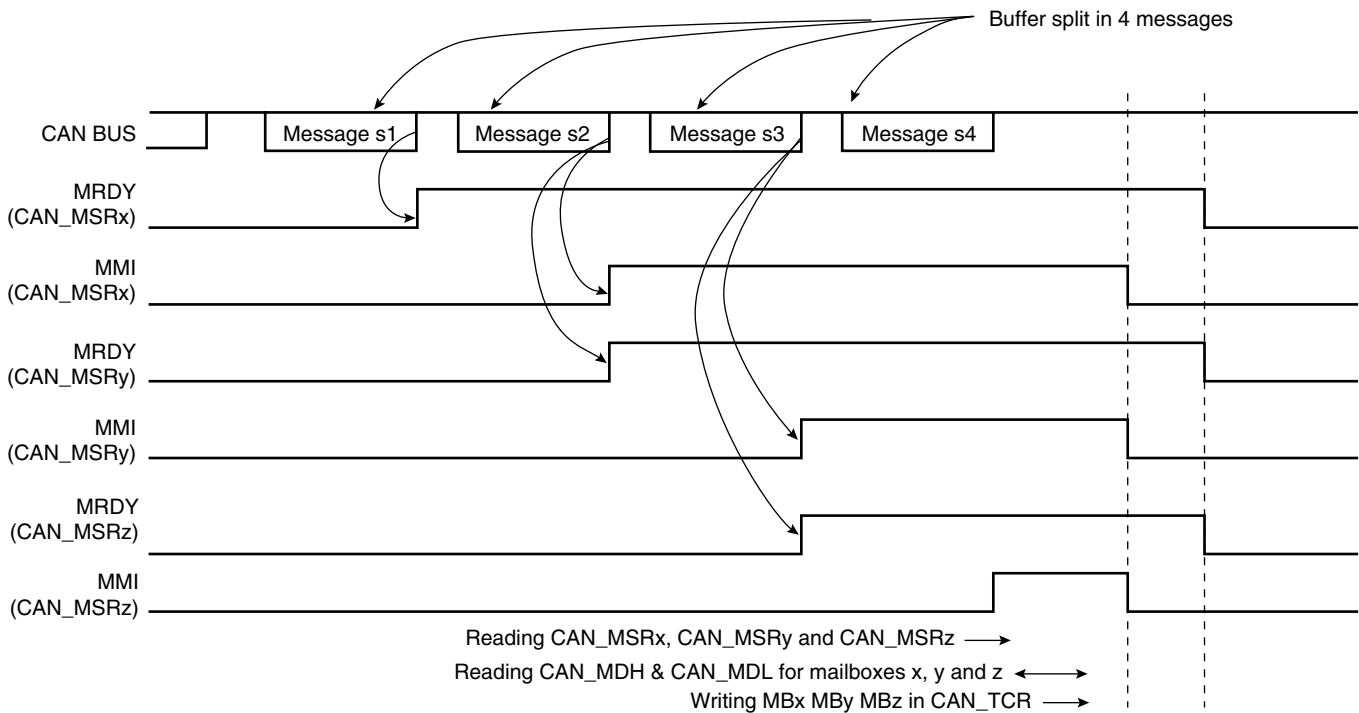
**Figure 36-13.** Chaining Three Mailboxes to Receive a Buffer Split into Three Messages



If the number of mailboxes is not sufficient (the MMI flag of the last mailbox raises), the user must read each data received on the last mailbox in order to retrieve all the messages of the buffer split (see [Figure 36-14](#)).



**Figure 36-14.** Chaining Three Mailboxes to Receive a Buffer Split into Four Messages



### 36.7.3.2 Transmission Handling

A mailbox is in Transmit Mode once the MOT field in the CAN\_MMRx register has been configured. Message ID and Message Acceptance mask must be set before Receive Mode is enabled.

After Transmit Mode is enabled, the MRDY flag in the CAN\_MSR register is automatically set until the first command is sent. When the MRDY flag is set, the software application can prepare a message to be sent by writing to the CAN\_MDx registers. The message is sent once the software asks for a transfer command setting the MTCR bit and the message data length in the CAN\_MCRx register.

The MRDY flag remains at zero as long as the message has not been sent or aborted. It is important to note that no access to the mailbox data register is allowed while the MRDY flag is cleared. An interrupt is pending for the mailbox while the MRDY flag is set. This interrupt can be masked depending on the mailbox flag in the CAN\_IMR global register.

It is also possible to send a remote frame setting the MRTR bit instead of setting the MDLC field. The answer to the remote frame is handled by another reception mailbox. In this case, the device acts as a consumer but with the help of two mailboxes. It is possible to handle the remote frame emission and the answer reception using only one mailbox configured in Consumer Mode. Refer to the section ["Remote Frame Handling" on page 514](#).

Several messages can try to win the bus arbitration in the same time. The message with the highest priority is sent first. Several transfer request commands can be generated at the same time by setting MBx bits in the CAN\_TCR register. The priority is set in the PRIOR field of the CAN\_MMRx register. Priority 0 is the highest priority, priority 15 is the lowest priority. Thus it is possible to use a part of the message ID to set the PRIOR field. If two mailboxes have the same priority, the message of the mailbox with the lowest number is sent first. Thus if mailbox

0 and mailbox 5 have the same priority and have a message to send at the same time, then the message of the mailbox 0 is sent first.

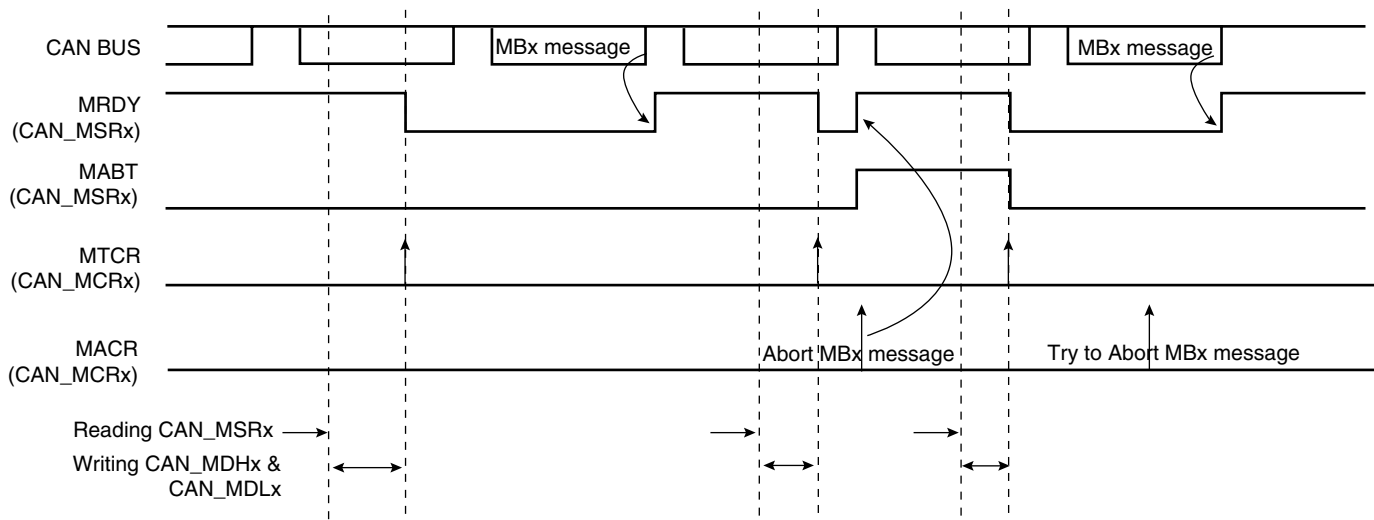
Setting the MACR bit in the CAN\_MCRx register aborts the transmission. Transmission for several mailboxes can be aborted by writing MBx fields in the CAN\_MACR register. If the message is being sent when the abort command is set, then the application is notified by the MRDY bit set and not the MABT in the CAN\_MSRx register. Otherwise, if the message has not been sent, then the MRDY and the MABT are set in the CAN\_MSR register.

When the bus arbitration is lost by a mailbox message, the CAN controller tries to win the next bus arbitration with the same message if this one still has the highest priority. Messages to be sent are re-tried automatically until they win the bus arbitration. This feature can be disabled by setting the bit DRPT in the CAN\_MR register. In this case if the message was not sent the first time it was transmitted to the CAN transceiver, it is automatically aborted. The MABT flag is set in the CAN\_MSRx register until the next transfer command.

Figure 36-15 shows three MBx message attempts being made (MRDY of MBx set to 0).

The first MBx message is sent, the second is aborted and the last one is trying to be aborted but too late because it has already been transmitted to the CAN transceiver.

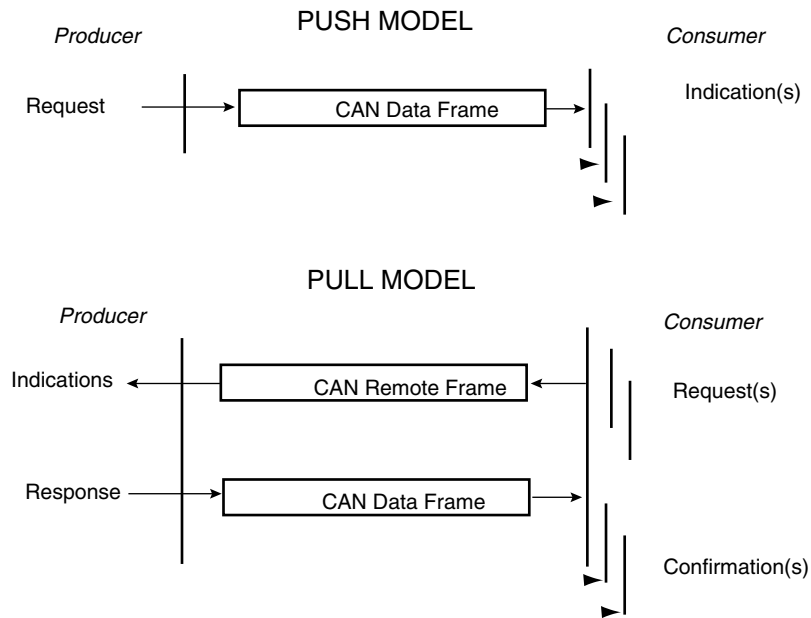
**Figure 36-15.** Transmitting Messages



### 36.7.3.3 Remote Frame Handling

Producer/consumer model is an efficient means of handling broadcasted messages. The push model allows a producer to broadcast messages; the pull model allows a customer to ask for messages.

**Figure 36-16.** Producer / Consumer Model



In Pull Mode, a consumer transmits a remote frame to the producer. When the producer receives a remote frame, it sends the answer accepted by one or many consumers. Using transmit and receive mailboxes, a consumer must dedicate two mailboxes, one in Transmit Mode to send remote frames, and at least one in Receive Mode to capture the producer's answer. The same structure is applicable to a producer: one reception mailbox is required to get the remote frame and one transmit mailbox to answer.

Mailboxes can be configured in Producer or Consumer Mode. A lonely mailbox can handle the remote frame and the answer. With 16 mailboxes, the CAN controller can handle 16 independent producers/consumers.

### Producer Configuration

A mailbox is in Producer Mode once the MOT field in the CAN\_MMRx register has been configured. Message ID and Message Acceptance masks must be set before Receive Mode is enabled.

After Producer Mode is enabled, the MRDY flag in the CAN\_MSR register is automatically set until the first transfer command. The software application prepares data to be sent by writing to the CAN\_MDHx and the CAN\_MDLx registers, then by setting the MTCR bit in the CAN\_MCRx register. Data is sent after the reception of a remote frame as soon as it wins the bus arbitration.

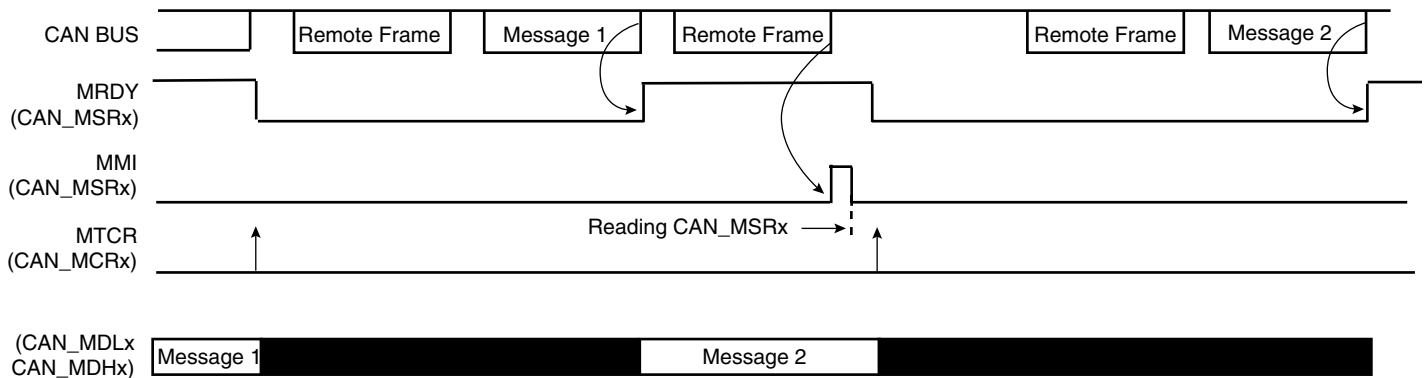
The MRDY flag remains at zero as long as the message has not been sent or aborted. No access to the mailbox data register can be done while MRDY flag is cleared. An interrupt is pending for the mailbox while the MRDY flag is set. This interrupt can be masked according to the mailbox flag in the CAN\_IMR global register.

If a remote frame is received while no data are ready to be sent (signal MRDY set in the CAN\_MSRx register), then the MMI signal is set in the CAN\_MSRx register. This bit is cleared by reading the CAN\_MSRx register.

The MRTR field in the CAN\_MSRx register has no meaning. This field is used only when using Receive and Receive with Overwrite modes.

After a remote frame has been received, the mailbox functions like a transmit mailbox. The message with the highest priority is sent first. The transmitted message may be aborted by setting the MACR bit in the CAN\_MCR register. Please refer to the section ["Transmission Handling"](#) on page 513.

**Figure 36-17. Producer Handling**



### Consumer Configuration

A mailbox is in Consumer Mode once the MOT field in the CAN\_MMRx register has been configured. Message ID and Message Acceptance masks must be set before Receive Mode is enabled.

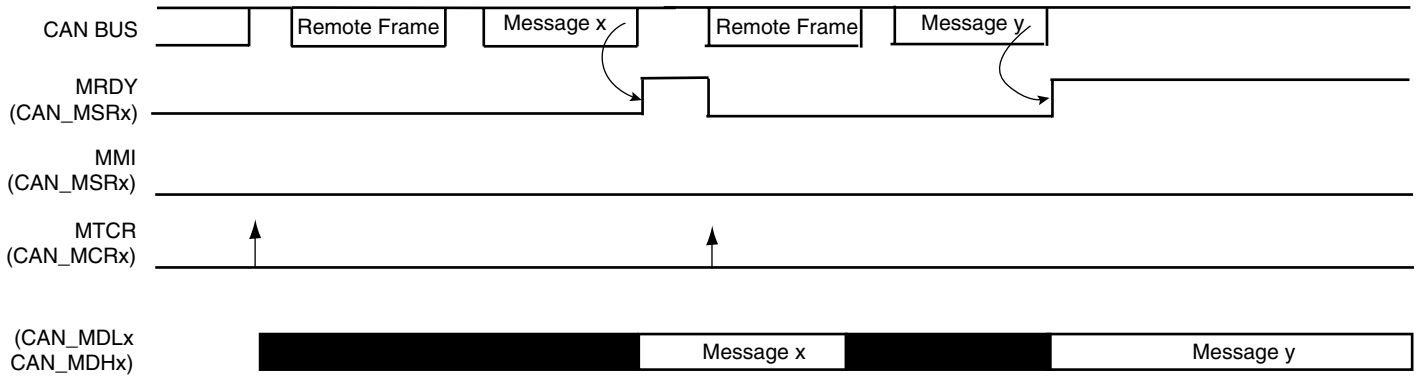
After Consumer Mode is enabled, the MRDY flag in the CAN\_MSR register is automatically cleared until the first transfer request command. The software application sends a remote frame by setting the MTCR bit in the CAN\_MCRx register or the MBx bit in the global CAN\_TCR register. The application is notified of the answer by the MRDY flag set in the CAN\_MSRx register. The application can read the data contents in the CAN\_MDLx and CAN\_MDHx registers. An interrupt is pending for the mailbox while the MRDY flag is set. This interrupt can be masked according to the mailbox flag in the CAN\_IMR global register.

The MRTR bit in the CAN\_MCRx register has no effect. This field is used only when using Transmit Mode.

After a remote frame has been sent, the consumer mailbox functions as a reception mailbox. The first message received is stored in the mailbox data registers. If other messages intended for this mailbox have been sent while the MRDY flag is set in the CAN\_MSRx register, they will be lost. The application is notified by reading the MMI field in the CAN\_MSRx register. The read operation automatically clears the MMI flag.

If several messages are answered by the Producer, the CAN controller may have one mailbox in consumer configuration, zero or several mailboxes in Receive Mode and one mailbox in Receive with Overwrite Mode. In this case, the consumer mailbox must have a lower number than the Receive with Overwrite mailbox. The transfer command can be triggered for all mailboxes at the same time by setting several MBx fields in the CAN\_TCR register.

Figure 36-18. Consumer Handling



## 36.7.4 CAN Controller Timing Modes

Using the free running 16-bit internal timer, the CAN controller can be set in one of the two following timing modes:

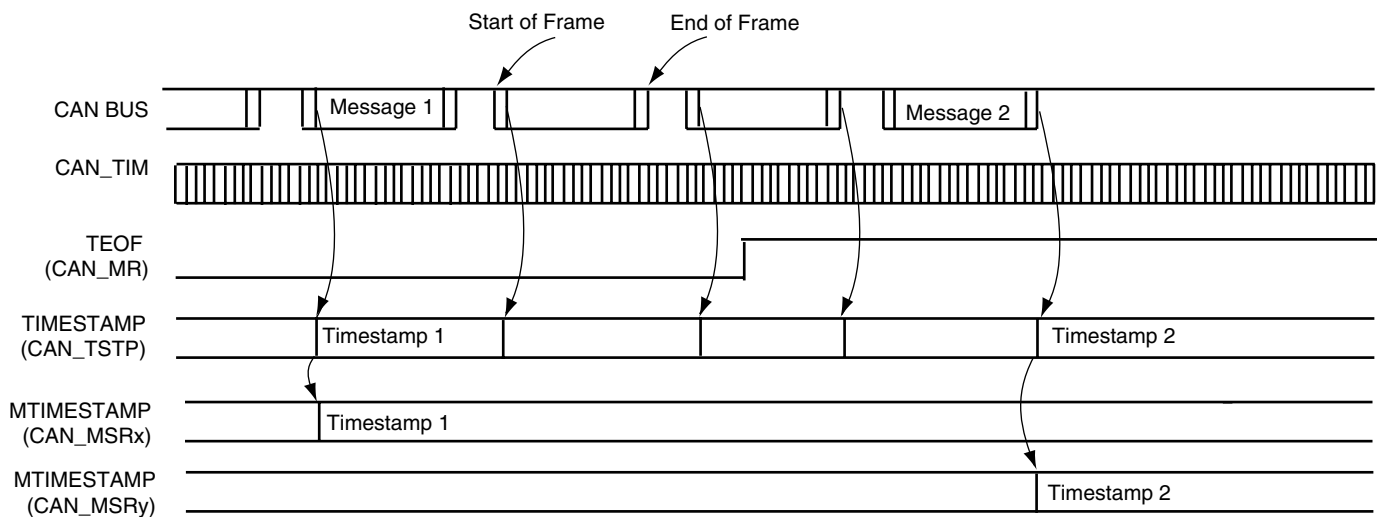
- **Timestamping Mode:** The value of the internal timer is captured at each Start Of Frame or each End Of Frame.
- **Time Triggered Mode:** The mailbox transfer operation is triggered when the internal timer reaches the mailbox trigger.

Timestamping Mode is enabled by clearing the TTM bit in the CAN\_MR register. Time Triggered Mode is enabled by setting the TTM bit in the CAN\_MR register.

### 36.7.4.1 Timestamping Mode

Each mailbox has its own timestamp value. Each time a message is sent or received by a mailbox, the 16-bit value MTIMESTAMP of the CAN\_TIMESTP register is transferred to the LSB bits of the CAN\_MSRx register. The value read in the CAN\_MSRx register corresponds to the internal timer value at the Start Of Frame or the End Of Frame of the message handled by the mailbox.

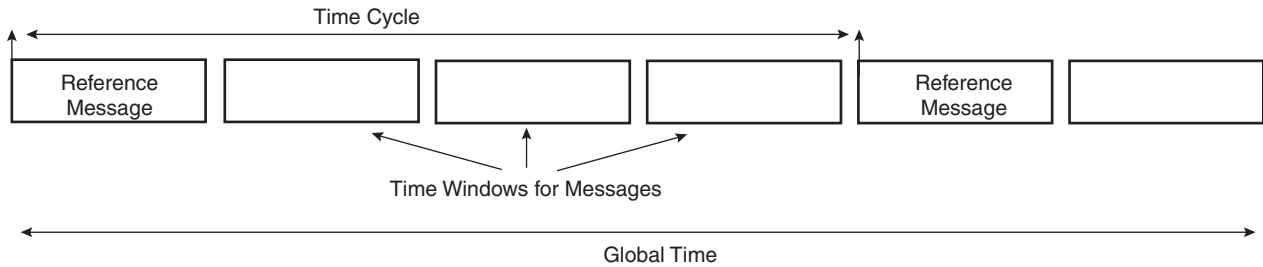
**Figure 36-19.** Mailbox Timestamp



### 36.7.4.2 Time Triggered Mode

In Time Triggered Mode, basic cycles can be split into several time windows. A basic cycle starts with a reference message. Each time a window is defined from the reference message, a transmit operation should occur within a pre-defined time window. A mailbox must not win the arbitration in a previous time window, and it must not be retried if the arbitration is lost in the time window.

**Figure 36-20.** Time Triggered Principle



Time Trigger Mode is enabled by setting the TTM field in the CAN\_MR register. In Time Triggered Mode, as in Timestamp Mode, the CAN\_TIMESTP field captures the values of the internal counter, but the MTIMESTAMP fields in the CAN\_MSRx registers are not active and are read at 0.

### *Synchronization by a Reference Message*

In Time Triggered Mode, the internal timer counter is automatically reset when a new message is received in the last mailbox. This reset occurs after the reception of the End Of Frame on the rising edge of the MRDY signal in the CAN\_MSRx register. This allows synchronization of the internal timer counter with the reception of a reference message and the start a new time window.

### *Transmitting within a Time Window*

A time mark is defined for each mailbox. It is defined in the 16-bit MTIMEMARK field of the CAN\_MMRx register. At each internal timer clock cycle, the value of the CAN\_TIM is compared with each mailbox time mark. When the internal timer counter reaches the MTIMEMARK value, an internal timer event for the mailbox is generated for the mailbox.

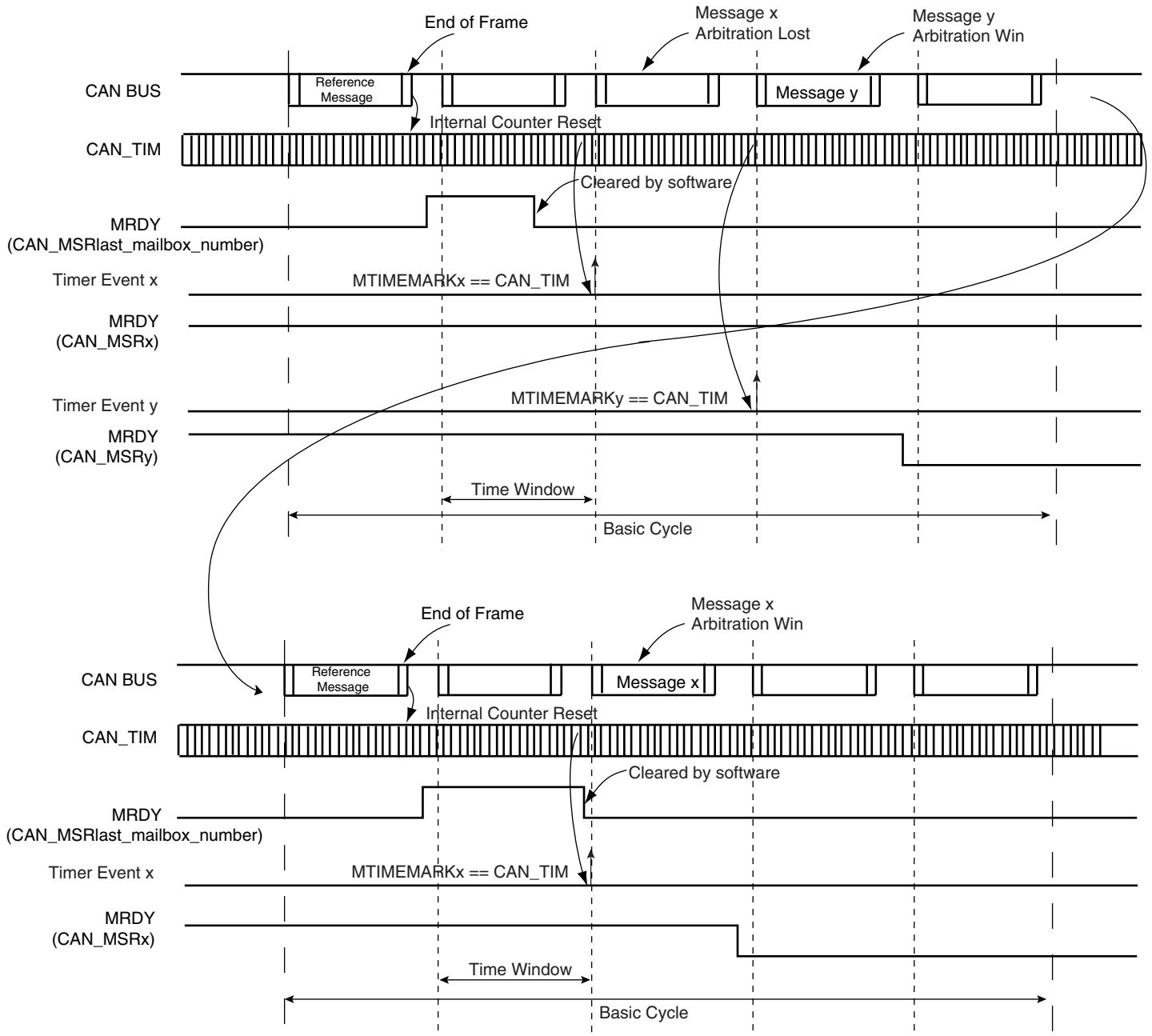
In Time Triggered Mode, transmit operations are delayed until the internal timer event for the mailbox. The application prepares a message to be sent by setting the MTCR in the CAN\_MCRx register. The message is not sent until the CAN\_TIM value is less than the MTIMEMARK value defined in the CAN\_MMRx register.

If the transmit operation is failed, i.e., the message loses the bus arbitration and the next transmit attempt is delayed until the next internal time trigger event. This prevents overlapping the next time window, but the message is still pending and is retried in the next time window when CAN\_TIM value equals the MTIMEMARK value. It is also possible to prevent a retry by setting the DRPT field in the CAN\_MR register.

### *Freezing the Internal Timer Counter*

The internal counter can be frozen by setting TIMFRZ in the CAN\_MR register. This prevents an unexpected roll-over when the counter reaches FFFFh. When this occurs, it automatically freezes until a new reset is issued, either due to a message received in the last mailbox or any other reset counter operations. The TOVF bit in the CAN\_SR register is set when the counter is frozen. The TOVF bit in the CAN\_SR register is cleared by reading the CAN\_SR register. Depending on the corresponding interrupt mask in the CAN\_IMR register, an interrupt is generated when TOVF is set.

**Figure 36-21. Time Triggered Operations**





## 36.8 Controller Area Network (CAN) User Interface

**Table 36-4.** CAN Memory Map

Offset	Register	Name	Access	Reset State
0x0000	Mode Register	CAN_MR	Read-Write	0x0
0x0004	Interrupt Enable Register	CAN_IER	Write-only	-
0x0008	Interrupt Disable Register	CAN_IDR	Write-only	-
0x000C	Interrupt Mask Register	CAN_IMR	Read-only	0x0
0x0010	Status Register	CAN_SR	Read-only	0x0
0x0014	Baudrate Register	CAN_BR	Read/Write	0x0
0x0018	Timer Register	CAN_TIM	Read-only	0x0
0x001C	Timestamp Register	CAN_TIMESTP	Read-only	0x0
0x0020	Error Counter Register	CAN_ECR	Read-only	0x0
0x0024	Transfer Command Register	CAN_TCR	Write-only	-
0x0028	Abort Command Register	CAN_ACR	Write-only	-
0x0100 - 0x01FC	Reserved	-	-	-
0x0200	Mailbox 0 Mode Register	CAN_MMR0	Read/Write	0x0
0x0204	Mailbox 0 Acceptance Mask Register	CAN_MAM0	Read/Write	0x0
0x0208	Mailbox 0 ID Register	CAN_MID0	Read/Write	0x0
0x020C	Mailbox 0 Family ID Register	CAN_MFID0	Read-only	0x0
0x0210	Mailbox 0 Status Register	CAN_MSR0	Read-only	0x0
0x0214	Mailbox 0 Data Low Register	CAN_MDL0	Read/Write	0x0
0x0218	Mailbox 0 Data High Register	CAN_MDH0	Read/Write	0x0
0x021C	Mailbox 0 Control Register	CAN_MCR0	Write-only	-
0x0220	Mailbox 1 Mode Register	CAN_MMR1	Read/Write	0x0
0x0224	Mailbox 1 Acceptance Mask Register	CAN_MAM1	Read/Write	0x0
0x0228	Mailbox 1 ID register	CAN_MID1	Read/Write	0x0
0x022C	Mailbox 1 Family ID Register	CAN_MFID1	Read-only	0x0
0x0230	Mailbox 1 Status Register	CAN_MSR1	Read-only	0x0
0x0234	Mailbox 1 Data Low Register	CAN_MDL1	Read/Write	0x0
0x0238	Mailbox 1 Data High Register	CAN_MDH1	Read/Write	0x0
0x023C	Mailbox 1 Control Register	CAN_MCR1	Write-only	-
...	...	...	...	-

## 36.8.1 CAN Mode Register

**Name:** CAN\_MR  
**Access Type:** Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–			
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
DRPT	TIMFRZ	TTM	TEOF	OVL	ABM	LPM	CANEN

- **CANEN: CAN Controller Enable**

0 = The CAN Controller is disabled.  
 1 = The CAN Controller is enabled.

- **LPM: Disable/Enable Low Power Mode**

0 = Disable w Power Mode.  
 1 = Enable Low Power Mo

CAN controller enters Low Power Mode once all pending messages have been transmitted.

- **ABM: Disable/Enable Autobaud/Listen mode**

0 = Disable Autobaud/listen mode.  
 1 = Enable Autobaud/listen mode.

- **OVL: Disable/Enable Overload Frame**

0 = No overload frame is generated.  
 1 = An overload frame is generated after each successful reception for mailboxes configured in Receive with/without overwrite Mode, Producer and Consumer.

- **TEOF: Timestamp messages at each end of Frame**

0 = The value of CAN\_TIM is captured in the CAN\_TIMESTP register at each Start Of Frame.  
 1 = The value of CAN\_TIM is captured in the CAN\_TIMESTP register at each End Of Frame.

- **TTM: Disable/Enable Time Triggered Mode**

0 = Time Triggered Mode is disabled.  
 1 = Time Triggered Mode is enabled.

- **TIMFRZ: Enable Timer Freeze**

0 = The internal timer continues to be incremented after it reached 0xFFFF.  
 1 = The internal timer stops incrementing after reaching 0xFFFF. It is restarted after a timer reset. See ["Freezing the Internal Timer Counter" on page 519.](#)

- **DRPT: Disable Repeat**

0 = When a transmit mailbox loses the bus arbitration, the transfer request remains pending.  
 1 = When a transmit mailbox lose the bus arbitration, the transfer request is automatically aborted. It automatically raises the MABT and MRDT flags in the corresponding CAN\_MSRx.

## 36.8.2 CAN Interrupt Enable Register

**Name:** CAN\_IER

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	BERR	FERR	AERR	SERR	CERR
23	22	21	20	19	18	17	16
TSTP	TOVF	WAKEUP	SLEEP	BOFF	ERRP	WARN	ERRA
15	14	13	12	11	10	9	8
MB15	MB14	MB13	MB12	MB11	MB10	MB9	MB8
7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

- **MBx: Mailbox x Interrupt Enable**

0 = No effect.

1 = Enable Mailbox x interrupt.

- **ERRA: Error Active mode Interrupt Enable**

0 = No effect.

1 = Enable ERRA interrupt.

- **WARN: Warning Limit Interrupt Enable**

0 = No effect.

1 = Enable WARN interrupt.

- **ERRP: Error Passive mode Interrupt Enable**

0 = No effect.

1 = Enable ERRP interrupt.

- **BOFF: Bus-off mode Interrupt Enable**

0 = No effect.

1 = Enable BOFF interrupt.

- **SLEEP: Sleep Interrupt Enable**

0 = No effect.

1 = Enable SLEEP interrupt.

- **WAKEUP: Wakeup Interrupt Enable**

0 = No effect.

1 = Enable SLEEP interrupt.

- **TOVF: Timer Overflow Interrupt Enable**

0 = No effect.

1 = Enable TOVF interrupt.

- **TSTP: TimeStamp Interrupt Enable**

0 = No effect.

1 = Enable TSTP interrupt.

- **CERR: CRC Error Interrupt Enable**

0 = No effect.

1 = Enable CRC Error interrupt.

- **SERR: Stuffing Error Interrupt Enable**

0 = No effect.

1 = Enable Stuffing Error interrupt.

- **AERR: Acknowledgment Error Interrupt Enable**

0 = No effect.

1 = Enable Acknowledgment Error interrupt.

- **FERR: Form Error Interrupt Enable**

0 = No effect.

1 = Enable Form Error interrupt.

- **BERR: Bit Error Interrupt Enable**

0 = No effect.

1 = Enable Bit Error interrupt.

## 36.8.3 CAN Interrupt Disable Register

**Name:** CAN\_IDR

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	BERR	FERR	AERR	SERR	CERR
23	22	21	20	19	18	17	16
TSTP	TOVF	WAKEUP	SLEEP	BOFF	ERRP	WARN	ERRA
15	14	13	12	11	10	9	8
MB15	MB14	MB13	MB12	MB11	MB10	MB9	MB8
7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

- **MBx: Mailbox x Interrupt Disable**

0 = No effect.

1 = Disable Mailbox x interrupt.

- **ERRA: Error Active Mode Interrupt Disable**

0 = No effect.

1 = Disable ERRA interrupt.

- **WARN: Warning Limit Interrupt Disable**

0 = No effect.

1 = Disable WARN interrupt.

- **ERRP: Error Passive mode Interrupt Disable**

0 = No effect.

1 = Disable ERRP interrupt.

- **BOFF: Bus-off mode Interrupt Disable**

0 = No effect.

1 = Disable BOFF interrupt.

- **SLEEP: Sleep Interrupt Disable**

0 = No effect.

1 = Disable SLEEP interrupt.

- **WAKEUP: Wakeup Interrupt Disable**

0 = No effect.

1 = Disable WAKEUP interrupt.

- **TOVF: Timer Overflow Interrupt**

0 = No effect.

1 = Disable TOVF interrupt.

- **TSTP: TimeStamp Interrupt Disable**

0 = No effect.

1 = Disable TSTP interrupt.

- **CERR: CRC Error Interrupt Disable**

0 = No effect.

1 = Disable CRC Error interrupt.

- **SERR: Stuffing Error Interrupt Disable**

0 = No effect.

1 = Disable Stuffing Error interrupt.

- **AERR: Acknowledgment Error Interrupt Disable**

0 = No effect.

1 = Disable Acknowledgment Error interrupt.

- **FERR: Form Error Interrupt Disable**

0 = No effect.

1 = Disable Form Error interrupt.

- **BERR: Bit Error Interrupt Disable**

0 = No effect.

1 = Disable Bit Error interrupt.

## 36.8.4 CAN Interrupt Mask Register

**Name:** CAN\_IMR

**Access Type:** Read-only

31	30	29	28	27	26	25	24
–	–	–	BERR	FERR	AERR	SERR	CERR
23	22	21	20	19	18	17	16
TSTP	TOVF	WAKEUP	SLEEP	BOFF	ERRP	WARN	ERRA
15	14	13	12	11	10	9	8
MB15	MB14	MB13	MB12	MB11	MB10	MB9	MB8
7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

- **MBx: Mailbox x Interrupt Mask**

0 = Mailbox x interrupt is disabled.

1 = Mailbox x interrupt is enabled.

- **ERRA: Error Active mode Interrupt Mask**

0 = ERRA interrupt is disabled.

1 = ERRA interrupt is enabled.

- **WARN: Warning Limit Interrupt Mask**

0 = Warning Limit interrupt is disabled.

1 = Warning Limit interrupt is enabled.

- **ERRP: Error Passive Mode Interrupt Mask**

0 = ERRP interrupt is disabled.

1 = ERRP interrupt is enabled.

- **BOFF: Bus-off Mode Interrupt Mask**

0 = BOFF interrupt is disabled.

1 = BOFF interrupt is enabled.

- **SLEEP: Sleep Interrupt Mask**

0 = SLEEP interrupt is disabled.

1 = SLEEP interrupt is enabled.

- **WAKEUP: Wakeup Interrupt Mask**

0 = WAKEUP interrupt is disabled.

1 = WAKEUP interrupt is enabled.

- **TOVF: Timer Overflow Interrupt Mask**

0 = TOVF interrupt is disabled.

1 = TOVF interrupt is enabled.

- **TSTP: Timestamp Interrupt Mask**

0 = TSTP interrupt is disabled.

1 = TSTP interrupt is enabled.

- **CERR: CRC Error Interrupt Mask**

0 = CRC Error interrupt is disabled.

1 = CRC Error interrupt is enabled.

- **SERR: Stuffing Error Interrupt Mask**

0 = Bit Stuffing Error interrupt is disabled.

1 = Bit Stuffing Error interrupt is enabled.

- **AERR: Acknowledgment Error Interrupt Mask**

0 = Acknowledgment Error interrupt is disabled.

1 = Acknowledgment Error interrupt is enabled.

- **FERR: Form Error Interrupt Mask**

0 = Form Error interrupt is disabled.

1 = Form Error interrupt is enabled.

- **BERR: Bit Error Interrupt Mask**

0 = Bit Error interrupt is disabled.

1 = Bit Error interrupt is enabled.



## 36.8.5 CAN Status Register

**Name:** CAN\_SR

**Access Type:** Read-only

31	30	29	28	27	26	25	24
OVLSY	TBSY	RBSY	BERR	FERR	AERR	SERR	CERR
23	22	21	20	19	18	17	16
TSTP	TOVF	WAKEUP	SLEEP	BOFF	ERRP	WARN	ERRA
15	14	13	12	11	10	9	8
MB15	MB14	MB13	MB12	MB11	MB10	MB9	MB8
7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

- **MBx: Mailbox x Event**

0 = No event occurred on Mailbox x.

1 = An event occurred on Mailbox x.

An event corresponds to MRDY, MABT fields in the CAN\_MS Rx register.

- **ERRA: Error Active mode**

0 = CAN controller is not in error active mode

1 = CAN controller is in error active mode

This flag is set depending on TEC and REC counter values. It is set when node is neither in error passive mode nor in bus off mode.

This flag is automatically reset when above condition is not satisfied.

- **WARN: Warning Limit**

0 = CAN controller Warning Limit is not reached.

1 = CAN controller Warning Limit is reached.

This flag is set depending on TEC and REC counters values. It is set when at least one of the counters values exceeds 96.

This flag is automatically reset when above condition is not satisfied.

- **ERRP: Error Passive mode**

0 = CAN controller is not in error passive mode

1 = CAN controller is in error passive mode

This flag is set depending on TEC and REC counters values.

A node is error passive when TEC counter is greater or equal to 128 (decimal) or when the REC counter is greater or equal to 128 (decimal) and less than 256.

This flag is automatically reset when above condition is not satisfied.

- **BOFF: Bus Off mode**

0 = CAN controller is not in bus-off mode

1 = CAN controller is in bus-off mode

This flag is set depending on TEC counter value. A node is bus off when TEC counter is greater or equal to 256 (decimal).

This flag is automatically reset when above condition is not satisfied.

- **SLEEP: CAN controller in Low power Mode**

0 = CAN controller is not in low power mode.

1 = CAN controller is in low power mode.

This flag is automatically reset when Low power mode is disabled

- **WAKEUP: CAN controller is not in Low power Mode**

0 = CAN controller is in low power mode.

1 = CAN controller is not in low power mode.

When a WAKEUP event occurs, the CAN controller is synchronized with the bus activity. Messages can be transmitted or received. The CAN controller clock must be available when a WAKEUP event occurs. This flag is automatically reset when the CAN Controller enters Low Power mode.

- **TOVF: Timer Overflow**

0 = The timer has not rolled-over FFFFh to 0000h.

1 = The timer rolls-over FFFFh to 0000h.

This flag is automatically cleared by reading CAN\_SR register.

- **TSTP Timestamp**

0 = No bus activity has been detected.

1 = A start of frame or an end of frame has been detected (according to the TEOF field in the CAN\_MR register).

This flag is automatically cleared by reading the CAN\_SR register.

- **CERR: Mailbox CRC Error**

0 = No CRC error occurred during a previous transfer.

1 = A CRC error occurred during a previous transfer.

A CRC error has been detected during last reception.

This flag is automatically cleared by reading CAN\_SR register.

- **SERR: Mailbox Stuffing Error**

0 = No stuffing error occurred during a previous transfer.

1 = A stuffing error occurred during a previous transfer.

A form error results from the detection of more than five consecutive bit with the same polarity.

This flag is automatically cleared by reading CAN\_SR register.

- **AERR: Acknowledgment Error**

0 = No acknowledgment error occurred during a previous transfer.

1 = An acknowledgment error occurred during a previous transfer.

An acknowledgment error is detected when no detection of the dominant bit in the acknowledge slot occurs.

This flag is automatically cleared by reading CAN\_SR register.

- **FERR: Form Error**

0 = No form error occurred during a previous transfer

1 = A form error occurred during a previous transfer

A form error results from violations on one or more of the fixed form of the following bit fields:

- CRC delimiter
- ACK delimiter
- End of frame
- Error delimiter
- Overload delimiter

This flag is automatically cleared by reading CAN\_SR register.

- **BERR: Bit Error**

0 = No bit error occurred during a previous transfer.

1 = A bit error occurred during a previous transfer.

A bit error is set when the bit value monitored on the line is different from the bit value sent.

This flag is automatically cleared by reading CAN\_SR register.

- **RBSY: Receiver busy**

0 = CAN receiver is not receiving a frame.

1 = CAN receiver is receiving a frame.

Receiver busy. This status bit is set by hardware while CAN receiver is acquiring or monitoring a frame (remote, data, overload or error frame). It is automatically reset when CAN is not receiving.

- **TBSY: Transmitter busy**

0 = CAN transmitter is not transmitting a frame.

1 = CAN transmitter is transmitting a frame.

Transmitter busy. This status bit is set by hardware while CAN transmitter is generating a frame (remote, data, overload or error frame). It is automatically reset when CAN is not transmitting.

- **OVLSY: Overload busy**

0 = CAN transmitter is not transmitting an overload frame.

1 = CAN transmitter is transmitting an overload frame.

It is automatically reset when the bus is not transmitting an overload frame.

## 36.8.6 CAN Baudrate Register

**Name:** CAN\_BR  
**Access Type:** Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	SMP
23	22	21	20	19	18	17	16
–	BRP						
15	14	13	12	11	10	9	8
–	–	SJW		–	PROPAG		
7	6	5	4	3	2	1	0
–	PHASE1			–	PHASE2		

Any modification on one of the fields of the CANBR register must be done while CAN module is disabled.

To compute the different Bit Timings, please refer to the [Section 36.6.4.1 "CAN Bit Timing Configuration" on page 500](#).

- **PHASE2: Phase 2 segment**

This phase is used to compensate the edge phase error.

$$t_{PHS2} = t_{CSC} \times (PHASE2 + 1)$$

**Warning:** PHASE2 value must be different from 0.

- **PHASE1: Phase 1 segment**

This phase is used to compensate for edge phase error.

$$t_{PHS1} = t_{CSC} \times (PHASE1 + 1)$$

- **PROPAG: Programming time segment**

This part of the bit time is used to compensate for the physical delay times within the network.

$$t_{PRS} = t_{CSC} \times (PROPAG + 1)$$

- **SJW: Re-synchronization jump width**

To compensate for phase shifts between clock oscillators of different controllers on bus. The controller must re-synchronize on any relevant signal edge of the current transmission. The synchronization jump width defines the maximum of clock cycles a bit period may be shortened or lengthened by re-synchronization.

$$t_{SJW} = t_{CSC} \times (SJW + 1)$$

- **BRP: Baudrate Prescaler.**

This field allows user to program the period of the CAN system clock to determine the individual bit timing.

$$t_{CSC} = (BRP + 1) / MCK$$

The BRP field must be within the range [1, 0x7F], i.e., BRP = 0 is not authorized.

- **SMP: Sampling Mode**

0 = The incoming bit stream is sampled once at sample point.

1 = The incoming bit stream is sampled three times with a period of a MCK clock period, centered on sample point.

SMP Sampling Mode is automatically disabled if BRP = 0.

## 36.8.7 CAN Timer Register

**Name:** CAN\_TIM

**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
TIMER15	TIMER14	TIMER13	TIMER12	TIMER11	TIMER10	TIMER9	TIMER8
7	6	5	4	3	2	1	0
TIMER7	TIMER6	TIMER5	TIMER4	TIMER3	TIMER2	TIMER1	TIMER0

- **TIMERx: Timer**

This field represents the internal CAN controller 16-bit timer value.

## 36.8.8 CAN Timestamp Register

**Name:** CAN\_TIMESTP

**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
MTIMESTAMP 15	MTIMESTAMP 14	MTIMESTAMP 13	MTIMESTAMP 12	MTIMESTAMP 11	MTIMESTAMP 10	MTIMESTAMP 9	MTIMESTAMP 8
7	6	5	4	3	2	1	0
MTIMESTAMP 7	MTIMESTAMP 6	MTIMESTAMP 5	MTIMESTAMP 4	MTIMESTAMP 3	MTIMESTAMP 2	MTIMESTAMP 1	MTIMESTAMP 0

- **MTIMESTAMPx: Timestamp**

This field represents the internal CAN controller 16-bit timer value.

If the TEOF bit is cleared in the CAN\_MR register, the internal Timer Counter value is captured in the MTIMESTAMP field at each start of frame. Else the value is captured at each end of frame. When the value is captured, the TSTP flag is set in the CAN\_SR register. If the TSTP mask in the CAN\_IMR register is set, an interrupt is generated while TSTP flag is set in the CAN\_SR register. This flag is cleared by reading the CAN\_SR register.

Note: The CAN\_TIMESTP register is reset when the CAN is disabled then enabled thanks to the CANEN bit in the CAN\_MR.

## 36.8.9 CAN Error Counter Register

**Name:** CAN\_ECR

**Access Type:** Read-only

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

### • REC: Receive Error Counter

When a receiver detects an error, REC will be increased by one, except when the detected error is a BIT ERROR while sending an ACTIVE ERROR FLAG or an OVERLOAD FLAG.

When a receiver detects a dominant bit as the first bit after sending an ERROR FLAG, REC is increased by 8.

When a receiver detects a BIT ERROR while sending an ACTIVE ERROR FLAG, REC is increased by 8.

Any node tolerates up to 7 consecutive dominant bits after sending an ACTIVE ERROR FLAG, PASSIVE ERROR FLAG or OVERLOAD FLAG. After detecting the 14th consecutive dominant bit (in case of an ACTIVE ERROR FLAG or an OVERLOAD FLAG) or after detecting the 8th consecutive dominant bit following a PASSIVE ERROR FLAG, and after each sequence of additional eight consecutive dominant bits, each receiver increases its REC by 8.

After successful reception of a message, REC is decreased by 1 if it was between 1 and 127. If REC was 0, it stays 0, and if it was greater than 127, then it is set to a value between 119 and 127.

### • TEC: Transmit Error Counter

When a transmitter sends an ERROR FLAG, TEC is increased by 8 except when

- the transmitter is error passive and detects an ACKNOWLEDGMENT ERROR because of not detecting a dominant ACK and does not detect a dominant bit while sending its PASSIVE ERROR FLAG.
- the transmitter sends an ERROR FLAG because a STUFF ERROR occurred during arbitration and should have been recessive and has been sent as recessive but monitored as dominant.

When a transmitter detects a BIT ERROR while sending an ACTIVE ERROR FLAG or an OVERLOAD FLAG, the TEC will be increased by 8.

Any node tolerates up to 7 consecutive dominant bits after sending an ACTIVE ERROR FLAG, PASSIVE ERROR FLAG or OVERLOAD FLAG. After detecting the 14th consecutive dominant bit (in case of an ACTIVE ERROR FLAG or an OVERLOAD FLAG) or after detecting the 8th consecutive dominant bit following a PASSIVE ERROR FLAG, and after each sequence of additional eight consecutive dominant bits every transmitter increases its TEC by 8.

After a successful transmission the TEC is decreased by 1 unless it was already 0.

## 36.8.10 CAN Transfer Command Register

**Name:** CAN\_TCR

**Access Type:** Write-only

31	30	29	28	27	26	25	24
TIMRST	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
MB15	MB14	MB13	MB12	MB11	MB10	MB9	MB8
7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

This register initializes several transfer requests at the same time.

- **MBx: Transfer Request for Mailbox x**

Mailbox Object Type	Description
Receive	It receives the next message.
Receive with overwrite	This triggers a new reception.
Transmit	Sends data prepared in the mailbox as soon as possible.
Consumer	Sends a remote frame.
Producer	Sends data prepared in the mailbox after receiving a remote frame from a consumer.

This flag clears the MRDY and MABT flags in the corresponding CAN\_MSRx register.

When several mailboxes are requested to be transmitted simultaneously, they are transmitted in turn, starting with the mailbox with the highest priority. If several mailboxes have the same priority, then the mailbox with the lowest number is sent first (i.e., MB0 will be transferred before MB1).

- **TIMRST: Timer Reset**

Resets the internal timer counter. If the internal timer counter is frozen, this command automatically re-enables it. This command is useful in Time Triggered mode.



## 36.8.11 CAN Abort Command Register

**Name:** CAN\_ACR

**Access Type:** Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
MB15	MB14	MB13	MB12	MB11	MB10	MB9	MB8
7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

This register initializes several abort requests at the same time.

- **MBx: Abort Request for Mailbox x**

Mailbox Object Type	Description
Receive	No action
Receive with overwrite	No action
Transmit	Cancels transfer request if the message has not been transmitted to the CAN transceiver.
Consumer	Cancels the current transfer before the remote frame has been sent.
Producer	Cancels the current transfer. The next remote frame is not serviced.

It is possible to set MACR field (in the CAN\_MCRx register) for each mailbox.

## 36.8.12 CAN Message Mode Register

**Name:** CAN\_MMRx

**Access Type:** Read/Write

31	30	29	28	27	26	25	24
-	-	-	-	-	MOT		
23	22	21	20	19	18	17	16
-	-	-	-	PRIOR			
15	14	13	12	11	10	9	8
MTIMEMARK15	MTIMEMARK14	MTIMEMARK13	MTIMEMARK12	MTIMEMARK11	MTIMEMARK10	MTIMEMARK9	MTIMEMARK8
7	6	5	4	3	2	1	0
MTIMEMARK7	MTIMEMARK6	MTIMEMARK5	MTIMEMARK4	MTIMEMARK3	MTIMEMARK2	MTIMEMARK1	MTIMEMARK0

- **MTIMEMARK: Mailbox Timemark**

This field is active in Time Triggered Mode. Transmit operations are allowed when the internal timer counter reaches the Mailbox Timemark. See ["Transmitting within a Time Window" on page 519](#).

In Timestamp Mode, MTIMEMARK is set to 0.

- **PRIOR: Mailbox Priority**

This field has no effect in receive and receive with overwrite modes. In these modes, the mailbox with the lowest number is serviced first.

When several mailboxes try to transmit a message at the same time, the mailbox with the highest priority is serviced first. If several mailboxes have the same priority, the mailbox with the lowest number is serviced first (i.e., MBx0 is serviced before MBx 15 if they have the same priority).

- **MOT: Mailbox Object Type**

This field allows the user to define the type of the mailbox. All mailboxes are independently configurable. Five different types are possible for each mailbox:

MOT			Mailbox Object Type
0	0	0	Mailbox is disabled. This prevents receiving or transmitting any messages with this mailbox.
0	0	1	Reception Mailbox. Mailbox is configured for reception. If a message is received while the mailbox data register is full, it is discarded.
0	1	0	Reception mailbox with overwrite. Mailbox is configured for reception. If a message is received while the mailbox is full, it overwrites the previous message.
0	1	1	Transmit mailbox. Mailbox is configured for transmission.
1	0	0	Consumer Mailbox. Mailbox is configured in reception but behaves as a Transmit Mailbox, i.e., it sends a remote frame and waits for an answer.
1	0	1	Producer Mailbox. Mailbox is configured in transmission but also behaves like a reception mailbox, i.e., it waits to receive a Remote Frame before sending its contents.
1	1	X	Reserved

## 36.8.13 CAN Message Acceptance Mask Register

**Name:** CAN\_MAMx

**Access Type:** Read/Write

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

To prevent concurrent access with the internal CAN core, the application must disable the mailbox before writing to CAN\_MAMx registers.

- **MIDvB: Complementary bits for identifier in extended frame mode**

Acceptance mask for corresponding field of the message IDvB register of the mailbox.

- **MIDvA: Identifier for standard frame mode**

Acceptance mask for corresponding field of the message IDvA register of the mailbox.

- **MIDE: Identifier Version**

0= Compares IDvA from the received frame with the CAN\_MIDx register masked with CAN\_MAMx register.

1= Compares IDvA and IDvB from the received frame with the CAN\_MIDx register masked with CAN\_MAMx register.

## 36.8.14 CAN Message ID Register

**Name:** CAN\_MIDx

**Access Type:** Read/Write



To prevent concurrent access with the internal CAN core, the application must disable the mailbox before writing to CAN\_MIDx registers.

- **MIDvB: Complementary bits for identifier in extended frame mode**

If MIDE is cleared, MIDvB value is 0.

- **MIDE: Identifier Version**

This bit allows the user to define the version of messages processed by the mailbox. If set, mailbox is dealing with version 2.0 Part B messages; otherwise, mailbox is dealing with version 2.0 Part A messages.

- **MIDvA: Identifier for standard frame mode**

## 36.8.15 CAN Message Family ID Register

**Name:** CAN\_MFIDx

**Access Type:** Read-only

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

- **MFID: Family ID**

This field contains the concatenation of CAN\_MIDx register bits masked by the CAN\_MAMx register. This field is useful to speed up message ID decoding. The message acceptance procedure is described below.

As an example:

```
CAN_MIDx = 0x305A4321
CAN_MAMx = 0x3FF0F0FF
CAN_MFIDx = 0x000000A3
```

## 36.8.16 CAN Message Status Register

**Name:** CAN\_MSRx

**Access Type:** Read only

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

These register fields are updated each time a message transfer is received or aborted.

MMI is cleared by reading the CAN\_MSRx register.

MRDY, MABT are cleared by writing MTCR or MACR in the CAN\_MCRx register.

**Warning:** MRTR and MDLC state depends partly on the mailbox object type.

- **MTIMESTAMP: Timer value**

This field is updated only when time-triggered operations are disabled (TTM cleared in CAN\_MR register). If the TEOF field in the CAN\_MR register is cleared, TIMESTAMP is the internal timer value at the start of frame of the last message received or sent by the mailbox. If the TEOF field in the CAN\_MR register is set, TIMESTAMP is the internal timer value at the end of frame of the last message received or sent by the mailbox.

In Time Triggered Mode, MTIMESTAMP is set to 0.

- **MDLC: Mailbox Data Length Code**

Mailbox Object Type	Description
Receive	Length of the first mailbox message received
Receive with overwrite	Length of the last mailbox message received
Transmit	No action
Consumer	Length of the mailbox message received
Producer	Length of the mailbox message to be sent after the remote frame reception

- **MRTR: Mailbox Remote Transmission Request**

Mailbox Object Type	Description
Receive	The first frame received has the RTR bit set.
Receive with overwrite	The last frame received has the RTR bit set.
Transmit	Reserved
Consumer	Reserved. After setting the MOT field in the CAN_MMR, MRTR is reset to 1.
Producer	Reserved. After setting the MOT field in the CAN_MMR, MRTR is reset to 0.

- **MABT: Mailbox Message Abort**

An interrupt is triggered when MABT is set.

0 = Previous transfer is not aborted.

1 = Previous transfer has been aborted.

This flag is cleared by writing to CAN\_MCRx register

Mailbox Object Type	Description
Receive	Reserved
Receive with overwrite	Reserved
Transmit	Previous transfer has been aborted
Consumer	The remote frame transfer request has been aborted.
Producer	The response to the remote frame transfer has been aborted.

- **MRDY: Mailbox Ready**

An interrupt is triggered when MRDY is set.

0 = Mailbox data registers can not be read/written by the software application. CAN\_MDx are locked by the CAN\_MDx.

1 = Mailbox data registers can be read/written by the software application.

This flag is cleared by writing to CAN\_MCRx register.

Mailbox Object Type	Description
Receive	At least one message has been received since the last mailbox transfer order. Data from the first frame received can be read in the CAN_MDxx registers. After setting the MOT field in the CAN_MMR, MRDY is reset to 0.
Receive with overwrite	At least one frame has been received since the last mailbox transfer order. Data from the last frame received can be read in the CAN_MDxx registers. After setting the MOT field in the CAN_MMR, MRDY is reset to 0.
Transmit	Mailbox data have been transmitted. After setting the MOT field in the CAN_MMR, MRDY is reset to 1.
Consumer	At least one message has been received since the last mailbox transfer order. Data from the first message received can be read in the CAN_MDxx registers. After setting the MOT field in the CAN_MMR, MRDY is reset to 0.
Producer	A remote frame has been received, mailbox data have been transmitted. After setting the MOT field in the CAN_MMR, MRDY is reset to 1.

- **MMI: Mailbox Message Ignored**

0 = No message has been ignored during the previous transfer

1 = At least one message has been ignored during the previous transfer

Cleared by reading the CAN\_MSrx register.

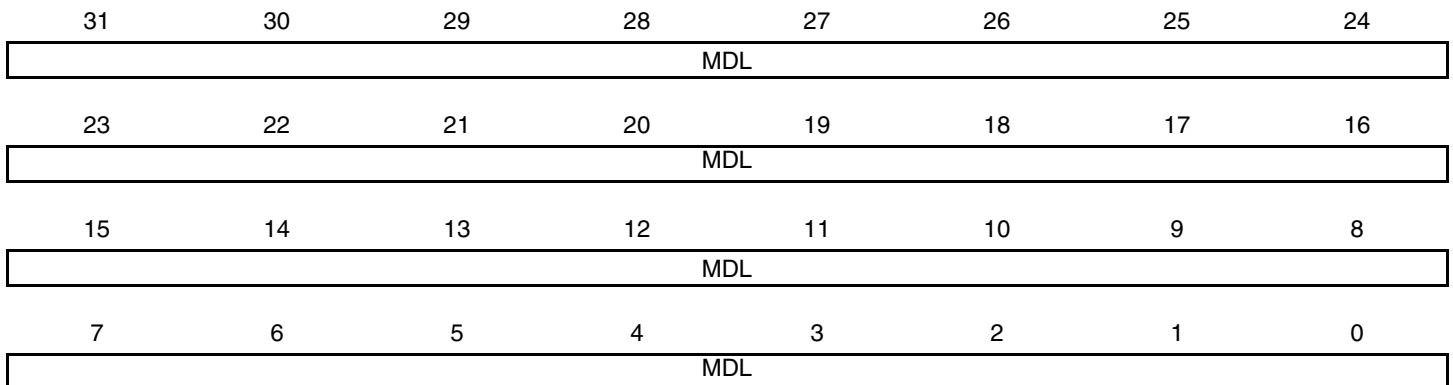
Mailbox Object Type	Description
Receive	Set when at least two messages intended for the mailbox have been sent. The first one is available in the mailbox data register. Others have been ignored. A mailbox with a lower priority may have accepted the message.
Receive with overwrite	Set when at least two messages intended for the mailbox have been sent. The last one is available in the mailbox data register. Previous ones have been lost.
Transmit	Reserved
Consumer	A remote frame has been sent by the mailbox but several messages have been received. The first one is available in the mailbox data register. Others have been ignored. Another mailbox with a lower priority may have accepted the message.
Producer	A remote frame has been received, but no data are available to be sent.



## 36.8.17 CAN Message Data Low Register

**Name:** CAN\_MDLx

**Access Type:** Read/Write



- **MDL: Message Data Low Value**

When MRDY field is set in the CAN\_MSRx register, the lower 32 bits of a received message can be read or written by the software application. Otherwise, the MDL value is locked by the CAN controller to send/receive a new message.

In Receive with overwrite, the CAN controller may modify MDL value while the software application reads MDH and MDL registers. To check that MDH and MDL do not belong to different messages, the application has to check the MMI field in the CAN\_MSRx register. In this mode, the software application must re-read CAN\_MDH and CAN\_MDL, while the MMI bit in the CAN\_MSRx register is set.

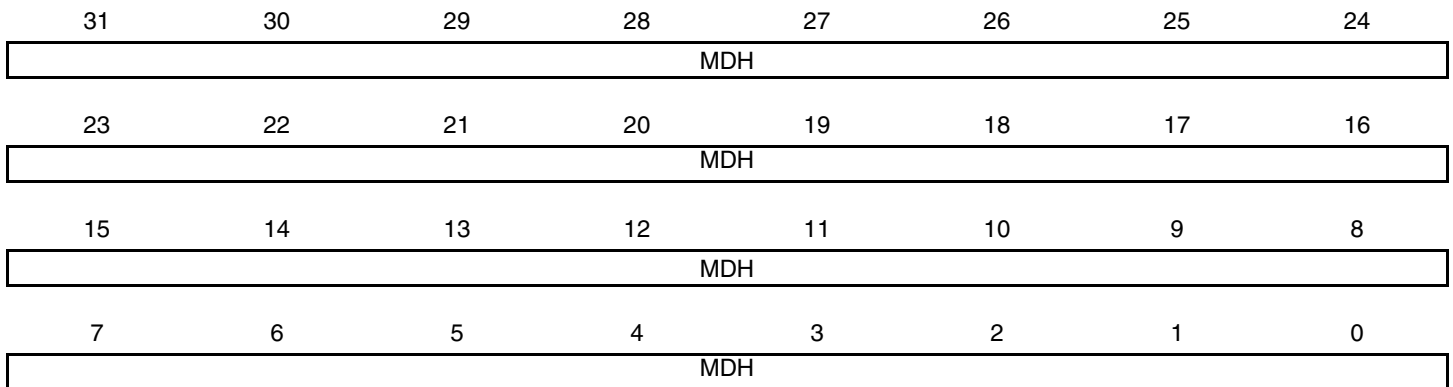
Bytes are received/sent on the bus in the following order:

1. CAN\_MDL[7:0]
2. CAN\_MDL[15:8]
3. CAN\_MDL[23:16]
4. CAN\_MDL[31:24]
5. CAN\_MDH[7:0]
6. CAN\_MDH[15:8]
7. CAN\_MDH[23:16]
8. CAN\_MDH[31:24]

## 36.8.18 CAN Message Data High Register

**Name:** CAN\_MDHx

**Access Type:** Read/Write



- **MDH: Message Data High Value**

When MRDY field is set in the CAN\_MSRx register, the upper 32 bits of a received message are read or written by the software application. Otherwise, the MDH value is locked by the CAN controller to send/receive a new message.

In Receive with overwrite, the CAN controller may modify MDH value while the software application reads MDH and MDL registers. To check that MDH and MDL do not belong to different messages, the application has to check the MMI field in the CAN\_MSRx register. In this mode, the software application must re-read CAN\_MDH and CAN\_MDL, while the MMI bit in the CAN\_MSRx register is set.

Bytes are received/sent on the bus in the following order:

1. CAN\_MDL[7:0]
2. CAN\_MDL[15:8]
3. CAN\_MDL[23:16]
4. CAN\_MDL[31:24]
5. CAN\_MDH[7:0]
6. CAN\_MDH[15:8]
7. CAN\_MDH[23:16]
8. CAN\_MDH[31:24]

## 36.8.19 CAN Message Control Register

**Name:** CAN\_MCRx

**Access Type:** Write-only

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

- **MDLC: Mailbox Data Length Code**

Mailbox Object Type	Description
Receive	No action.
Receive with overwrite	No action.
Transmit	Length of the mailbox message.
Consumer	No action.
Producer	Length of the mailbox message to be sent after the remote frame reception.

- **MRTR: Mailbox Remote Transmission Request**

Mailbox Object Type	Description
Receive	No action
Receive with overwrite	No action
Transmit	Set the RTR bit in the sent frame
Consumer	No action, the RTR bit in the sent frame is set automatically
Producer	No action

Consumer situations can be handled automatically by setting the mailbox object type in Consumer. This requires only one mailbox.

It can also be handled using two mailboxes, one in reception, the other in transmission. The MRTR and the MTCR bits must be set in the same time.

- **MACR: Abort Request for Mailbox x**

Mailbox Object Type	Description
Receive	No action
Receive with overwrite	No action
Transmit	Cancels transfer request if the message has not been transmitted to the CAN transceiver.
Consumer	Cancels the current transfer before the remote frame has been sent.
Producer	Cancels the current transfer. The next remote frame will not be serviced.

It is possible to set MACR field for several mailboxes in the same time, setting several bits to the CAN\_ACR register.

- **MTCR: Mailbox Transfer Command**

Mailbox Object Type	Description
Receive	Allows the reception of the next message.
Receive with overwrite	Triggers a new reception.
Transmit	Sends data prepared in the mailbox as soon as possible.
Consumer	Sends a remote transmission frame.
Producer	Sends data prepared in the mailbox after receiving a remote frame from a Consumer.

This flag clears the MRDY and MABT flags in the CAN\_MSRx register.

When several mailboxes are requested to be transmitted simultaneously, they are transmitted in turn. The mailbox with the highest priority is serviced first. If several mailboxes have the same priority, the mailbox with the lowest number is serviced first (i.e., MBx0 will be serviced before MBx 15 if they have the same priority).

It is possible to set MTCR for several mailboxes at the same time by writing to the CAN\_TCR register.

## 37. AT91SAM7A3 Electrical Characteristics

### 37.1 Absolute Maximum Ratings

**Table 37-1.** Absolute Maximum Ratings\*

Operating Temperature (Industrial).....	-40°C to +85°C
Storage Temperature .....	-60°C to +150°C
Voltage on Input Pins with Respect to Ground .....	-0.3V to +5.5V
Maximum Operating Voltage (VDD1V8 and VDDPLL) .....	1.95V
Maximum Operating Voltage (VDD3V3, VDDBU and VDDANA) .....	3.6V
Total DC Output Current on all I/O lines .....	200 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.

### 37.2 DC Characteristics

The following characteristics are applicable to the operating temperature range:  $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ , unless otherwise specified and are certified for a junction temperature up to  $T_J = 100^\circ\text{C}$  and for VDD3V3 between 3.0 and 3.6V.

**Table 37-2.** DC Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V <sub>VDD1V8</sub>	DC Supply Core		1.65		1.95	V
V <sub>VDDPLL</sub>	DC Supply PLL		1.65		1.95	V
V <sub>VDD3V3</sub>	DC Supply I/Os and Flash		3.0		3.6	V
V <sub>VDDBU</sub>	DC Supply Backup I/O Lines		3.0		3.6	V
V <sub>VDDANA</sub>	DC Supply Analog		3.0		3.6	V
V <sub>IL</sub>	Input Low-level Voltage		-0.3		0.8	V
V <sub>IH</sub>	Input High-level Voltage		2.0		5.5	V
V <sub>OL</sub>	Output Low-level Voltage	I <sub>O</sub> = 2 mA			0.4	V
V <sub>OH</sub>	Output High-level Voltage	I <sub>O</sub> = 2 mA		V <sub>DD3V3</sub> - 0.4		V
I <sub>LEAK</sub>	Input Leakage Current	Pullup resistors disabled (Typ: T <sub>A</sub> = 25°C, Max: T <sub>A</sub> = 85°C)		20	200	nA
I <sub>PULLUP</sub>	Input Pull-up Current	PA0-PA31 PB0-PB29 Connected to ground	143	321	600	μA
I <sub>PULLDOWN</sub>	Input Pull-down Current, (TST, JTAGSEL)	Pins connected to V <sub>VDD3V3</sub>	135	295	550	μA
C <sub>IN</sub>	Input Capacitance	100-pin LQFP Package			14.1	pF

**Table 37-2.** DC Characteristics (Continued)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
I <sub>SC</sub>	Static Current	On V <sub>VDD3V3</sub> = 3.3V, MCK = 500 Hz  All inputs driven at 1 (including TMS, TDI, TCK, NRST) Flash in standby mode All peripherals off	T <sub>A</sub> = 25°C	175	650	μA
			T <sub>A</sub> = 85°C	750	3200	
		On V <sub>VDDBU</sub> = 3.6V  All inputs driven FWKUP, WKUP0, WKUP1 = 0	T <sub>A</sub> = 25°C	8.2	30	μA
			T <sub>A</sub> = 85°C	50	150	
I <sub>O</sub>	Output Current	PA0-PA31, PB0-PB29, NRST			2	mA

**Table 37-3.** 1.8V Voltage Regulator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V <sub>DD3V3</sub>	Supply Voltage		3.0	3.3	3.6	V
V <sub>DD1V8</sub>	Output Voltage		1.65	1.8	1.95	V
I <sub>VDD3V3</sub>	Current Consumption	After startup, no load		70	120	μA
T <sub>START</sub>	Startup Time	C <sub>load</sub> = 2.2 μF, after V <sub>DD3V3</sub> > 3.0V			150	μS
I <sub>O</sub>	Maximum DC Output Current	V <sub>DD3V3</sub> = 3.3V			130	mA

**Table 37-4.** DC Flash Characteristics

Symbol	Parameter	Conditions	Min	Max	Units
I <sub>SB</sub>	Standby current	@ 25°C onto VDD1V8 = 1.8V onto VDD3V3 = 3.3V		10 20	μA
		Random Read @ 35MHz onto VDD1V8 = 1.8V onto VDD3V3 = 3.3V		3.0 0.8	
I <sub>CC</sub>	Active current	Write onto VDD1V8 = 1.8V onto VDD3V3 = 3.3V		400 5.5	μA mA

## 37.3 Power Consumption

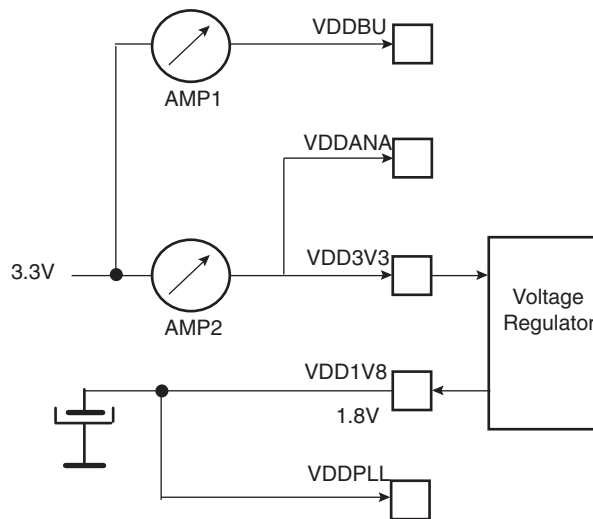
- Typical power consumption of PLLs, Slow Clock and Main Oscillator.
- Power consumption of power supply in three different modes: Active, Ultra Low-power and Backup.
- Power consumption by peripheral: calculated as the difference in current measurement after having enabled then disabled the corresponding clock.

### 37.3.1 Power Consumption versus Modes

The values in [Table 37-5](#) and [Table 37-6](#) on page 552 are measured values of the power consumption with operating conditions as follows:

- $V_{DD3V3} = V_{DDBU} = V_{DDANA} = 3.3V$
- $V_{DD1V8} = V_{DDPLL} = 1.8V$  (internal regulator output)
- $T_A = 25^\circ C$
- USB Pads deactivated
- There is no consumption on the I/Os of the device

**Figure 37-1.** Measures Schematics



These figures represent the power consumption typically measured on the power supplies.

**Table 37-5.** Typical Power Consumption for Different Modes

Mode	Conditions	Consumption	Unit
Active	Flash is read. ARM Core clock is 60 MHz. Analog-to-Digital Converter activated. All peripheral clocks activated. USB transceiver enabled onto AMP2	70	mA
Ultra low power	Flash is in standby mode. ARM Core in idle mode. MCK @ 500 Hz Analog-to-Digital Converter de-activated. All peripheral clocks de-activated. USB transceiver disabled onto AMP2	175	$\mu$ A
Backup	Device only $V_{DDBU}$ powered onto AMP1	8.2	$\mu$ A

**Table 37-6.** Typical Power Consumption by Peripheral in Active Mode

Peripheral	Consumption	Unit
PIO Controller	7.5	NA/MHz
USART	27	
UDP	27	
PWM	34	
CAN	145	
TWI	2	
SPI	16	
MCI	35	
SSC	35	
Timer Counter Channels	2	
ARM7 and System Peripherals	510	

Note:  $V_{VDD3V3} = 3.3V$ ,  $T_A = 25^\circ C$ , MCK = 48 MHz



## 37.4 Crystal Oscillator Characteristics

The following characteristics are applicable to the operating temperature range:  $T_A = -40^{\circ}\text{C}$  to  $85^{\circ}\text{C}$  and worst case of power supply, unless otherwise specified.

### 37.4.1 RC Oscillator Characteristics

**Table 37-7.** RC Oscillator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$1/(t_{\text{CPRC}})$	RC Oscillator Frequency	$V_{\text{DDBU}} = 3\text{V}$	22	32	42	KHz
	Duty Cycle		45	50	55	%
$t_{\text{ST}}$	Startup Time	$V_{\text{DDBU}} = 3\text{V}$			75	$\mu\text{s}$
$I_{\text{OSC}}$	Current Consumption	After Startup Time			2.5	$\mu\text{A}$

### 37.4.2 Main Oscillator Characteristics

**Table 37-8.** Main Oscillator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$1/(t_{\text{CPMAIN}})$	Crystal Oscillator Frequency		3	16	20	MHz
$C_{\text{L1}}, C_{\text{L2}}$	Internal Load Capacitance ( $C_{\text{L1}} = C_{\text{L2}}$ )	Integrated Load Capacitance ((XIN or XOUT))	34	40	46	pF
$C_{\text{L}}$	Equivalent Load Capacitance	Integrated Load Capacitance (XIN and XOUT in series)	17	20	23	pF
	Duty Cycle		30	50	70	%
$t_{\text{ST}}$	Startup Time	$V_{\text{DDPLL}} = 1.2$ to $2\text{V}$ $C_{\text{S}} = 3 \text{ pF}^{(1)}$ $1/(t_{\text{CPMAIN}}) = 3 \text{ MHz}$ $C_{\text{S}} = 7 \text{ pF}^{(1)}$ $1/(t_{\text{CPMAIN}}) = 16 \text{ MHz}$ $C_{\text{S}} = 7 \text{ pF}^{(1)}$ $1/(t_{\text{CPMAIN}}) = 20 \text{ MHz}$			14.5 1.4 1	ms
$I_{\text{DDST}}$	Standby Current Consumption	Standby mode			1	$\mu\text{A}$
$P_{\text{ON}}$	Drive level	@3 MHz @8 MHz @16 MHz @20 MHz			15 30 50 50	$\mu\text{W}$
$I_{\text{DD ON}}$	Current dissipation	@3 MHz <sup>(2)</sup> @8 MHz <sup>(3)</sup> @16 MHz <sup>(4)</sup> @20 MHz <sup>(5)</sup>		150 150 300 400	250 250 450 550	$\mu\text{A}$
$C_{\text{LEXT}}$	Maximum external capacitor on XIN and XOUT				10	pF

- Notes:
- $C_{\text{S}}$  is the shunt capacitance.
  - $R_{\text{S}} = 100\text{-}200 \Omega$ ;  $C_{\text{SHUNT}} = 2.0 - 2.5 \text{ pF}$ ;  $C_{\text{M}} = 2 - 1.5 \text{ fF}$  (typ, worst case) using 1 K ohm serial resistor on xout.
  - $R_{\text{S}} = 50\text{-}100 \Omega$ ;  $C_{\text{SHUNT}} = 2.0 - 2.5 \text{ pF}$ ;  $C_{\text{M}} = 4 - 3 \text{ fF}$  (typ, worst case).
  - $R_{\text{S}} = 25\text{-}50 \Omega$ ;  $C_{\text{SHUNT}} = 2.5 - 3.0 \text{ pF}$ ;  $C_{\text{M}} = 7 - 5 \text{ fF}$  (typ, worst case).
  - $R_{\text{S}} = 20\text{-}50 \Omega$ ;  $C_{\text{SHUNT}} = 3.2 - 4.0 \text{ pF}$ ;  $C_{\text{M}} = 10 - 8 \text{ fF}$  (typ, worst case).

### 37.4.3 Crystal Characteristics

**Table 37-9.** Crystal Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
ESR	Equivalent Series Resistor $R_s$	Fundamental @3 MHz Fundamental @8 MHz Fundamental @16 MHz Fundamental @20 MHz			200 100 80 50	$\Omega$
$C_M$	Motional capacitance				8	fF
$C_{SHUNT}$	Shunt capacitance				7	pF

### 37.4.4 XIN Clock Characteristics

**Table 37-10.** XIN Clock Electrical Characteristics

Symbol	Parameter	Conditions	Min	Max	Units
$1/(t_{CPXIN})$	XIN Clock Frequency	(1)		50.0	MHz
$t_{CPXIN}$	XIN Clock Period	(1)	20.0		ns
$t_{CHXIN}$	XIN Clock High Half-period	(1)	$0.4 \times t_{CPXIN}$	$0.6 \times t_{CPXIN}$	
$t_{CLXIN}$	XIN Clock Low Half-period	(1)	$0.4 \times t_{CPXIN}$	$0.6 \times t_{CPXIN}$	
$C_{IN}$	XIN Input Capacitance	(1)		46	pF
$R_{IN}$	XIN Pull-down Resistor	(1)		500	$k\Omega$
$V_{XIN\_IL}$	$V_{XIN}$ Input Low-level Voltage	(1)	-0.3	$0.2 \times V_{DDPLL}$	V
$V_{XIN\_IH}$	$V_{XIN}$ Input High-level Voltage	(1)	$0.8 \times V_{DDPLL}$	1.95	V
$I_{DDBP}$	Bypass Current Consumption	(1)		15	$\mu W/MHz$

Note: 1. These characteristics apply only when the Main Oscillator is in bypass mode (i.e., when  $MOSCEN = 0$  and  $OSCBYPASS = 1$  in the  $CKGR\_MOR$  register, see the Clock Generator Main Oscillator Register

## 37.4.4.1 XIN Clock Characteristics

**Table 37-11.** XIN Clock Electrical Characteristics

Symbol	Parameter	Conditions	Min	Max	Units
$1/(t_{CPXIN})$	XIN Clock Frequency	(1)		50.0	MHz
$t_{CPXIN}$	XIN Clock Period	(1)	20.0		ns
$t_{CHXIN}$	XIN Clock High Half-period	(1)	$0.4 \times t_{CPXIN}$	$0.6 \times t_{CPXIN}$	
$t_{CLXIN}$	XIN Clock Low Half-period	(1)	$0.4 \times t_{CPXIN}$	$0.6 \times t_{CPXIN}$	
$C_{IN}$	XIN Input Capacitance	(1)		25	pF
$R_{IN}$	XIN Pull-down Resistor	(1)		500	k $\Omega$
$V_{XIN\_IL}$	$V_{XIN}$ Input Low-level Voltage	(1)	-0.3	$0.2 \times V_{DDPLL}$	V
$V_{XIN\_IH}$	$V_{XIN}$ Input High-level Voltage	(1)	$0.8 \times V_{DDPLL}$	1.95	V

Note: 1. These characteristics apply only when the Main Oscillator is in bypass mode (i.e., when MOSCEN = 0 and OSCBYPASS = 1 in the CKGR\_MOR register. (Refer to the “PMC Clock Generator Main CLock Frequency Register”).)

## 37.5 PLL Characteristics

**Table 37-12.** Phase Lock Loop Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$F_{OUT}$	Output Frequency	Field OUT of CKGR_PLL is 00	80		160	MHz
		Field OUT of CKGR_PLL is 10	150		200	MHz
$F_{IN}$	Input Frequency		1		32	MHz
$I_{PLL}$	Current Consumption	Active mode			4	mA
		Standby mode			1	$\mu$ A

Note: Startup time depends on PLL RC filter. A calculation tool is provided by Atmel.

## 37.6 USB Transceiver Characteristics

### 37.6.1 Electrical Characteristics

**Table 37-13.** Electrical Parameters

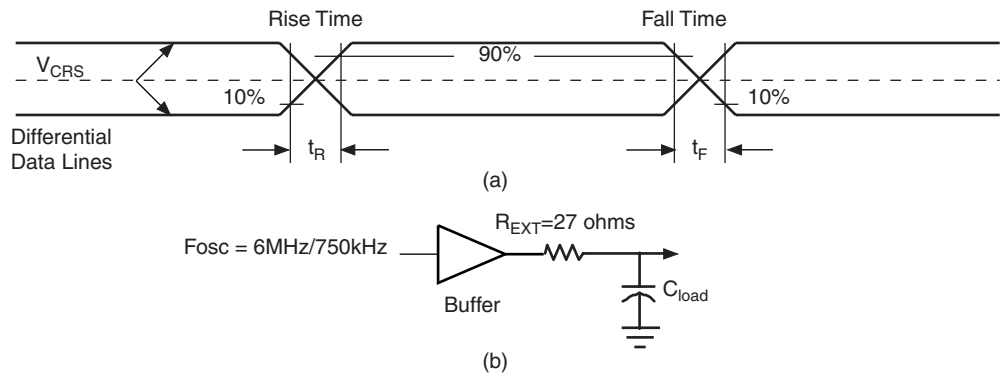
Symbol	Parameter	Conditions	Min	Typ	Max	Unit
<b>Input Levels</b>						
V <sub>IL</sub>	Low Level				0.8	V
V <sub>IH</sub>	High Level		2.0			V
V <sub>DI</sub>	Differential Input Sensitivity	D+ - D-	0.2			V
V <sub>CM</sub>	Differential Input Common Mode Range		0.8		2.5	V
C <sub>IN</sub>	Transceiver capacitance	Capacitance to ground on each line			9.18	pF
I	Hi-Z State Data Line Leakage	0V < V <sub>IN</sub> < 3.3V	-10		+10	μA
R <sub>EXT</sub>	Recommended External USB Series Resistor	In series with each USB pin with ±5%		27		Ω
<b>Output Levels</b>						
V <sub>OL</sub>	Low Level Output	Measured with R <sub>L</sub> of 1.425 kOhm tied to 3.6V	0.0		0.3	V
V <sub>OH</sub>	High Level Output	Measured with R <sub>L</sub> of 14.25 kOhm tied to GND	2.8		3.6	V
V <sub>CRS</sub>	Output Signal Crossover Voltage	Measure conditions described in <a href="#">Figure 37-2</a>	1.3		2.0	V
<b>Consumption</b>						
I <sub>VDD3V3</sub>	Current Consumption	Transceiver enabled in input mode DDP=1 and DDM=0		105	200	μA
I <sub>VDD1V8</sub>	Current Consumption			80	150	μA

### 37.6.2 Switching Characteristics

**Table 37-14.** In Full Speed

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
t <sub>FR</sub>	Transition Rise Time	C <sub>LOAD</sub> = 50 pF	4		20	ns
t <sub>FE</sub>	Transition Fall Time	C <sub>LOAD</sub> = 50 pF	4		20	ns
t <sub>FRFM</sub>	Rise/Fall time Matching		90		111.11	%

**Figure 37-2.** USB Data Signal Rise and Fall Times



## 37.7 Analog-to-Digital Converter Characteristics

**Table 37-15.** Channel Conversion Time and ADC CLock

Parameter	Conditions	Min	Typ	Max	Units
ADC Clock Frequency	10-bit resolution mode			5	MHz
ADC Clock Frequency	8-bit resolution mode			8	MHz
Startup Time	Return from Idle Mode			20	μs
Track and Hold Acquisition Time		600			ns
Conversion Time	ADC Clock = 5 MHz			2	μs
Conversion Time	ADC Clock = 8 MHz			1.25	μs
Throughput Rate	ADC Clock = 5 MHz			384 <sup>(1)</sup>	kSPS
Throughput Rate	ADC Clock = 8 MHz			533 <sup>(2)</sup>	kSPS

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 37-16.** External Voltage Reference Input

Parameter	Conditions	Min	Typ	Max	Units
ADVREF Input Voltage Range		2.6		V <sub>VDD3V3</sub>	V
ADVREF Average Current	On 13 samples with ADC Clock = 5 MHz		200	250	μA
Current Consumption on VDD3V3			0.55	1	mA

**Table 37-17.** Analog Inputs

Parameter	Min	Typ	Max	Units
Input Voltage Range	0		V <sub>ADVREF</sub>	
Input Leakage Current		1		μA
Input Capacitance		12	14	pF

The user can drive ADC input with impedance up to:

- $Z_{OUT} \leq (SHTIM - 470) \times 10$  in 8-bit resolution mode
- $Z_{OUT} \leq (SHTIM - 589) \times 7.69$  in 10-bit resolution mode

with SHTIM (Sample and Hold Time register) expressed in ns and  $Z_{OUT}$  expressed in ohms.

**Table 37-18.** Transfer Characteristics

Parameter	Min	Typ	Max	Units
Resolution		10		Bit
Integral Non-linearity		±1	±3	LSB
Differential Non-linearity		±0.5	±2	LSB
Offset Error		±1	±2	LSB
Gain Error		±0.5	±2	LSB

## 37.8 AC Characteristics

### 37.8.1 Master Clock Characteristics

**Table 37-19.** Master Clock Waveform Parameters

Symbol	Parameter	Conditions	Min	Max	Units
$1/(t_{CPMCK})$	Master Clock Frequency			60	MHz

### 37.8.2 I/O Characteristics

Criteria used to define the maximum frequency of the I/Os:

- output duty cycle (30%-70%)
- minimum output swing: 100 mV to VDD3V3 - 100 mV
- Addition of rising and falling time inferior to 75% of the period

**Table 37-20.** I/O Characteristics

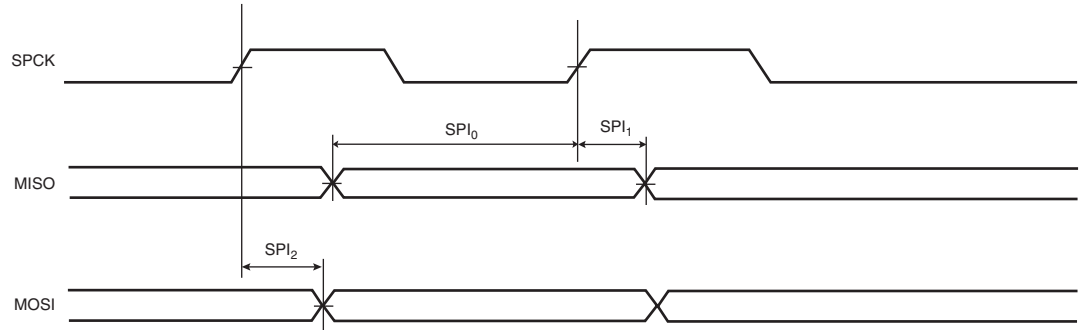
Symbol	Parameter	Conditions	Min	Max	Units
FreqMax <sub>I01</sub>	Pin Group 1 <sup>(1)</sup> frequency	load: 20 pF		16	MHz
PulseminH <sub>I01</sub>	Pin Group 1 <sup>(1)</sup> High Level Pulse Width	load: 20 pF	32		ns
PulseminL <sub>I01</sub>	Pin Group 1 <sup>(1)</sup> Low Level Pulse Width	load: 20 pF	32		ns
FreqMax <sub>I02</sub>	Pin Group 2 <sup>(2)</sup> frequency	load: 20 pF		18	MHz
PulseminH <sub>I02</sub>	Pin Group 2 <sup>(2)</sup> High Level Pulse Width	load: 20 pF	27		ns
PulseminL <sub>I02</sub>	Pin Group 2 <sup>(2)</sup> Low Level Pulse Width	load: 20 pF	27		ns

Notes: 1. Pin Group 1 = PA0 to PA31 and PB0-PB13  
 2. Pin Group 2 = PB14 to PB29

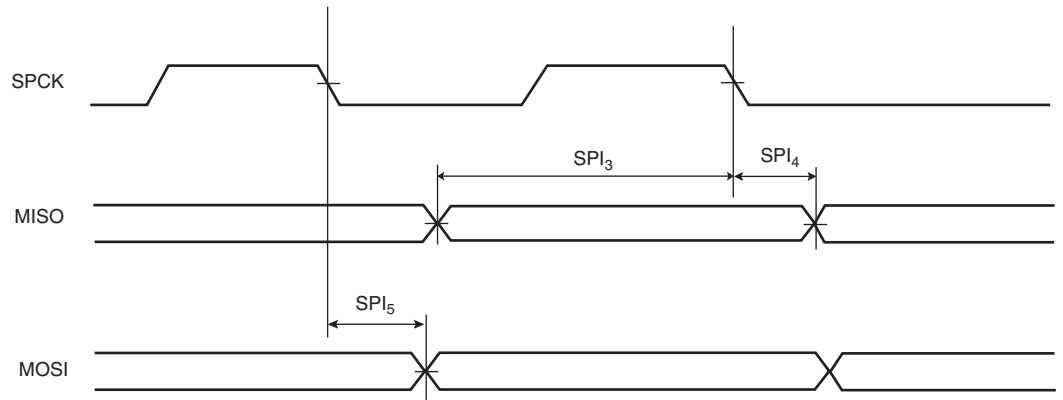


## 37.8.3 SPI Characteristics

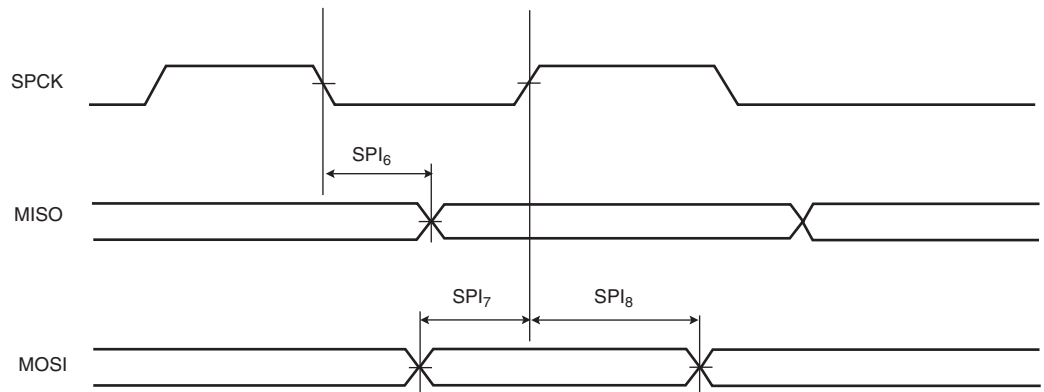
**Figure 37-3.** SPI Master Mode with (CPOL = NCPHA = 0) or (CPOL= NCPHA = 1)



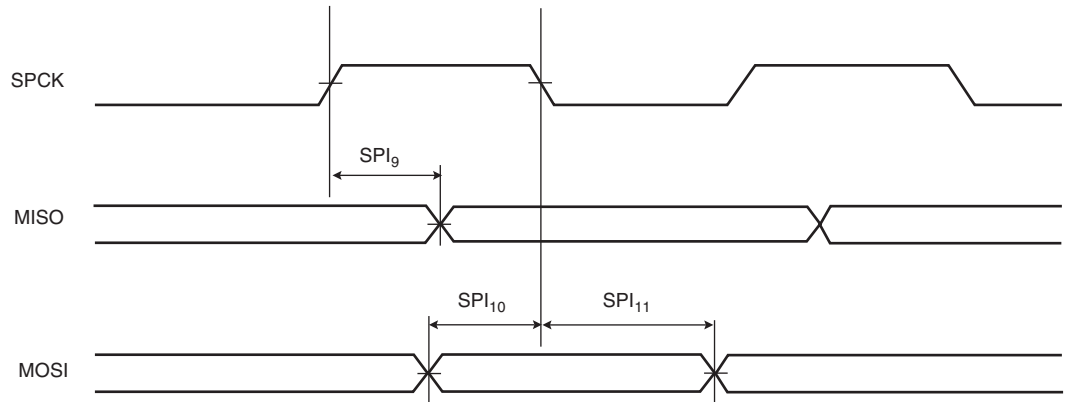
**Figure 37-4.** SPI Master Mode with (CPOL=0 and NCPHA=1) or (CPOL=1 and NCPHA=0)



**Figure 37-5.** SPI Slave Mode with (CPO =0 and NCPH =1) or (CPOL=1 and NCPHA=0)



**Figure 37-6.** SPI Slave Mode with (CPOL = NCPHA = 0) or (CPOL = NCPHA = 1)



**Table 37-21.** SPI Timings

Symbol	Parameter	Conditions	Min	Max	Units
SPI <sub>0</sub>	MISO Setup time before SPCK rises (master)	(1)	$28.5 + (t_{CPMCK})/2^{(2)}$		ns
SPI <sub>1</sub>	MISO Hold time after SPCK rises (master)	(1)	0		ns
SPI <sub>2</sub>	SPCK rising to MOSI Delay (master)	(1)		2	ns
SPI <sub>3</sub>	MISO Setup time before SPCK falls (master)	(1)	$26.5 + (t_{CPMCK})/2^{(2)}$		ns
SPI <sub>4</sub>	MISO Hold time after SPCK falls (master)	(1)	0		ns
SPI <sub>5</sub>	SPCK falling to MOSI Delay (master)	(1)		2	ns
SPI <sub>6</sub>	SPCK falling to MISO Delay (slave)	(1)		28	ns
SPI <sub>7</sub>	MOSI Setup time before SPCK rises (slave)	(1)	2		ns
SPI <sub>8</sub>	MOSI Hold time after SPCK rises (slave)	(1)	3		ns
SPI <sub>9</sub>	SPCK rising to MISO Delay (slave)	(1)		28	ns
SPI <sub>10</sub>	MOSI Setup time before SPCK falls (slave)	(1)	3		ns
SPI <sub>11</sub>	MOSI Hold time after SPCK falls (slave)	(1)	3		ns

- Note:
1. Maximum external capacitor = 20 pF.
  2.  $t_{CPMCK}$ : Master Clock period in ns.

Note that in SPI master mode the ATSAM7SE512/256/32 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 37-3](#) and [Figure 37-4](#).

## 37.8.4 Embedded Flash Characteristics

The maximum operating frequency is given in [Table 37-22](#) but is limited by the Embedded Flash access time when the processor is fetching code out of it. [Table 37-22](#) gives the device maximum operating frequency depending on the field FWS of the MC\_FMR register. This field defines the number of wait states required to access the Embedded Flash Memory.

**Table 37-22.** Embedded Flash Wait States

FWS	Read Operations	Maximum Operating Frequency (MHz)
0	1 cycle	30
1	2 cycles	60
2	3 cycles	60
3	4 cycles	60

**Table 37-23.** AC Flash Characteristics

Parameter	Condition	Min	Max	Units
Program Cycle Time	per page including auto-erase		6	ms
	per page including auto-erase		3	ms
Full Chip Erase		15		ms

### 37.8.5 JTAG/ICE Interface Timings and Signals

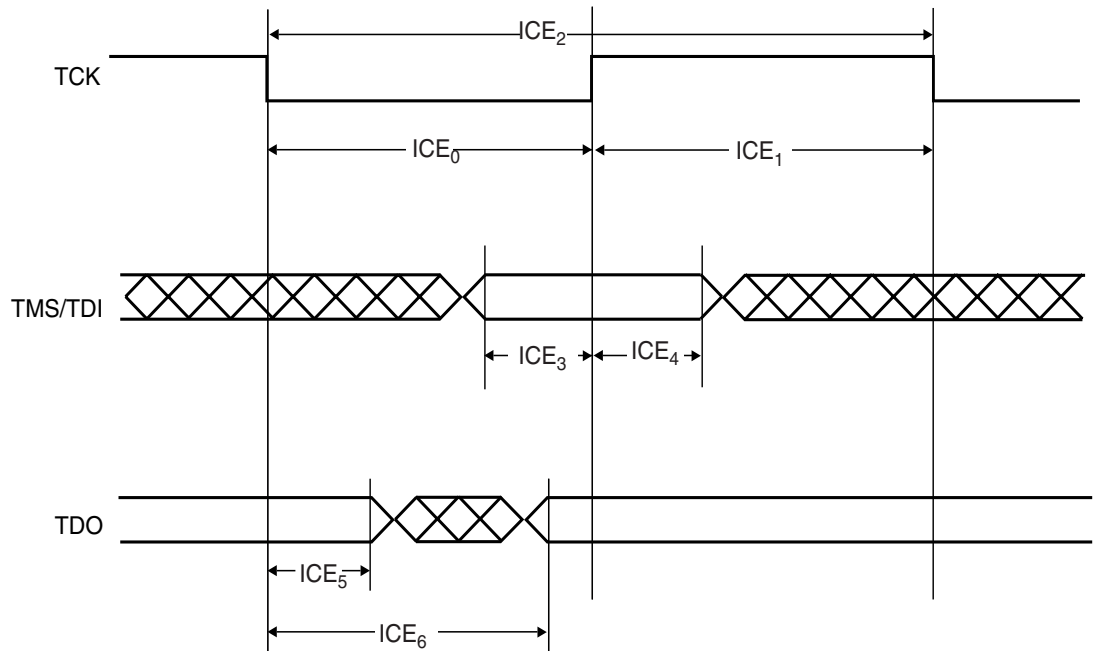
#### 37.8.5.1 ICE Interface Timings and Signals

**Table 37-24.** ICE Interface Timing Specification

Symbol	Parameter	Conditions	Min	Max	Units
ICE <sub>0</sub>	TCK Low Half-period	(1)	51		ns
ICE <sub>1</sub>	TCK High Half-period	(1)	51		ns
ICE <sub>2</sub>	TCK Period	(1)	102		ns
ICE <sub>3</sub>	TDI, TMS, Setup before TCK High	(1)	0		ns
ICE <sub>4</sub>	TDI, TMS, Hold after TCK High	(1)	3		ns
ICE <sub>5</sub>	TDO Hold Time	(1)	13		ns
ICE <sub>6</sub>	TCK Low to TDO Valid	(1)		20	ns

Note: 1.  $V_{DD3V3}$  from 3.0V to 3.6V, maximum external capacitor = 40pF

**Figure 37-7.** ICE Interface Signals



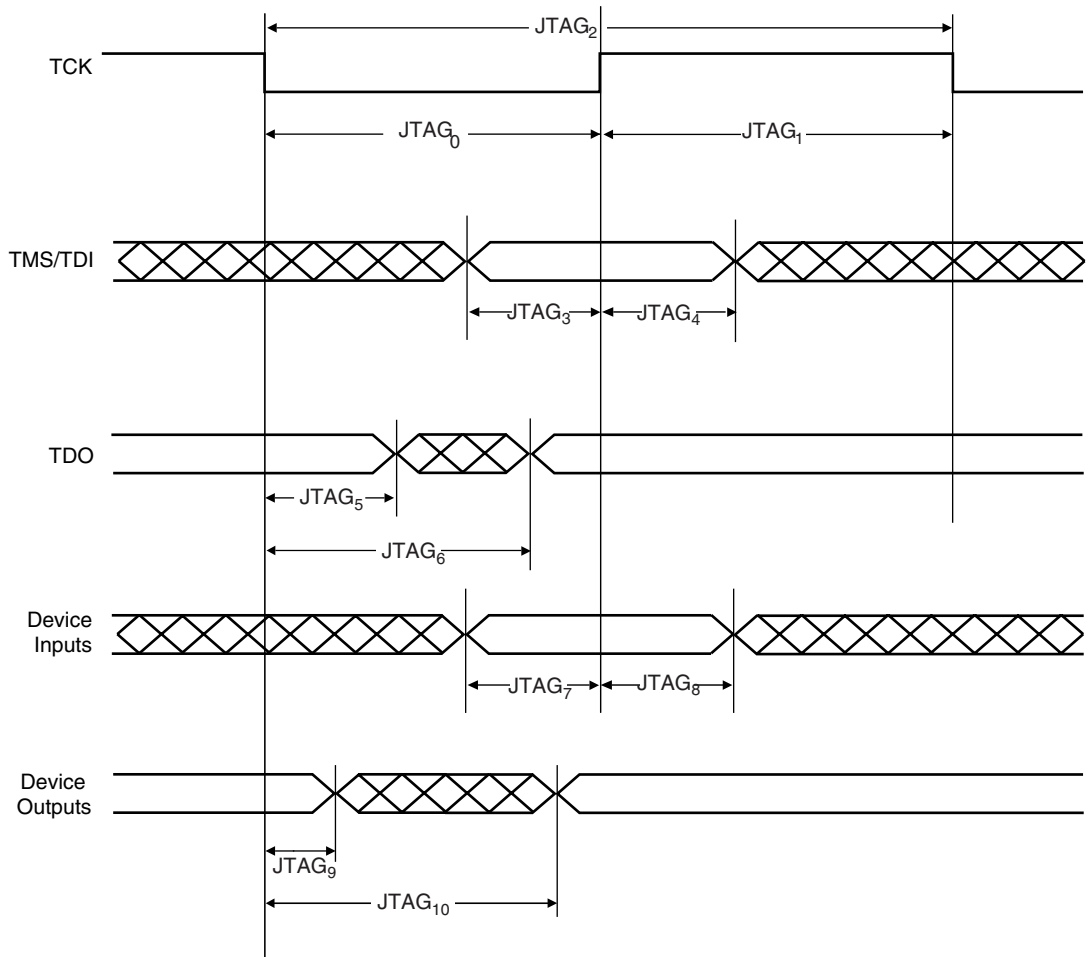
## 37.8.5.2 JTAG Interface Timings and Signals

**Table 37-25.** JTAG Interface Timing Specification

Symbol	Parameter	Conditions	Min	Max	Units
JTAG <sub>0</sub>	TCK Low Half-period	(1)	6.5		ns
JTAG <sub>1</sub>	TCK High Half-period	(1)	5.5		ns
JTAG <sub>2</sub>	TCK Period	(1)	12		ns
JTAG <sub>3</sub>	TDI, TMS Setup before TCK High	(1)	2		ns
JTAG <sub>4</sub>	TDI, TMS Hold after TCK High	(1)	3		ns
JTAG <sub>5</sub>	TDO Hold Time	(1)	4		ns
JTAG <sub>6</sub>	TCK Low to TDO Valid	(1)		16	ns
JTAG <sub>7</sub>	Device Inputs Setup Time	(1)	0		ns
JTAG <sub>8</sub>	Device Inputs Hold Time	(1)	3		ns
JTAG <sub>9</sub>	Device Outputs Hold Time	(1)	6		ns
JTAG <sub>10</sub>	TCK to Device Outputs Valid	(1)		18	ns

Note: 1.  $V_{DD3V3}$  from 3.0V to 3.6V, maximum external capacitor = 40pF

**Figure 37-8.** JTAG Interface Signals



## 38. AT91SAM7A3 Mechanical Characteristics

### 38.1 Thermal Considerations

#### 38.1.1 Thermal Data

[Table 38-1](#) summarizes the thermal resistance data depending on the package.

**Table 38-1.** Thermal Resistance Data

Symbol	Parameter	Condition	Package	Typ	Unit
$\theta_{JA}$	Junction-to-ambient thermal resistance	Still Air	LQFP100	38.3	°C/W
$\theta_{JC}$	Junction-to-case thermal resistance		LQFP100	8.7	

#### 38.1.2 Junction Temperature

The average chip-junction temperature,  $T_J$ , in °C can be obtained from the following:

1.  $T_J = T_A + (P_D \times \theta_{JA})$
2.  $T_J = T_A + (P_D \times (\theta_{HEATSINK} + \theta_{JC}))$

where:

- $\theta_{JA}$  = package thermal resistance, Junction-to-ambient (°C/W), provided in [Table 38-1 on page 566](#).
- $\theta_{JC}$  = package thermal resistance, Junction-to-case thermal resistance (°C/W), provided in [Table 38-1 on page 566](#).
- $\theta_{HEAT\ SINK}$  = cooling device thermal resistance (°C/W), provided in the device datasheet.
- $P_D$  = device power consumption (W) estimated from data provided in the section "[Power Consumption](#)" on page 551.
- $T_A$  = ambient temperature (°C).



**Table 38-2.** 100-lead LQFP Package Dimensions

Symbol	Millimeter			Inch		
	Min	Nom	Max	Min	Nom	Max
A			1.60			0.63
A1	0.05		0.15	0.002		0.006
A2	1.35	1.40	1.45	0.053	0.055	0.057
D	16.00 BSC			0.630 BSC		
D1	14.00 BSC			0.551 BSC		
E	16.00 BSC			0.630 BSC		
E1	14.00 BSC			0.551 BSC		
R2	0.08		0.20	0.003		0.008
R1	0.08			0.003		
Q	0°	3.5°	7°	0°	3.5°	7°
Θ1	0°			0°		
Θ2	11°	12°	13°	11°	12°	13°
Θ3	11°	12°	13°	11°	12°	13°
c	0.09		0.20	0.004		0.008
L	0.45	0.60	0.75	0.018	0.024	0.030
L1	1.00 REF			0.039 REF		
S	0.20			0.008		
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		

**Table 38-3.** Device and 100-lead LQFP Package Maximum Weight

800	mg
-----	----

**Table 38-4.** 100-lead LQFP Package Characteristics

Moisture Sensitivity Level	3
----------------------------	---



## 38.3 Soldering Profile

Table 38-5 gives the recommended soldering profile from J-STD-020C.

**Table 38-5.** 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.



### 39. AT91SAM7A3 Ordering Information

Table 39-1. Ordering Information

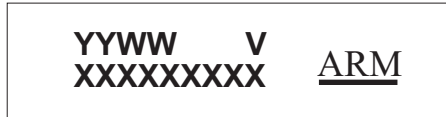
Ordering Code	Package	Package Type	Temperature Operating Range
AT91SAM7A3-AU	LQFP 100	Green	Industrial (-40°C to 85°C)

## 40. Errata

### 40.1 Marking

All devices are marked with the Atmel logo and the ordering code.

Additional marking has the following format:



where

- “YY”: manufactory year
- “WW”: manufactory week
- “V”: revision

“XXXXXXXXX”: lot number

## 40.2 AT91SAM7A3 Errata - Rev. A Parts

Refer to [Section 40.1 "Marking"](#), on page 571

### 40.2.1 MultiMedia Card Interface (MCI)

#### 40.2.1.1 MCI: Data Timeout Error Flag

As the data Timeout Error Flag cannot rise, the MCI can be stalled waiting indefinitely the Data start bit.

##### **Problem Fix/Workaround**

A Software Timeout must be implemented and a STOP command must be sent when the time-out has elapsed.

#### 40.2.1.2 MCI: STREAM Command Not Supported

The STREAM READ/WRITE commands are not supported by the MCI.

##### **Problem Fix/Workaround**

None.

#### 40.2.1.3 MCI: Data FIFO and Status Bits

Do not read/write the Data FIFO if RXRDY/TXRDY status bits are not set.

##### **Problem Fix/Workaround**

None.

#### 40.2.1.4 MCI: DATA FIFO Problem with PDC

The shared FIFO is reset at the beginning of a transfer command.

##### **Problem Fix/Workaround**

In order not to lose data, it is mandatory to enable the PDC channel after having written to the command register. In order to achieve this sequence correctly it is mandatory to disable all IT sources.

### 40.2.2 Parallel Input/Output Controller (PIO)

#### 40.2.2.1 PIO: Leakage on PB14 - PB29

When PB14 - PB29, (the I/O lines multiplexed with the analog inputs) are set as digital inputs with pull-up disabled, the leakage can be 25  $\mu$ A in worst case and 90 nA in typical case per I/O when the I/O is set externally at low level.

##### **Problem Fix/Workaround**

Set the I/O to VDDIO by internal or external pull-up.

## 40.2.2.2 PIO: Electrical Characteristics PA0-PA31 and PB0-PB13

When PA0-PA31 or PB0-PB13 are set as digital inputs with pull-up enabled, the voltage of the I/O stabilizes at VPull-up.

### Vpull-up

VPull-up Min	VPull-up Max
VDDIO - 0.65 V	VDDIO - 0.45 V

This condition causes a leakage through VDDIO. This leakage is 45  $\mu$ A per pad in worst case at 3.3 V.

### I Leakage

Parameter	Typ	Max
I Leakage at 3.3V	2.5 $\mu$ A	45 $\mu$ A

### Problem Fix/Workaround

It is recommended to use an external pull-up if needed.

## 40.2.2.3 PIO: Drive Low PA0-PA31 and PB0-PB13

When NRST or PA0-PA31 and or PB0-PB13 are set as digital inputs with pull-up enabled, driving the I/O with an output impedance higher than 500 ohms may not drive the I/O to a logical zero.

### Problem Fix/Workaround

Output impedance must be lower than 500 ohms.

## 40.2.3 Power Management Controller (PMC)

### 40.2.3.1 PMC: Main Oscillator Measure in Bypass Mode

It is not possible to measure the main oscillator in bypass mode (number of Main Clock cycles during 16 periods of slow clock set in CKGR\_MCFR). Measurement is possible with an external crystal.

### Problem Fix/Workaround

None.

## 40.2.4 Pulse Width Modulation (PWM) Controller

### 40.2.4.1 PWM: Constraints on Duty Cycle Value

A value of 0 is forbidden in the Channel Duty Cycle Register (PWM\_CDTYx).

### Problem Fix/Workaround

0 corresponds to a permanent high or low signal. The PIO controller may ensure this level when needed by disabling PWM, and using the corresponding I/O as an output with a value 0 or 1.

### 40.2.4.2 PWM: Update when PWM\_CCNTx = 0 or 1

If the Channel Counter Register value is 0 or 1, the Channel Period Register or Channel Duty Cycle Register is directly modified when writing the Channel Update Register.

### **Problem Fix/Workaround**

Check the Channel Counter Register before writing the update register.

#### **40.2.4.3**     *PWM: Update when PWM\_CPRDx = 0*

When Channel Period Register equals 0, the period update is not operational.

### **Problem Fix/Workaround**

Do not write 0 in the period register.

#### **40.2.4.4**     *PWM: Counter Start Value*

In left-aligned mode, the first start value of the counter is 0. For the other periods, the counter starts at 1.

### **Problem Fix/Workaround**

None.

#### **40.2.4.5**     *PWM: Behavior of CHIDx Status Bits in the PWM\_SR Register*

Erratic behavior of the CHIDx status bit in the PWM\_SR Register. When a channel is disabled by writing in the PWM\_DIS Register just after enabling it (before completion of a Clock Period of the clock selected for the channel), the PWM line is internally disabled but the CHIDx status bit in the PWM\_SR stays at 1.

### **Problem Fix/Workaround**

Do not disable a channel before completion of one period of the selected clock.

### **40.2.5**     **Real Time Timer: RTT**

#### **40.2.5.1**     *RTT: Possible Event Loss when Reading RTT\_SR*

If an event (RTTINC or ALMS) occurs within the same slow clock cycle during which the RTT\_SR is read, the corresponding bit might be cleared. This can lead to the loss of this event.

### **Problem Fix/Workaround:**

The software must handle the RTT event as an interrupt and should not poll RTT\_SR.

### **40.2.6**     **Serial Peripheral Interface (SPI)**

#### **40.2.6.1**     *SPI: Pulse Generation on SPCK*

In Master Mode, there is an additional pulse generated on SPCK when the SPI is configured as follows:

- The Baudrate is odd and different from 1
- The polarity is set to 1
- The phase is set to 0

### **Problem Fix/Workaround**

None.

## 40.2.6.2 SPI: Bad tx\_ready Behavior when CSAAT = 1 and SCBR = 1

If the SPI2 is programmed with CSAAT = 1, SCBR(baudrate) = 1 and two transfers are performed consecutively on the same slave with an IDLE state between them, the tx\_ready signal does not rise after the second data has been transferred in the shifter. This can imply for example, that the second data is sent twice.

### **Problem Fix/Workaround**

Do not use the combination CSAAT = 1 and SCBR = 1.

## 40.2.6.3 SPI: LASTXFER (Last Transfer) Behavior

In FIXED Mode, with CSAAT bit set, and in “PDC mode” the Chip Select can rise depending on the data written in the SPI\_TDR when the TX\_EMPTY flag is set. If for example, the PDC writes a “1” in the bit 24 (LASTXFER bit) of the SPI\_TDR, the chip select will rise as soon as the TXEMPTY flag is set.

### **Problem Fix/Workaround**

Use the CS in PIO mode when PDC mode is required and CS has to be maintained between transfers.

## 40.2.6.4 SPI: SPCK Behavior in Master Mode

SPCK pin can toggle out before the first transfer in Master Mode.

### **Problem Fix/Workaround**

In Master Mode, MSTR bit must be set (in SPI\_MR register) before configuring SPI\_CSRx registers.

## 40.2.6.5 SPI: Chip Select and Fixed Mode

In fixed Mode, if a transfer is performed through a PDC on a Chip select different from the Chip select 0, the output spi\_size sampled by the PDC will depend on the field, BITS (Bits per Transfer) of SPI\_CSR0 register, whatever the selected Chip select is. For example, if SPI\_CSR0 is configured for a 10-bit transfer whereas SPI\_CSR1 is configured for an 8-bit transfer, when a transfer is performed in Fixed mode through the PDC, on Chip select 1, the transfer will be considered as a HalfWord transfer.

### **Problem Fix/Workaround**

If a PDC transfer has to be performed in 8 bits, on a Chip select y (y as different from 0), the BITS field of the SPI\_CSR0 must be configured in 8 bits, in the same way as the BITS field of the CSRy Register.

## 40.2.6.6 SPI: Baudrate Set to 1

When Baudrate is set at 1 (i.e. when serial clock frequency equals the system clock frequency) and when the BITS field of the SPI\_CSR register (number of bits to be transmitted) equals an ODD value (in this case 9,11,13 or 15), an additional pulse will be generated on output SPCK.

Everything is OK if the BITS field equals 8,10,12,14 or 16 and Baudrate = 1.

### **Problem Fix/Workaround**

None.

#### 40.2.6.7 *SPI: Disable Issue*

The SPI Command “SPI Disable” is not possible during a transfer, it must be performed only after TX\_EMPTY rising else there is everlasting dummy transfers occur.

##### **Problem Fix/Workaround**

None.

#### 40.2.6.8 *SPI: Disable In Slave Mode*

The SPI disable is not possible in slave mode.

##### **Problem Fix/Workaround**

Read first the received data, then perform the software reset.

#### 40.2.6.9 *SPI: Software Reset and SPIEN Bit*

The SPI Command “software reset” does not reset the SPIEN config bit. Therefore rewriting an SPI enable command does not set TX\_READY, TX\_EMPTY flags.

##### **Problem Fix/Workaround**

Send SPI disable command after a software reset.

#### 40.2.6.10 *SPI: CSAAT = 1 and Delay*

If CSAAT = 1 for current access and there is no more TX request for a time greater than DLYBCT + DLYBCS, then if an access is requested on another slave, the NPCS bus switches from one CS to the one requested without DLYBCS. External Slaves may reach a contention on SPI\_MISO line for a short period.

##### **Problem Fix/Workaround**

Assert the Last Transfer Command (NPCS de-activation) for the last character of each slave.

### 40.2.7 **Synchronous Serial Controller (SSC)**

#### 40.2.7.1 *SSC: Periodic Transmission Limitations in Master Mode*

If the Least Significant Bit is sent first (MSBF = 0), the first TAG during the frame synchro is not sent.

##### **Problem Fix/Workaround**

None.

#### 40.2.7.2 *SSC: Transmitter Limitations in Slave Mode*

If TK is programmed as output and TF is programmed as input, it is impossible to emit data when the start of edge (rising or falling) of synchro has a Start Delay equal to zero.

##### **Problem Fix/Workaround**

None.

#### 40.2.7.3 *SSC: Transmitter Limitations in Slave Mode*

If TK is programmed as an input and TF is programmed as an output and requested to be set to low/high during data emission, the Frame Synchro signal is generated one bit clock period after

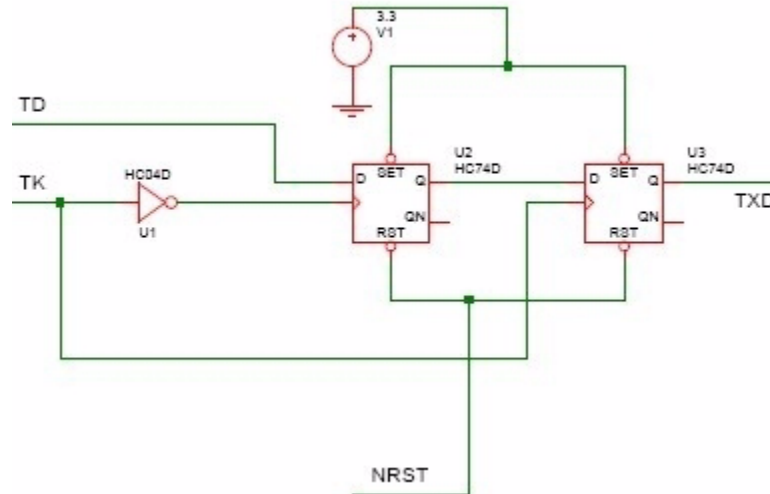


the data start and one data bit is lost. This problem does not exist when generating a periodic synchro.

### Problem Fix/Workaround

The data need to be delayed for one bit clock period with an external assembly.

In the following schematic, TD, TK and NRST are AT91SAM7A3 signals, TXD is the delayed data to connect to the device.



## 40.2.8 Two-wire Interface (TWI)

### 40.2.8.1 TWI: Clock Divider

The value of  $CLDIV \times 2^{CKDIV}$  must be less than or equal to 8191, the value of  $CHDIV \times 2^{CKDIV}$  must be less than or equal to 8191.

### Problem Fix/Workaround

None.

### 40.2.8.2 TWI: Possible Receive Holding Register Corruption

When loading the TWI\_RHR, the transfer direction is ignored. The last data byte received in the TWI\_RHR is corrupted at the end of the first subsequent transmit data byte. Neither RXRDY nor OVERRUN status bits are set if this occurs.

### Problem Fix/Workaround

The user must be sure that received data is read before transmitting any new data.

### 40.2.8.3 TWI: NACK Status Bit Lost

During a master frame, if TWI\_SR is read between the Non Acknowledge condition detection and the TXCOMP bit rising in the TWI\_SR, the NACK bit is not set.

### Problem Fix/Workaround

The user must wait for the TXCOMP status bit by interrupt and must not read the TWI\_SR as long as transmission is not completed.

Note: TXCOMP and NACK fields are set simultaneously and the NACK field is reset after the read of the TWI\_SR.

#### 40.2.8.4 TWI: Disabling Does not Operate Correctly

Any transfer in progress is immediately frozen if the Control Register (TWI\_CR) is written with the bit MSDIS at 1. Furthermore, the status bits TXCOMP and TXRDY in the Status Register (TWI\_SR) are not reset.

##### **Problem Fix/Workaround**

The user must wait for the end of transfer before disabling the TWI. In addition, the interrupts must be disabled before disabling the TWI.

#### 40.2.8.5 TWI: Software Reset

when a software reset is performed during a frame and when TWCK is low, it is impossible to initiate a new transfer in READ or WRITE mode.

##### **Problem Fix/Workaround**

None.

### 40.2.9 Universal Synchronous Asynchronous Receiver Transmitter (USART)

#### 40.2.9.1 USART: Hardware Handshaking – Two Characters Sent

If CTS switches from 0 to 1 during the TX of a character and if the holding register (US\_THR) is not empty, the content of US\_THR will also be transmitted.

##### **Problem Fix/Workaround**

Don't use the PDC in transmit mode and do not fill US\_THR before TXEMPTY is set at 1.

#### 40.2.9.2 USART: XOFF Character Bad Behavior

The XOFF character is sent only when the receive buffer is detected full. While the XOFF is being sent, the remote transmitter is still transmitting. As only one Holding register is available in the receiver, characters will be lost in reception. This makes the software handshaking functionality ineffective.

##### **Problem Fix/Workaround**

None.

### 40.2.10 VDDBU

#### 40.2.10.1 VDDBU: Idd When Accessing the APB

During access to APB (via any peripheral) the current consumption on VDDBU increases.

It typically adds 20  $\mu$ A.

##### **Problem Fix/Workaround**

Minimize the APB access if the current consumption over VDDBU is a key point in the application.

## 40.2.10.2 *VDDBU: Idd When VDDCORE Falls*

When VDDCORE rises or falls, there is a dynamic power consumption over VDDBU.

This transient consumption is typically 400  $\mu$ A during 10 ms.

### **Problem Fix/Workaround**

Minimize the number of power-on and power-off on VDDCORE if the current consumption over VDDBU is a key point in the application.



## Table of Contents

	<b>Features .....</b>	<b>1</b>
<b>1</b>	<b>Description .....</b>	<b>3</b>
<b>2</b>	<b>Block Diagram .....</b>	<b>4</b>
<b>3</b>	<b>Signal Description .....</b>	<b>5</b>
<b>4</b>	<b>Package .....</b>	<b>8</b>
	4.1 100-lead LQFP Package Outline .....	8
	4.2 Pinout .....	9
<b>5</b>	<b>Power Considerations .....</b>	<b>10</b>
	5.1 Power Supplies .....	10
	5.2 Voltage Regulator .....	10
	5.3 Typical Powering Schematics .....	11
<b>6</b>	<b>I/O Lines Considerations .....</b>	<b>12</b>
	6.1 JTAG Port Pins .....	12
	6.2 Test Pin .....	12
	6.3 Reset Pin .....	12
	6.4 PIO Controller A and B Lines .....	12
	6.5 Shutdown Logic Pins .....	12
	6.6 I/O Line Drive Levels .....	12
<b>7</b>	<b>Processor and Architecture .....</b>	<b>13</b>
	7.1 ARM7TDMI Processor .....	13
	7.2 Debug and Test Features .....	13
	7.3 Memory Controller .....	13
	7.4 Peripheral DMA Controller .....	14
<b>8</b>	<b>Memory .....</b>	<b>15</b>
	8.1 Embedded Memories .....	15
	8.2 Memory Mapping .....	17
	8.3 Embedded Flash .....	17
<b>9</b>	<b>System Controller .....</b>	<b>19</b>
	9.1 System Controller Mapping .....	20



9.2	Reset Controller .....	21
9.3	Clock Generator .....	21
9.4	Power Management Controller .....	21
9.5	Advanced Interrupt Controller .....	22
9.6	Debug Unit .....	23
9.7	Period Interval Timer .....	23
9.8	Watchdog Timer .....	23
9.9	Real-time Timer .....	23
9.10	Shutdown Controller .....	23
9.11	PIO Controllers A and B .....	23
<b>10</b>	<b>Peripherals .....</b>	<b>25</b>
10.1	Peripheral Mapping .....	25
10.2	Peripheral Multiplexing on PIO Lines .....	26
10.3	PIO Controller A Multiplexing .....	27
10.4	PIO Controller B Multiplexing .....	28
<b>11</b>	<b>Peripheral Identifiers .....</b>	<b>29</b>
11.1	Serial Peripheral Interface .....	30
11.2	Two-wire Interface .....	30
11.3	USART .....	30
11.4	Serial Synchronous Controller .....	31
11.5	Timer Counter .....	31
11.6	PWM Controller .....	31
11.7	USB Device Port .....	32
11.8	Multimedia Card Interface .....	32
11.9	CAN Controller .....	32
11.10	Analog-to-Digital Converter .....	33
<b>12</b>	<b>ARM7TDMI Processor .....</b>	<b>35</b>
12.1	Overview .....	35
12.2	ARM7TDMI Processor .....	35
<b>13</b>	<b>AT91SAM7A3 Debug and Test Features .....</b>	<b>41</b>
13.1	Overview .....	41
13.2	Block Diagram .....	41
13.3	Application Examples .....	42
13.4	Test Environment .....	43
13.5	Debug and Test Pin Description .....	43

13.6	Functional Description .....	44
<b>14</b>	<b>Reset Controller (RSTC) .....</b>	<b>53</b>
14.1	Overview .....	53
14.2	Block Diagram .....	53
14.3	Functional Description .....	53
14.4	Reset Controller (RSTC) User Interface .....	61
<b>15</b>	<b>Real-time Timer (RTT) .....</b>	<b>65</b>
15.1	Overview .....	65
15.2	Block Diagram .....	65
15.3	Functional Description .....	65
15.4	Real-time Timer (RTT) User Interface .....	67
<b>16</b>	<b>Periodic Interval Timer (PIT) .....</b>	<b>71</b>
16.1	Overview .....	71
16.2	Block Diagram .....	71
16.3	Functional Description .....	71
16.4	Periodic Interval Timer (PIT) User Interface .....	73
<b>17</b>	<b>Watchdog Timer (WDT) .....</b>	<b>77</b>
17.1	Overview .....	77
17.2	Block Diagram .....	77
17.3	Functional Description .....	78
17.4	Watchdog Timer (WDT) User Interface .....	80
<b>18</b>	<b>Shutdown Controller (SHDWC) .....</b>	<b>83</b>
18.1	Overview .....	83
18.2	Block Diagram .....	83
18.3	I/O Lines Description .....	84
18.4	Product Dependencies .....	84
18.5	Functional Description .....	84
18.6	Shutdown Controller (SHDWC) User Interface .....	86
<b>19</b>	<b>Memory Controller (MC) .....</b>	<b>89</b>
19.1	Overview .....	89
19.2	Block Diagram .....	89
19.3	Functional Description .....	90
19.4	Memory Controller (MC) User Interface .....	94
<b>20</b>	<b>Embedded Flash Controller (EFC) .....</b>	<b>101</b>



20.1	Overview .....	101
20.2	Functional Description .....	101
20.3	Embedded Flash Controller (EFC) User Interface .....	109
<b>21</b>	<b><i>Peripheral DMA Controller (PDC) .....</i></b>	<b>115</b>
21.1	Overview .....	115
21.2	Block Diagram .....	115
21.3	Functional Description .....	116
21.4	Peripheral DMA Controller (PDC) User Interface .....	118
<b>22</b>	<b><i>Advanced Interrupt Controller (AIC) .....</i></b>	<b>125</b>
22.1	Overview .....	125
22.2	Block Diagram .....	125
22.3	Application Block Diagram .....	126
22.4	AIC Detailed Block Diagram .....	126
22.5	I/O Line Description .....	126
22.6	Product Dependencies .....	127
22.7	Functional Description .....	128
22.8	Advanced Interrupt Controller (AIC) User Interface .....	140
<b>23</b>	<b><i>Clock Generator .....</i></b>	<b>151</b>
23.1	Description .....	151
23.2	Slow Clock RC Oscillator .....	151
23.3	Main Oscillator .....	151
23.4	Divider and PLL Block .....	153
<b>24</b>	<b><i>Power Management Controller (PMC) .....</i></b>	<b>155</b>
24.1	Description .....	155
24.2	Master Clock Controller .....	155
24.3	Processor Clock Controller .....	156
24.4	USB Clock Controller .....	156
24.5	Peripheral Clock Controller .....	156
24.6	Programmable Clock Output Controller .....	157
24.7	Programming Sequence .....	157
24.8	Clock Switching Details .....	161
24.9	Power Management Controller (PMC) User Interface .....	164
<b>25</b>	<b><i>Debug Unit (DBGU) .....</i></b>	<b>181</b>
25.1	Overview .....	181



25.2	Block Diagram .....	182
25.3	Product Dependencies .....	183
25.4	UART Operations .....	183
25.5	Debug Unit User Interface .....	190
<b>26</b>	<b><i>Parallel Input Output Controller (PIO)</i></b> .....	<b>205</b>
26.1	Overview .....	205
26.2	Block Diagram .....	206
26.3	Product Dependencies .....	207
26.4	Functional Description .....	208
26.5	I/O Lines Programming Example .....	213
26.6	User Interface .....	214
<b>27</b>	<b><i>Serial Peripheral Interface (SPI)</i></b> .....	<b>231</b>
27.1	Overview .....	231
27.2	Block Diagram .....	232
27.3	Application Block Diagram .....	233
27.4	Signal Description .....	234
27.5	Product Dependencies .....	234
27.6	Functional Description .....	235
27.7	Serial Peripheral Interface (SPI) User Interface .....	244
<b>28</b>	<b><i>Two-wire Interface (TWI)</i></b> .....	<b>257</b>
28.1	Overview .....	257
28.2	Block Diagram .....	257
28.3	Application Block Diagram .....	257
28.4	Product Dependencies .....	258
28.5	Functional Description .....	259
28.6	Two-wire Interface (TWI) User Interface .....	264
<b>29</b>	<b><i>Universal Synchronous Asynchronous Receiver Transceiver (USART)</i></b> .....	<b>273</b>
29.1	Block Diagram .....	274
29.2	Application Block Diagram .....	275
29.3	I/O Lines Description .....	275
29.4	Product Dependencies .....	276
29.5	Functional Description .....	277
29.6	USART User Interface .....	300
<b>30</b>	<b><i>Synchronous Serial Controller (SSC)</i></b> .....	<b>317</b>



30.1	Overview .....	317
30.2	Block Diagram .....	318
30.3	Application Block Diagram .....	318
30.4	Pin Name List .....	319
30.5	Product Dependencies .....	319
30.6	Functional Description .....	319
30.7	SSC Application Examples .....	332
30.8	Synchronous Serial Controller (SSC) User Interface .....	334
<b>31</b>	<b><i>Timer/Counter (TC)</i></b> .....	<b>357</b>
31.1	Overview .....	357
31.2	Block Diagram .....	358
31.3	Pin Name List .....	359
31.4	Product Dependencies .....	359
31.5	Functional Description .....	360
31.6	Timer/Counter (TC) User Interface .....	373
<b>32</b>	<b><i>Pulse Width Modulation Controller (PWM)</i></b> .....	<b>391</b>
32.1	Overview .....	391
32.2	Block Diagram .....	391
32.3	I/O Lines Description .....	392
32.4	Product Dependencies .....	392
32.5	Functional Description .....	392
32.6	Pulse Width Modulation (PWM) Controller User Interface .....	401
<b>33</b>	<b><i>USB Device Port (UDP)</i></b> .....	<b>411</b>
33.1	Overview .....	411
33.2	Block Diagram .....	412
33.3	Product Dependencies .....	413
33.4	Typical Connection .....	414
33.5	Functional Description .....	415
33.6	USB Device Port (UDP) User Interface .....	429
<b>34</b>	<b><i>MultiMedia Card Interface (MCI)</i></b> .....	<b>447</b>
34.1	Overview .....	447
34.2	Block Diagram .....	448
34.3	Application Block Diagram .....	448
34.4	Pin Name List .....	449
34.5	Product Dependencies .....	449

34.6	Bus Topology .....	449
34.7	MultiMedia Card Operations .....	451
34.8	SD Card Operations .....	459
34.9	MultiMedia Card Interface (MCI) User Interface .....	460
<b>35</b>	<b><i>Analog-to-Digital Converter (ADC) .....</i></b>	<b>475</b>
35.1	Overview .....	475
35.2	Block Diagram .....	475
35.3	Signal Description .....	476
35.4	Product Dependencies .....	476
35.5	Functional Description .....	477
35.6	Analog-to-digital Converter (ADC) User Interface .....	482
<b>36</b>	<b><i>Controller Area Network (CAN) .....</i></b>	<b>493</b>
36.1	Overview .....	493
36.2	Block Diagram .....	494
36.3	Application Block Diagram .....	494
36.4	I/O Lines Description .....	495
36.5	Product Dependencies .....	495
36.6	CAN Controller Features .....	496
36.7	Functional Description .....	508
36.8	Controller Area Network (CAN) User Interface .....	521
<b>37</b>	<b><i>AT91SAM7A3 Electrical Characteristics .....</i></b>	<b>549</b>
37.1	Absolute Maximum Ratings .....	549
37.2	DC Characteristics .....	549
37.3	Power Consumption .....	551
37.4	Crystal Oscillator Characteristics .....	553
37.5	PLL Characteristics .....	556
37.6	USB Transceiver Characteristics .....	557
37.7	Analog-to-Digital Converter Characteristics .....	559
37.8	AC Characteristics .....	560
<b>38</b>	<b><i>AT91SAM7A3 Mechanical Characteristics .....</i></b>	<b>566</b>
38.1	Thermal Considerations .....	566
38.2	Package Drawing .....	567
38.3	Soldering Profile .....	569
<b>39</b>	<b><i>AT91SAM7A3 Ordering Information .....</i></b>	<b>570</b>



<b>40</b>	<b><i>Errata</i></b> .....	<b>571</b>
40.1	Marking .....	571
40.2	AT91SAM7A3 Errata - Rev. A Parts .....	572
	.....	<i>i</i>
	<b><i>Table of Contents</i></b> .....	<i>i</i>
	<b><i>Revision History</i></b> .....	<i>ix</i>



## Revision History

Version	Comments	Change Request Ref.
6042A	23-Dec-2004 First issue	
6042B	30-Sep-2005	
	In "Features" , corrected number of battery backup registers in Features. Updated details of Page Programming Time.	05-231
	Changed signal names VDDIN and VDDIO to VDD3V3 and VDDOUT and VDDCORE to VDD1V8. Removed signal names GNDANA, GNDBU and GNDPLL in <a href="#">Figure 2-1, "AT91SAM7A3 Block Diagram"</a> , <a href="#">Table 3-1, "Signal Description"</a> and <a href="#">Table 4-1, "Pinout in 100-lead LQFP Package"</a> .	
	Updated <a href="#">Section 5.1, "Power Supplies"</a> with new signal names and new information.	
	Updated <a href="#">Figure 5-1, "3.3V System Single Power Supply Schematics"</a> with new signal names.	
	Updated <a href="#">Section 8.1 "Embedded Memories" on page 15</a> with correct lock region size.	05-440
	Updated <a href="#">Section 8.3, "Embedded Flash"</a> with information on standby mode for Flash.	
	Updated <a href="#">Figure 9-1, "System Controller Block Diagram"</a> with new signal names.	
	Corrected start and end addresses of GPBR in <a href="#">Figure 9-2, " System Controller Mapping"</a> ;	05-233
	Updated chip ID reference in <a href="#">Section 9.6, "Debug Unit"</a> .	
	Updated <a href="#">Figure 18-1, "Shutdown Controller Block Diagram"</a> to remove RTC.	05-230
	Updated SHDW_MR register with RTTWKEN bit and definition.	
	Updated formula in definition of CPTWKx pin in SHDW_MR register.	05-235
	Updated SHDW_SR register with RTTWK bit and definition	05-016
	Updated information on FMCN bit: removed note, changed bit description in MC Flash Mode Register .	05-380
	Added external interrupt source type to SRCTYPE bit in AIC_SMRx register.	
	Section Debug Unit updated to remove all references to FORCE_NRST.	
	Updated bit descriptions for SRAMSIZ and ARCH in Debug Unit Chip ID Register.	05-115, 05-306, 05-307
	Pull-up resistor value changed to 10 Ohm in <a href="#">Section 26.4.1, "Pull-up Resistor Control"</a> .	
	IADRSIZ[9:8] bit description changed in TWI Master Mode Register	05-057
	IADR bit description changed in TWI Internal Address Register	05-060
	TXCOMP bit description changed in TWI Status Register	05-062
	Corrected error in text of <a href="#">Section 29.5.3.5, "Parity"</a> , in USART section.	05-067
	Corrected error in <a href="#">Figure 29-17, "Receiver Behavior when Operating with Hardware Handshaking"</a> in USART section.	04-467
	Corrected error in <a href="#">Figure 29-23, "IrDA Modulation"</a> in USART section.	
	Clock gating feature added in Receive and Transmit Clock Mode Registers in SSC section.	
	In section "Synchronous Serial Controller", compare functions added.	
	PWM, added note on polarity and alignment in <a href="#">Section 32.5.3.3, "Changing the Duty Cycle or the Period"</a> .	

Version	Comments	Change Request Ref.
6042B	Added <a href="#">Section 32.5.3.4, "Interrupts"</a> in PWM section.	
	Added warning regarding UDP peripheral clock in PMC and UDP_TXCV register to <a href="#">"Power Management"</a> , <a href="#">"USB Device Port (UDP) User Interface"</a> and <a href="#">"UDP Transceiver Control Register"</a>	05-413
	Corrected Reset State for UDP_TXVC register.	05-100
	Changed register names from USB_xx to UDP_xx.	05-147
	Corrected bit name to read ENDBURSES in <a href="#">"UDP Interrupt Clear Register"</a> .	05-148
	Added missing bit EP5INT in <a href="#">Section 33.6.9, "UDP Reset Endpoint Register"</a>	
	Corrected pin names in <a href="#">Figure 34-4, "MMC Bus Connections (One Slot)"</a> ,	05-194
	Corrected formula for entering power saving mode in MultiMedia Card Interface (MCI).	04-489
	Changed GOVRE bit description in <a href="#">Section 35.6.6, "ADC Status Register"</a> i	05-045
	Corrected typo in LPM bit description in <a href="#">Section 36.8.1, "CAN Mode Register"</a> .	05-206
	Corrected and updated <a href="#">Section 36.6.4.1, "CAN Bit Timing Configuration"</a> ; corrected <a href="#">Figure 36-5, "CAN Bit Timing"</a> ; updated <a href="#">"Example of bit timing determination for CAN baudrate of 500 Kbit/s:"</a> , added <a href="#">Figure 36-6, "CAN Resynchronization"</a> .	05-336, 05-420
	Corrected and updated <a href="#">Section 36.8.6, "CAN Baudrate Register"</a>	05-420
	Updated <a href="#">Section 37., "AT91SAM7A3 Electrical Characteristics"</a> , <a href="#">Section 38., "AT91SAM7A3 Mechanical Characteristics"</a> and <a href="#">Section 39., "AT91SAM7A3 Ordering Information"</a> .	
<b>6042C</b>	<b>21-Nov-2005</b>	
	<a href="#">Section 40., "Errata"</a> added to datasheet. Replaces AT91SAM7A3 Errata sheet Atmel °6145	05-524
	Evolution in IP blocks: PMC, <a href="#">Section 24.7, "Programming Sequence"</a> change to some text and code. SPI, change to <a href="#">Figure 27-1, "Block Diagram"</a> , changes in <a href="#">Section 27.7, "Serial Peripheral Interface (SPI) User Interface"</a>	05-525
	UDP, warning added to <a href="#">Section 33.3.2, "Power Management"</a> , <a href="#">Section 33.6, "USB Device Port (UDP) User Interface"</a> and <a href="#">Section 33.6.12, "UDP Transceiver Control Register"</a>	
	Change to Register bit-field descriptions: <a href="#">"SVMST0: Saved PDC Abort Source"</a> and <a href="#">"SVMST1: Saved ARM7TDMI Abort Source"</a> in the <a href="#">"MC Abort Status Register"</a>	05-527
	PLL conditions changed in <a href="#">Table 37-12, "Phase Lock Loop Characteristics"</a>	05-528
	In Features and global: <a href="#">"EmbeddedICE™ In-circuit Emulation, Debug Communication Channel Support"</a> EmbeddedICE to replace usage of <a href="#">"embedded in-circuit emulator"</a>	SJO Legal
<b>6042D</b>	<b>04-Apr-2006</b>	
	Errata added: <a href="#">Section 40.2.6.4, "SPI: SPCK Behavior in Master Mode"</a> , <a href="#">Section 40.2.8.2, "TWI: Possible Receive Holding Register Corruption"</a> , <a href="#">Section 40.2.8.3, "TWI: NACK Status Bit Lost"</a> and <a href="#">Section 40.2.8.4, "TWI: Disabling Does not Operate Correctly"</a>	#2454
	Reference to SRCTYPE field corrected in <a href="#">Section 22.7.3.1, "Priority Controller"</a> in AIC section.	#2512

Version 6042E	Comments	Change Request Ref.
	Section 4.1, "100-lead LQFP Package Outline" replaces Mechanical Overview, Figure 4-1, "100-lead LQFP Outline (Top View)" replaced.	3180
	Peripheral and System Controller Memory Maps consolidated in Figure 8-1 on page 16.	
	Table 11-1, "Peripheral Identifiers," on page 29, SYSIRQ changed to SYSC.	
	Table 3-1, "Signal Description," on page 5, SHDW and FWUP, comments updated	rfo
	Figure 8-2, "Internal Memory Mapping" reference to boot memory removed.	
	Debug and Test: Section 13.6.3, "Debug Unit" Debug Unit Chip ID value is 0x260A0941	rfo
	RSTC: Section 14.3.1, "Reset Controller Overview", updated with crystal oscillator effect on startup counter.	3005
	RTT: Added note to Section 15.3, "Functional Description"	2522
	WDT: WV changed to WDV in Section 17.2, "Block Diagram" and in Section 17.3, "Functional Description", and the 6th and 7th paragraph rewritten on page 78 beginning with "To prevent a software deadlock"	3002
	PDC: Section 21.1, "Overview", user interface description updated.	
	AIC: "Register Mapping" on page 140, footnote <sup>(2)</sup> added in reference to PID2...PID31 bit fields.	2548
	PMC: Section 23.3.1, "Main Oscillator Connections", updated.	3282
	Added a note defining PIDx to Section 24.9.4, "PMC Peripheral Clock Enable Register", Section 24.9.5, "PMC Peripheral Clock Disable Register" and Section 24.9.6, "PMC Peripheral Clock Status Register"	2468
	DBGU: Section 25.5.10, "Debug Unit Chip ID Register" update to the table in "ARCH: Architecture Identifier" on page 203.	1744/05-459
	Figure 25-1, "Debug Unit Functional Block Diagram", "ice_nreset" signal name replaced by pad name "Power-on Reset".	2832
	PIO: Figure 26-3, "I/O Line Control Logic" updated,	3053
	SPI: Figure 27-9, "Slave Mode Functional Block Diagram", FLOAD removed.	1542
	Section 27.6.3, "Master Mode Operations", update to SPI_RDR information	1543
	Section 27.7.1, "SPI Control Register", added information to bit description "SWRST: SPI Software Reset" on page 245.	
	Section 27.7.9, "SPI Chip Select Register", corrected equation in "DLYBCT: Delay Between Consecutive Transfers" on page 256.	
	Section 27.6.3.8, "Mode Fault Detection", updated.	1676
	TWI: Section 28.6, "Two-wire Interface (TWI) User Interface" UNRE and OVRE bit fields removed from TWI Status and Interrupt register tables.	2470

Version 6042E	Comments	Change Request Ref.
	<p>Section 29., "Universal Synchronous Asynchronous Receiver Transceiver (USART)" extensive update since last version, should be considered carefully. Updates include the following:</p> <p>Section 29.5.4.1, "ISO7816 Mode Overview", Clarification of PAR configuration.</p> <p>Section 29.4.1, "I/O Lines", text concerning TXD line added.</p> <p>Table 29-3, "Binary and Decimal Values for Di" and Table 29-4, "Binary and Decimal Values for Fi," on page 280, DI and Fi properly referenced in titles.</p> <p>Figure 29-24, "IrDA Demodulator Operations", figure modified.</p> <p>Table 29-2, "Baud Rate Example (OVER = 0)", last two lines of table removed.</p> <p>Section 29.5.3.2, "Asynchronous Receiver", change to second sentence in fourth paragraph "For the synchronization mechanism...."</p> <p>Section 29.5.3.8, "Receiver Time-out", list of user options rewritten.</p> <p>Section 29.6.1, "USART Control Register" STTTO bit function related to TIMEOUT in US_CSR register.</p> <p>Section 29.6.6, "USART Channel Status Register" TIMEOUT bit function related to STTTO in US_CR register</p>	<p>1552 rfo review</p> <p>2942 3023</p>
	<p>TC: added Table 31-1, "Timer Counter Clock Assignment," on page 357.</p> <p>Section 31.5.4, "External Event/Trigger Conditions" "...(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.</p> <p>The note<sup>(1)</sup> attached to the register bit description "EEVT: External Event Selection" on page 380 further clarifies this condition.</p> <p>Figure 31-2, "Clock Chaining Selection", added to demonstrate clock chaining.</p>	<p>2470 2704</p> <p>3342</p>
	<p>PWM: Section 32.5.3.3, "Changing the Duty Cycle or the Period", Update to info on waveform generation.</p>	<p>1677</p>
	<p>Section 33., "USB Device Port (UDP)", extensive update since last version, should be considered carefully. Updates include, but are not limited to the following:</p> <p>Figure 33-2 on page 414 updated.</p> <p>Section 33.4.1, "USB Device Transceiver" and Section 33.4.2, "VBUS Monitoring" added.</p> <p>Section 33.5.1.2, "USB Bus Transactions" updated.</p> <p>Section 33.5.1.3, "USB Transfer Event Definitions", updated with endpoint information.</p> <p>Section 33.5.2.2, "Data IN Transaction" endpoint use updated.</p> <p>Section 33.5.2, "Handling Transactions with USB V2.0 Device Peripheral", section reworked, warnings added.</p> <p>Section 33.5.3, "Controlling Device States" section reworked.</p> <p>Section 33.5.3.1, "Not Powered State" and Section 33.5.3.2, "Entering Attached State" have been added.</p> <p>Section 33.6.2, "UDP Global State Register", updated.</p> <p>Section 33.6.10, "UDP Endpoint Control and Status Register", updated.</p>	<p>rfo rework + 2259,2542, 2544, 2826, 2905, 3014, 3048, 3055,</p>
	<p>Section 34., "MultiMedia Card Interface (MCI)"</p> <p>Figure 34-8, "Read Functional Flow Diagram", update to flow chart.</p> <p>Figure 34-9, "Write Functional Flow Diagram", update to flow chart.</p> <p>Figure 34-10, "Multiple Write Functional Flow Diagram", added flow chart.</p> <p>Section 34.9.10, "MCI Status Register", update to descriptions of BLKE and NOTBUSY flags.</p> <p>Section 34.9.10, "MCI Status Register", added reset information on DCRCE and DTOE bits.</p>	<p>2462</p> <p>2593</p>
	<p>ADC: editor use of variables in register tables added,</p> <p>Section 35.5.7, "ADC Timings" In the Warning: "See ADC Characteristics...", typo fixed</p> <p>In the Block Diagram, Figure 35-1 on page 475, dedicated and I/O line multiplexed analog inputs differentiated.</p>	<p>1749 2830 3052</p>



Version 6042E	Comments	Change Request Ref.
	<p>CAN: <a href="#">Section 36.6.2.1 "Message Acceptance Procedure" on page 496</a> update to message acceptance example</p> <p><a href="#">Section 36.8.17, "CAN Message Data Low Register"</a> updated MDL bit description, and <a href="#">Section 36.8.18, "CAN Message Data High Register"</a> New information on byte priority added to both registers.</p> <p>Update to specify allowed values for BRP field on <a href="#">page 501</a> and <a href="#">Section 36.8.6, "CAN Baudrate Register"</a></p>	<p>2295/2296</p> <p>2467</p> <p>2597</p>
	<p>Electrical Characteristics: <a href="#">Figure 37-2, "DC Characteristics"</a>, ERASE pin removed from table.</p> <p><a href="#">Table 37-6, "Typical Power Consumption by Peripheral in Active Mode"</a>, Consumption values updated, ARM7 and Syst Peripherals added to table, ADC removed.</p> <p><a href="#">Table 37-8, "Main Oscillator Characteristics"</a>, updated.</p> <p><a href="#">Table 37-9, "Crystal Characteristics"</a>, added.</p> <p><a href="#">Table 37-10, "XIN Clock Electrical Characteristics"</a>, updated.</p> <p><a href="#">Table 37-21, "SPI Timings"</a>, "Min" updated for SPI<sub>0</sub> and SPI<sub>1</sub> Note 2 added to table, text added after table.</p>	<p>rfo</p>
	<p><a href="#">Section 40. "Errata" on page 571</a> added <a href="#">Section 40.1, "Marking"</a></p>	<p>3180</p>
	<p><a href="#">Section 40.2.2.1, "PIO: Leakage on PB14 - PB29"</a>, the leakage can be 25 <math>\mu</math>A in worst case....</p> <p>The following errata have been added.</p> <p><a href="#">Section 40.2.1.1, "MCI: Data Timeout Error Flag"</a></p> <p><a href="#">Section 40.2.1.2, "MCI: STREAM Command Not Supported"</a></p> <p><a href="#">Section 40.2.1.3, "MCI: Data FIFO and Status Bits"</a></p> <p><a href="#">Section 40.2.1.4, "MCI: DATA FIFO Problem with PDC"</a></p> <p><a href="#">Section 40.2.5.1, "RTT: Possible Event Loss when Reading RTT_SR"</a></p> <p><a href="#">Section 40.2.8, "Two-wire Interface (TWI)"</a> Behavior of OVRE bit removed from TWI errata. (2470)</p> <p><a href="#">Section 40.2.8.5, "TWI: Software Reset"</a></p> <p><a href="#">Section 40.2.9.1, "USART: Hardware Handshaking – Two Characters Sent"</a></p> <p><a href="#">Section 40.2.9.2, "USART: XOFF Character Bad Behavior"</a></p> <p><a href="#">Section 40.2.6.5, "SPI: Chip Select and Fixed Mode"</a></p> <p><a href="#">Section 40.2.6.6, "SPI: Baudrate Set to 1"</a></p> <p><a href="#">Section 40.2.6.7, "SPI: Disable Issue"</a></p> <p><a href="#">Section 40.2.6.8, "SPI: Disable In Slave Mode"</a></p> <p><a href="#">Section 40.2.6.9, "SPI: Software Reset and SPIEN Bit"</a></p> <p><a href="#">Section 40.2.6.10, "SPI: CSAAT = 1 and Delay"</a></p>	<p>rfo</p> <p>2870/2873</p>



## Atmel Corporation

2325 Orchard Parkway  
San Jose, CA 95131, USA  
Tel: 1(408) 441-0311  
Fax: 1(408) 487-2600

## Regional Headquarters

### Europe

Atmel Sarl  
Route des Arsenalux 41  
Case Postale 80  
CH-1705 Fribourg  
Switzerland  
Tel: (41) 26-426-5555  
Fax: (41) 26-426-5500

### Asia

Room 1219  
Chinachem Golden Plaza  
77 Mody Road Tsimshatsui  
East Kowloon  
Hong Kong  
Tel: (852) 2721-9778  
Fax: (852) 2722-1369

### Japan

9F, Tonetsu Shinkawa Bldg.  
1-24-8 Shinkawa  
Chuo-ku, Tokyo 104-0033  
Japan  
Tel: (81) 3-3523-3551  
Fax: (81) 3-3523-7581

## Atmel Operations

### Memory

2325 Orchard Parkway  
San Jose, CA 95131, USA  
Tel: 1(408) 441-0311  
Fax: 1(408) 436-4314

### Microcontrollers

2325 Orchard Parkway  
San Jose, CA 95131, USA  
Tel: 1(408) 441-0311  
Fax: 1(408) 436-4314

### La Chantrerie

BP 70602  
44306 Nantes Cedex 3, France  
Tel: (33) 2-40-18-18-18  
Fax: (33) 2-40-18-19-60

### ASIC/ASSP/Smart Cards

Zone Industrielle  
13106 Rousset Cedex, France  
Tel: (33) 4-42-53-60-00  
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.  
Colorado Springs, CO 80906, USA  
Tel: 1(719) 576-3300  
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park  
Maxwell Building  
East Kilbride G75 0QR, Scotland  
Tel: (44) 1355-803-000  
Fax: (44) 1355-242-743

### RF/Automotive

Theresienstrasse 2  
Postfach 3535  
74025 Heilbronn, Germany  
Tel: (49) 71-31-67-0  
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.  
Colorado Springs, CO 80906, USA  
Tel: 1(719) 576-3300  
Fax: 1(719) 540-1759

### Biometrics

Avenue de Rochepleine  
BP 123  
38521 Saint-Egreve Cedex, France  
Tel: (33) 4-76-58-47-50  
Fax: (33) 4-76-58-47-60



## Literature Requests

[www.atmel.com/literature](http://www.atmel.com/literature)

**Disclaimer:** The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. **EXCEPT AS SET FORTH IN ATMEL'S TERMS AND CONDITIONS OF SALE LOCATED ON ATMEL'S WEB SITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

© 2006 Atmel Corporation. All rights reserved. Atmel®, logo and combinations thereof, Everywhere You Are®, DataFlash® and others are the registered trademarks or trademarks of Atmel Corporation or its subsidiaries. ARM®, the ARM Powered® logo and others are registered trademarks or trademarks of ARM Ltd. Other terms and product names may be the trademarks of others.